CS 6476 - Computer Vision

1A-L1 Introduction

1 - Taking over for Aaron

Hello and welcome to the Computer Vision class. I'm Irfan Essa and I'm a professor here in the School of Interactive Computing and the College of Computing in Georgia Tech. This is a class on computer vision. It's aimed at covering the foundational aspects of how to analyze images and to extract content from images. That is, how can we build a computer or a machine that can see and interpret an image. First what do I mean by foundational? I mean that we are going to cover the mathematical and computational methods to provide you with core concepts of how can a computer be built to interpret images. Notice I am using the word interpret. In Computer Vision we are interested in extracting information, knowledge from an image. Many want to go beyond processing an image to really knowing what is inside the image, what's the content of the image. So we will learn the math and the basic concepts how to compute with an image and extract information from it. So this class will have lectures, of course what is a class without without a lecture? All the video lectures are by my friend and colleague Aaron Balik. And I'm biased but I've known Aaron for a long time and I can tell you he's a great lecturer. He actually and seriously thinks he's funny but I guess I will let you be the judge of that. I've known Aaron for over 20 years now and we've worked together. I started at MIT and I learn computer vision early days at MIT also. And then we were both here at Georgia Tech for a while, until he decided recently to become Dean of Engineering at the Washington University in St Louis. So bidding him farewell, and since I really do love this material and Computer Vision, I've decided to take over this class from now on. So the video lectures are still going to be by Aaron, but I will now manage an able team of TA's and course developers over the term and will work with them to provide additional material like the assignments and exams for this term. I'll also be the one you will hear from on various communication channels for this class. It's now back quickly to a bit more about Computer Vision. Computer Vision is really about analyzing images and videos to extract knowledge from them. Mostly these images are of real scenes like that of a street image with cars and such where autonomous vehicle they'll have to navigate through or it could be other types of images like that of an X-ray inside a human head and we need to do image analysis to be able to extract things about of interest in medical applications. So essentially the goal is image and video understanding which means labeling interesting things in an image and also tracking them as they move. In this class you will learn about the foundations of how to build systems that can do this kind of image understanding. In addition to math and the foundational theoretical context, you will also learn about doing a series of assignments to give you real hands on experience with computer vision. I look forward to seeing all of you online and engage with you as you learn this exciting material.

2 - Difference between CV and CP

As some of you know, I also teach the computational photography class. One question always comes up. What is the difference between these two classes and the material covered in it? There is indeed some overlap between the classes, especially in the initial few modules where we learn about computing with images and extracting information from images. Computational photography is really about capturing a light from a scene to record a scene into a photograph or such other related novel artifact that showcases the scene. Image analysis is done to support the capture and display of the scene in novel ways. Some of it's actually about building newer forms of cameras and softwares to facilitate that process. Computer vision is really about interpreting an analysis of the scene. That is what is the content of the image of the scene, who is in there, what is in the image and what is happening. Needless to say, there is overlap and one can benefit from the other. I also

enjoy both of them and I do work in research and education in both these topics because they're about building intelligence systems and about actually processing and analyzing images and photographs. I'm excited about it, I look forward to engaging with all of you on either, or both, of these topics

3 - Introduction to Computer Vision Course

Hi. Welcome to the Introduction to Computer Vision. I'm Professor Aaron Bobick here at Georgia Tech. I'm thrilled to be able to teach this course in this medium. I've been teaching this for quite a while. I've been a professor for, oh my gosh, almost 25 years in computer vision. Spent some time at the MIT Media Lab, and I've been at Georgia Tech for the last 15 years. My own research has been predominantly in activity recognition. And now it's moved a little more to robotics in terms of vision for robots understanding people. But the basics of this course is, is to cover sort of the fundamentals of computer vision, to give you a background enough that you could just enough to be dangerous. So before we tell you more about the course, it's important that you meet a couple of other people who are responsible for the success of this course. And it will be a success or I'll be really, really annoyed and, and then people, see everybody's going to see this and they'll think really poorly of me. Anyway, the first person you have to meet is probably the most important person in making this a success, no pressure, is Dr. Arpin Chuckarvarde who is our course developer. Come on over, Arpin. So tell the good folks at home or sitting at their desks at work a little something about yourself. Sure I got my PhD from North Carolina State University in biologically inspired computer vision. What is, what is that? Oh you take human vision systems and animal vision systems. Study them, how they function and use those ideas in an artificial setting. Cool. And what made you want to do this course development work. Well I heard that Udacity is making this online computer vision course with Georgia Tech and I thought, this is a good way to use the knowledge I've put together over the years. Okay, great, well thanks, and you guys will be hearing more from Arpin as the time goes on. So the other really important person you have to meet is Megan Smith. Megan is our video guru. She is the film person, the editing person, the production person. She makes my hair look non-existent and yeah, just makes everything look great. She also allows me to illustrate the difference between the illusion of power and actual power. See, the illusion of power is I can call Megan over and she'll probably come over and say hello. Megan, come on over and say hello. Say hello to everybody. Hello. The reason I say that that's the illusion of power. The person who has the real power is Megan because she does the video editing. So, if she wanted to, she could make me look incredibly foolish. But, you would never do that, would you? [SOUND]. Never. Uh-huh. Okay, well you'll be hearing from her sometimes when we ask some questions and she makes some interesting noises in the background. But I just wanted you to put a, a face with the beautiful melodic voice that you'll be hearing later. So thanks, Megan. All right, so that's the crew. And I think this is going to be a good time. I mean it's maybe, maybe, okay. Maybe that's a little bit much. I think you'll enjoy the class. There is a lot to be learned. We can't actually cover all of computer vision, but we can sort of go over a whole bunch of the fundamentals and I think you'll learn a lot if you actually do the work. And it is a bunch of work, I don't want to, I don't want to pretend otherwise. But if you do it I think you'll come away with the ability to do things that you couldn't do before.

4 - What is Computer Vision

All right. So let's begin our trip down computer vision lane here. So the first question you might ask is, what is computer vision? What is computer vision? See, I told you you were going to hear from Megan. Well, there's a couple of ways of thinking about it. I like this slide that I borrowed from Steve Seitz, where he talks about every picture tells a story. And one way of thinking about computer vision is the goal of computer vision is to interpret images. That is, say something about what's present in the scene or what's actually going on. So, what we're doing is, we're going to take images in, and what's going to come out is something that has some meaning to it. That is, we're going to extract, we're going to create some sort of interpretations, some sort of an understanding of

what that image is representative of. This is different, many of you may have some exposure to image processing, which is the manipulation of images. That's images in and images out. And we'll talk a little bit about that because you use image processing for per, for computer vision. But fundamentally computer vision is about understanding something that's in the image.

5 - Identify Objects Quiz

So as a thought experiment and actually as one of these arpin quizzes. That's what we're going to call these things, arpin quizzes. Here's a scene. So I want you to just take a look at that scene. And then you can stare at it as long as you want. Okay, you got it? And here's what I want you to do. Type in the little text box there as many items that you could label or identify as you want. Make sure when you do you type it with a comma separated list. That way the machine can understand what's going on.

6 - Identify Objects Solution

So, here you can see a list of items that were put in by somebody, not all that imaginative. No, it wasn't Art. It was somebody else. Really. You know, sofa, table, chair, things that were in the room. But notice, those are specific things. You could have said, wall, floor. You might've said that's a living room, that's probably a house. That's a modern house. Or you could have said that's the house of an upper-middle class family taken back in 1997 somewhere near Ohio. How you would know that I don't know, but maybe you know something about interior decorating.

7 - Recognize Action Quiz

That was cool. Picture, tell me what's in the picture. But those labels, you notice, those were all about sort of, static objects that were present in a single image. These days, especially with sort of the ubiquity of media, computer vision is not limited to just processing a single image. In fact, we can process a whole bunch of images such as a video. And video allows us to start talking about thinks like action. So to illustrate that, let's call back on Arpan again. Arpan, come on over. All right, Arpan, do your thing. So, here's a quick quiz, what did Arpan just do?

8 - Recognize Action Solution

Well, what Arpan just did is he flapped his arms like a bird. And not such a great bird, but a bird. Okay? And the question you might ask is, how would I get a computer to do that? How would I get a computer to do that? Well the good news is, at the very end of this class. We'll give you at least a few ways that you could actually think abut a computer processing video, that'll be able to recognize simple actions done by a person

9 - Why Study Computer Vision

If that’s computer is, vision is why would you want to study this? Well, I mean, a, you might want to get a Masters in Computer Science degree at Georgia Tech, and everything else was filled. And you got stuck with Bob, and well, so here you are. But there’s actually some really good reasons to do that. These days images and imagery have become ubiquitous in all of our technology. cameras, video, you can stream them, you can send them etc. So what's become fundamental to an awful lot of systems is the manipulation in the processing of imagery. And extracting information from that. There are domains such as surveillance. There are building 3D models for medical imaging. Or capturing for motion capture. These are all different current industries that leverage computer vision in a variety of ways. But most of all, the reason to do it is, it is just a really cool and deep set of problems. And it's way more fun that learning how to build compilers. And now, I have to go apologize to all my compiler friends, but they know it's true.

10 - OCR and Face Recognition

Interesting question is sort of what is the state of the art in computer vision now? What are things that are people doing with computer vision? How might that compare a little bit to the way humans do vision? Here is a, a simple example of stuff that, fact used to be sort of considered to be difficult, but is actually now pretty standard. So let's talk about simple optical character recognition. So here's an example from some license plate readers. And license plates are somewhat easier because there's a fixed font. In fact, not that long ago doing OCR was considered very hard. Today if you have a scanner, or if you have Adobe Acrobat, it comes with OCR built in. Because that's how ubiquitous and sort of easy it is. A little more challenging, many of you may have started using automated teller machines. Where you can deposit bank checks with hand written numbers that are the amount. And also, for quite a while the Post Office has been recognizing the ZIP codes using, machines. Again, on handwritten envelopes. So that's an example of computer vision extracting the meaning. What are the numbers that are there? Another thing that's very common these days is face detection. Just about any digital camera that you buy today, you pick it up, using the default setting, it will find the faces. So, here's an example. One of the cool things, by the way, is bunch of lectures in the future, we'll talk about the technology that basically does exactly this. So the next time, so when we get to those, you pick up a camera and it finds the faces. You'll say, oh, I know how they're doing that. But actually now, cameras can do more. I think the one on the left, I think is from the web, from Fiji, that if you take a picture of somebody, and they blink. You know, and that can be really annoying, it'll tell you they blinked. Maybe even more interestingly, Sony has something called the Smile Shutter, which will watch for people. And you sort of press it, and say, take a picture now. But actually it waits until you, it sees the person smile. And even further these days, there are cameras that will recognize who you are. So this is a screen taken from a shot where it does camera based login. So it knows about a bunch of different people. You walk up to the computer, you say, yo, computer it's me. Actually you don't have to say anything, and the computer says, it's you. And it logs you in. So that's face recognition. We're going to talk also, a little bit, about face recognition. Although the face detection stuff techno, that technology is one that will be sort of more fundamental to the class.

11 - Object Recognition

In sort of a related way, there's a lot of technology these days involving object recognition. So there was this company. There is this company, Evolution Robotics, that had developed this thing called Lane Hawk, which basically prevented some of you. None of you. From putting stuff on the bottom of a basket and then wheeling it out and forgetting to mention to the cashier that it was there. This is actually a huge problem. And the system, you can see the camera here. Right, see, that's a camera down here, looking at this. I wonder if we're going to have to, like, erase that that's a beer. Yeah, you see, now nobody can see. Okay, you can't tell what that is. Okay. And it can detect what that is, which is not only pretty cool. But if you go on their website they'll tell you that five years later, that product was bought by a different company. So, there's money to be made in computer vision. Go do a really cool startup. Object recognition used to require a lot of computing power. It's, the computing power has gotten smaller. And it allows us to now operate in smaller packages. So, there's this whole area of augmented reality and object recognition by mobile devices. So, here's a system of where you, you're showing it a picture of this statue. It recognizes who this is and what the monument is. And here's an old picture from Nokia, where you can actually go off to the web, pull out information and display it to you. So for a while we were talking about doing this on smart phones, now it sits on your face. This is Professor Thad Starner, a Georgia Tech professor as well. He was instrumental in the developement of Google Glass. And one of the things that Glass does is you've got a camera looking out of what you're seeing. And can, through the same object recognition methods, can give you information about what you're looking at. And this is also part of computer vision.

12 - Special Effects and 3D Modeling

There's a area of computer vision people know a little bit less about. It's used a lot in special effects, everything from capturing the shape of somebody, so you take the scan of somebody's face, whether it's laser or otherwise. You build models, and then you can make lots of these people, and you can light them from different sides and different directions because you have a full 3D model. Likewise, motion capture, so if you saw Pirates of the Caribbean, the one with the, the guy with all the weird things on his face, and of course, you know, that's all CGI, but the question is, how do they know exactly where to put his face and everything? Well, that, there are these markers that are being worn that are being tracked by these cameras. And they have to figure out the three-dimensional geometry, and that's also a form of computer vision. Another area that's become but this is a shot from Google Earth actually, this is from Microsoft's Virtual Earth. Google Earth is yet another version of it. Where basically, they can take imagery, so here's imagery, aerial imagery. But also, they can use that to figure out the models of the buildings. Put those three-dimensional models in there, and then you can fly around them however you want. So that's a structure for motion method of using lots of images, a sequence, to recover the three-dimensional structure. We'll talk only a little bit about that. We'll focus mostly on a couple of images.

13 - Smart Cars

Another area that you know, really blossomed lately is the use of computer vision for automotive. This is a, a website, web picture taken from Mobileye, which is a company out of Israel. And they've developed all sorts of technologies that use computer vision that are relevant to automobiles. Everything from automatically recognizing signs to, here it's a little hard to see, red outline. The system is automatically identifying where the pedestrians are. They have a system that alerts you if pedestrians getting close and you're, seem to be going too fast. You can also build systems that either brake or slow down or whatever. But the idea is that computer vision has really gotten into smart cars. And in fact, smart cars are here. So this car, some of you know, that is Stanley. Stanley was the Stanford. That's the little red S there. Entry into the Urban Grand Challenge run by DARPA. And it was started by. Stanley was run by this guy. What's his name? Oh yeah, Thrun something, I don't know Sebastian. He's the guy who also started Udacity. He's sort of a way under achieving, no ambition kind of guy. They won that. And then Sebastian, because he's just out there, convinced Google to get involved in the making the automobile process. The self driving car, which most of you have heard about. Here's a picture of it out on the highway. And the real mark that these things are here today is. States now have starting passing legislation that helps detail, well who's at fault if an accident happens on a particular road and it's a self driving car. So this is where technology starts to hit policy and economics, and that's when you know it's real.

14 - Sports

Another big economic place for computer vision is sports. If you watch any sort of professional sports on TV, you see them leveraging computer vision sometimes in powerful ways, sometimes in simple ways but are really useful. So here, this is Sportvision first down line for American football. I say American football because sometimes when I go to Europe and I say football they think a ball is round. We know that footballs are not round. And what's interesting about this line here is this is the line that he has to cross to make a first down. Of course that's not on the field, it's drawn in there. The only interesting computer vision that's going on here is you'll notice that that line does not go through him. So the system had to separate out the player from the background, okay? And even though, by the way, kind of greenish color in there, it's able to separate the grass from the player. In fact, if you ever watch one of these games when it's rainy and it's an outdoor field and it gets all muddy, the first down line stops working so well. Because the grass is now mud and it doesn't look like grass anymore and it can't tell the players and the whole thing doesn't work so well. And you can see a computer vision failure right there.

15 - Vision Based Interaction

Something else that has really changed lately, is the pushing of computer vision down into video games. So one of the first places to do that was the Nintendo Wii. The remote control, there's actually a camera system built right into here, that tracks the two dots that are from the sensor bar and reports that information. But the real game changer in terms of computer vision being involved in games, was the Microsoft Kinect. The Microsoft Kinect is a depth sensor. And what I mean by that, is it can produce a scene like this, that is, it can, it can produce an image like this from a scene. And this is a depth image, right? So darker is farther away. And brighter is closer. And gray here is sort of in between. These white stripes, these are where the thing is shadowed. Don't worry about that. So from a technology perspective. From a raw technology perspective, most people think that what was important about the Kinect was that it created a depth image. But you will know better. No. The important thing about a Kinect, is that it can produce skeletal descriptions. Using a combination of machine learning techniques applied to computer vision, the folks at Microsoft and this came out of Microsoft Cambridge in the UK. Were able to recover the skeleton geometry of people from the depth image. And by the way, they do it instantaneously. Every frame, they do it differently. They don't even track it, they just do it one frame at a time. It's that robust. Right? And because they can get the skeleton information, you can build really cool user interfaces. So you could you know, do driving games and go like this, to steer your car. Now personally, I would get tired of that after a little while. But you know, if you're a prepubescent you know, maybe this is a lot of fun. Much more interesting of course, is playing with robots. And here you see, this is Simon, in Andrea Tomas' lab, here at Georgia Tech. And the reason that Simon is waving to this student, is that behind Simon over here, there's a Kinect that's looking at the student waving. And is getting back the depth information. Also the skeleton information is then interpreting the skeleton information, in terms of what the human is doing. And that in turn, allows the robot a decision about what the robot wants to do. So even though the Kinect was sort of promoted and invented as a way of impacting games. And early on, there was some uncertainty at Microsoft whether they wanted to open up that, system and let people use it. They quickly realized that this is something that everybody wants to use. And it has really revolutionized the way people think about depth imaging, and computer vision applied to depth imagery.

16 - Security and Medical Imaging

Finally, two more surveillance is a huge issue. This ideas of being able to monitor environment for crowd safety, a variety of reasons. These are screenshots taken from Siemens sells a system for doing port monitoring. Just to know whether, say, people are loitering, or vehicles are approaching that aren't supposed to be. This is also a computer vision. A more direct effect on a single individual with computer vision is work in medical imagining. Okay, so here you see an example. There's all sorts of 3D imaging, MRI, CAT scan, stuff like that, but here you see some work. This came out of Eric Grimson's lab a while ago at MIT where on a screen, a the computer vision system is registering the skull that's on the table with a model that has been created from the 3D imaging. So while, when the surgeon looks at this monitor, he sees the real person, and by the way, if there were a scalpel here, he'd see the hand with the scalpel or drill or whatever you're using to make a hole in the person's head. And where the various structures are underneath that you wanted to see, and that's also computer vision. Look, this is just a quick taste of the state of the art, hopefully, it inspires you. One of the things I want you to realize is that almost everything that I showed you here is less than ten years only, and in fact, many of them are less than five years old. So the field is rapidly changing, and there's a tremendous amount of opportunity in computer vision right now, so this is exactly the right time for you to take this class. So that you can invent some cool technology. Make, $3 gazillion and send a very large donation to the people who made it possible.

17 - A Novel Application Quiz

Quick! Put on your entrepreneur cap. Think of a situation, maybe from your own day-to-day life, where computer vision could be applied and maybe isn't quite yet.

18 - A Novel Application Solution

Obviously, there's no particularly right or wrong answer. With computing having gotten so small and powerful, and cameras being so easy to produce and cheap, you can start doing computer vision all sorts of places.

19 - Why is This Hard

Look. You opened your eyes sometime, you know, when you were really, really small. kind of fuzzy for a while. But after a while, you just saw. So you might wonder, you know, why is this hard? Let me give you a couple of examples. So here we have a, what you think of as a simple scene, and this was generated by Ted Adelson up at MIT. Allright? And you see on this screen, we've got this checker board here. And we've got this cylinder casting a shadow. All right? And there are light and dark squares. And you see these two squares, the A square and the B square? Which of those two squares do you think is actually darker? Okay. Well if you're like me you say, well duh, it's the A square. Okay that's, the A is dark, and B is light. Maybe not so fast. All right? So here I have a gray bar, and this gray bar is the same intensity all the way down. And through the magic of Powerpoint, we're going to slide that grey bar over, and now we have a problem. There's no edge here, there's no edge there. That's a constant grey bar, right? That means squares A and B are actually the same color on this screen. Really. In fact if I put two of those bars there, you start to be able to see it. Let me go back and forth there. Do you see that? When there is only one bar, your brain just wants you to believe that that is a checkerboard. And that it, the cast shadow, and that that's the light square. If you actually took a photometer and you measured, you would see that are as many photons coming off of this region, as off of that region. But the photometer doesn't have your brain. It doesn't understand. No, no. no, no, that's actually a light square in a cast shadow. All right? This is why computer vision is hard.

20 - Vision is NOT Image Processing

So in that example, the thing to realize is those two squares that have the same measurement of intensity. All right? So seeing is not the same thing as just measuring image properties. In fact, seeing is called, we refer to it as building a percept, a build up a description in your head of what's actually going on there, based upon what's in the measurements. That can be illustrated in a couple of ways. Here's one, so this is a very, very old stereoscope. This is a slide borrowed from Michael Black. And we'll talk about this in detail when we do stereo. And if, and the, the stereoscope here is designed so that one image goes through one eye, and the other image goes to the other eyes. So, here they, here they are. If you take a look at those two images, this is the left image, and this is the right image. Left, right, left, right. It's the difference between those two images that lets your brain realize that the dad holding the kid is in front of the bed where the mom is. Okay, and it's that difference you're building that description from the difference between those images. So I'm going to show you a video now from Dan Kersten that shows you that perception is an active construction on the part of your head. So here, there's going to be this ball rolling back and forth, okay, and the ball's going to do the same thing, but the shadows are going to do something different. Ready, watch. Whoa. Okay. So let's do that again. So you see the ball, and the shadow goes with it, and you see the ball going to the back corner of the checkerboard. And now, we move the shadow sideways, and you see the ball lifting up. The ball did the same thing in both cases. All right? But your brain, because of the shadow, says the ball did something different. So again, that’s not a property of measurement. So here’s another example. So what you’re going to see is what

looks like a green piece of paper or something on a checkerboard. You saw that green thing lift up. Right? You saw it come right off the checkerboard, right? Let's do it again, all right? Here we go, it comes up off the checkerboard, right? Just look at the top left-hand corner of that green thing, and you will see that absolutely nothing moves. None of the green pixels move at all. The only thing that moves are those darkish pixels, which your brain says is a cast shadow. So that means your brain had to pretend there was some illuminant up in the sky over here, making these shadows. And the only explanation is that the, the green thing came up off the, the, the checkerboard. So it's another example of your brain doing this construction. So the previous examples I was showing you, are in some sense the human system doing something wrong, right? The ball didn't actually change what it did. But actually, what happened there is that, there's ambiguity as to what the ball is doing, and your brain is creating the story, it is making the description, and the difference between straight image processing and computer vision is building that description. And we're going to show you some of the methods that you would use to help do that construction.

21 - Course Overview

So let me give you a little bit of a course overview. To do that, let me explain sort of how the course is thought of, at least by me, and Arpin thought this was really good, and Arpin's really smart, so if he thought this was really good, maybe it's, it's good. You can think of computer vision as being a relationship between sort of three ways of thinking about what goes on. At the top of the triangle is the computational model. Often that's the math. The, and I'll use an example from stereo. So here we're showing you sort of the, the mathematics geometry behind, if you have multiple views looking at a point out there, how you could reason about it, and how what the math would be behind trying to find its actual depth, right? The idea, you know, here's its center of one camera, the center of another camera, here is some point, possibly here, here, here, and by matching, we're going to be able to figure out what the depth is. Once we have the math, we can develop an algorithm, so here's an example of doing stereo by what's called just correlating two patches along an epipolar line. Don't worry, you're going to learn what these things are. And computing say, the sum of square differences. Just how well they match and you would measure that, and the point at the minimum, that's what this bottom point here is, that would be the right match. And by knowing that match, I would know what the depth is. So that's the algorithm. And then the algorithm when we describe it to you in class or in lecture, the algorithm always works. But of course, the last point of the triangle are real images. And when you apply these algorithms to real images, you're going to find that, oh, you know, Professor Bobic, he just lied. [LAUGH] Okay. In particular, what you're going to find is that to make these things work requires some reasonable amount of experimentation. So we'll give you images in stereo, actually, we'll give you some scenes for which there exists ground truth. That is, you actually know what the right answer is. And the question will be, how close is your result to the right answer? The answer's going to be not really close. [LAUGH] Because you're going to be implementing a relatively straightforward stereo algorithm. And the reason there are more complicated stereo algorithms out there is because the straightforward one doesn't handle all the little issues that show up in a real image. This triangle of three ways of thinking about computer vision, computational models, algorithm, and real imagery, that's really, that triangle is how the course is structured. What we'll do is, we'll develop the theory, which will explain to you why it's possible to compute these things, and then, we'll show some algorithms that implement that theory by making certain assumptions or certain ways of going about handling the images. And then, you'll actually apply them to real images, which have all of their own mysterious properties. That's going to be, will talk more about, in a minute about the problem sets. The idea of the problem sets is to get you to understand sort of the interplay between the theory, the algorithm, and the images.

22 - Topic Outline

The course itself is structured in these sort of ten overall units. Don't worry, they're not all as equally long and pedantic as this introduction. We'll start off early by doing some image processing

because you have to manipulate images to do a variety of types of interaction with them later. And we actually have a problem set early on there that goes from image processing to getting some basic structure. Then we'll talk about the geometry of cameras and camera models, and what happens when you have multiple views and how you can relate them to each other. We'll get down to the image and talk about features, computing something about one image and figuring out where in the other image those same points occur. because once you can do that, there's a lot you can do in terms of thinking about either transforming images or computing geometry between them. We'll spend just a little bit of time talking about how images get formed in the first place, what we call lightness and brightness. How does light interact with material and then come to your imaging sensor in order to make a picture? We'll spend a bit of time talking about motion, remember we talked about not just static images, but we can actually have sequences of images? So the question of how things are moving can be looked at. And these are sort of separated into motion in the image, and that, that is sort of how the pixels change, and then tracking the object. And we'll focus a bit on tracking as well. Then we'll get to something which we're going to do just a little bit of, of classification and recognition. And some of you may be saying, aw that's disappointing. No, no, no, it's not disappointing, okay? A lot of classification recognition is deeply steeped in machine learning. So what we'll do is, we'll do some of it here, sort of some basic pattern recognition, and then computer vision applied to recognition. And then if you want to do more in the recognition universe, you'll have to learn some machine learning as well. The course concludes with just some extra stuff that is just useful to know, if you're actually going to do computer vision work. And then finally, because I can't help myself and neither can Arpin, because he has true inspiration from biological systems, we'll talk a little bit about how the human vision system works.

23 - Course Details

As I mentioned, the core of the doing of this class are the problem sets. It says 8. The first one is a very simple thing just to make sure that you can get your images in, do some basic manipulation of the pixels, and submit them. It's a way of making sure the mechanics of everything are working for you. Then problem sets one through seven are actually to implement a variety of the algorithms that we talk about sometimes some simplified versions of them. Apply them to imagery that we'll give you and see what sort of results you can get. A quick thing also, we'll be using Piazza. Piazza is a social network thing for academics. I've been using this for a couple of years now in my class here at Georgia Tech. Is a great way for students to communicate about whatever's bothering them. Now what's bothering them? It's always the problem sets. Every now and then it's the lecturer but mostly it's the problem sets. And you'll see, you know, Piazza used and then all of the sudden the problem set is due and Piazza use goes like this and then it goes like that. So I encourage you strongly to use the forums. If you have a question, almost certainly somebody else has that same question too. So go ahead and post it, and somebody's going to say, man I'm glad you asked that question. Or, you may find they already asked that question, and either or I or somebody else has already answered that question. So I encourage you to actively use that forum. Whenever you have a sharing system like this, people always say, well, how much exactly are we allowed to share? And it's always a challenge, and here's what I tell students. Full blackboard or whiteboard collaboration is fine. So if you guys want to stand or talk to each other in front of a whiteboard and sketch out how the code works and what you had to do and how you had to change the parameters, that's great. What I want you to do, though, is write your own code. So, if you're posting an answer to Piazza, don't post big chunks of code, because it's just going to be too tempting for somebody to just take that. Post sort of what you had to do. If you don't write your own code, you won't understand why it is that certain things make things happen. I every now and then get a complaint from people, why is it that we're implementing things that just exist in libraries? And the real answer is, "so when it doesn't work, you'll know why." Because guess what? It often won't work in the library either or won't work as well. And then you'll have no understanding of what goes on. So, yes some of what you will do will say you're not allowed to use now that function A, B or C. We'll talk about that in a minute. And it's not just because it's good for your health right it's not just like you're eating spinach, it actually is important for you to develop some understanding as to how these algorithms

work. There will be a final exam. It is not designed to be hard. It is simply designed to force you to go back and look at the material and just for a second time get it into your head, and it also allows us to ask some questions about stuff that didn't show up on the problem sets. Normally when I give the exam here it's a three-hour slot and it takes people an hour. But it just basically covers the base material. The grading, current rubric is that 85% of the grade is based upon the problem sets, which means 15% of the grade is based upon the final, and the other 10% of the grade is based upon how I feel.

24 - Software

All right, so the last thing we have to talk about are the logistics of how you actually do the work. And essentially, there's going to be two places that you're going to be doing the doing. Well, three. First, there are these Arpin quizzes, where you just have to click on the dots or fill in the things, and usually those are just going to be making sure that you're not asleep while watching the videos. The real work's going to come in terms of doing some computer vision work. First, whenever we introduce certain image processing or computer vision types of new algorithms or methods, as often as we can, we'll produce some embedded programming exercises, which you can actually use while you're doing the videos and work within them. And in particular there, there may be a function that Matlab already implements. And so, you will play with the function in the embedded system that you can just use, so you might come to understand the different, say, the effect of the parameters. But then, when we get to the problem sets, you'll actually be writing some of these functions for your own. So let's talk about your own software environments. Most of the examples that I will describe and talk about come from using Matlab. I use Matlab normally when I teach here. I use the base Matlab, plus what's called the image processing toolbox. By the way, there is a computer vision toolbox also for Matlab. I don't make use of that in this course. It has some of the more advanced things. For what we do, we just need Matlab and the image processing toolbox. By the way, for any of you that are actual students with a student ID or academic affiliation, there is a student version of Matlab, which costs less than most computer vision textbooks. If you go to a company and you have to buy Matlab, it is a relatively significant piece of software to acquire. The fact that you as a student can get Matlab at a student discount, and by the way, the license never expires, you just get to use it, I highly recommend you go out and do that. And no, MathWorks has not paid me anything to say that. Really, honestly. Honest, I wouldn't kid you. But, if you don't want to do that, there is an open source version of something called Octave, and Octave is a open source, sort of equivalent of Matlab. And there's also image processing toolbox equivalent available for Octave. Arpin will give you a little more detail about exactly how to get that, in order to get that installed. But that's the examples, that's the sort of system that I'll be using as example. For some of you, you might rather do this using Python and OpenCV. OpenCV is something that originally came out of Intel, took a winding path through Willow Garage for a while, is now a part of, I think the Open Source Robotics something. It's moved into a not-for-profit, that's sustaining it. OpenCV has really enabled people to do a lot of image manipulation stuff in a robust way, from either Python or C++. When we give out the problem sets, if there's something special that you need to know about how to do it in Python/OpenCV versus Matlab, we'll describe both of them. And you can use either one of those programming environments. Matlab tends to make things a little easier, things like plotting and clicking on images and seeing what's going on. Python and Open CV is a little more connected to what you might actually have to do out there in the job world, to actually produce computer vision code.

25 - Matlab

If you choose to use MATLAB for this course, you will need base MATLAB along with the image processing toolbox. Head over to the MathWorks website and look for the latest release. Click Buy Online to see pricing options. We recommend getting a student edition. You can either choose the MATLAB and Simulink Student Suite, which includes the image processing toolbox, or, you can choose the base MATLAB portion, and the image processing toolbox separately. Know that you

may need MATLAB or specific toolboxes for other courses, so choose wisely. Your institution may also have a volume licensing agreement with MathWorks. So find that out before you download it for yourself. Once you have MATLAB installed, try out a few simple commands. The command window is connected to an interpreter. This is where you will spend most of your time. You can type in arithmetic expressions that are evaluated on the spot. You can also create variables. Note how it printed out the value of the variable here. If you want to suppress that, use a semicolon at the end. You can also create vectors and matrices and run operations on them. You might have noticed this workspace being updated. It lists all the variables currently in memory and the value assigned to them. There is also a file browser that lists the files in the current folder. MATLAB has built-in functionality to read images. We will use the imread command. Note that you can use tab completion to make your life easier. Tab completion also works with file names in the current directory. You can display an image using imshow. The image processing toolbox contains several useful commands for working with images. One of these is rgb2gray. It converts a red-green-blue colored image into a grayscale or monochrome image. The MATLAB website provides great resources to help you get started, including code examples, webinars, and extensive documentation.

26 - Octave

Octave is a free and open source language for numerical and scientific computing. It is mostly syntax and feature compatible with MATLAB. You can download it from the GNU Octave website. Follow the instructions for your corresponding system. Linux is probably the easiest, and Windows is a bit of a pain. I am on an OS X machine here. There is a binary installer for OS X. But I've had problems with it later on when trying to install packages. So I prefer the Homebrew route. If you don't have Homebrew set up already, first install XCode. Then open up a terminal and use the Xcode-select command to install command line tools. Now you can install Homebrew. You will also need a LaTeX distribution, like MacTeX. This may take a while to download. Once you have Xcode, its command line tools, Homebrew, and MapTech installed. Continue with the instructions on the wiki. Once you have Octave installed and it's in your path you should be able to run it by simply typing Octave. This is the Octave Interpreter. Similar to the Math Lab command window. You can type arithmetic expressions, create variables, and even work with vectors and matrices. To suppress the result of an expression showing up in the output, use a semicolon at the end. You can type exit to come out of the interpreter. You can also run Octave in GUI-mode. Note that this might still be an experimental feature, and tends to break very easily. You can carry out the same operations here as the console version. In addition you have a work space which lists the variables currently in memory. And a file browser showing contents of the current directory. As in Matlab, you can read images using the imread command and display them using imshow. The base installation of Octave does not include any additional packages. To install a package, we need to use the pkg install command. The -forge option pulls packages from the online Octave-Forge repository. Octave-Forge contains an extensive collection of packages. We are interested in the image package. Notice that this package has some dependencies. We can install the image package, along with all its dependencies, namely, general, control, and signal, in a single command. Once installed, they should show up in the list of packages. Now you can load the image package by typing pkg load image, and then use functions from it like rgb2gray. It converts a color RGB image into a gray scale or monochrome image. For more information on the image package, check out it's Octave Forge page. There is an extensive function reference that you'll need to use frequently. The Octave Wiki is very useful as well. It has installation instructions for a number of platforms, as well as tutorials and examples to help you get started. Play around with these to become more familiar with the Octave environment.

27 - Learning Goals Quiz

To help inform us a little bit, what do you expect to learn from this course? Put it in the text box here. We'll collect them all. And as we go through them, it'll help inform just a little bit in terms of how we think about the answers to the questions and the discussions that we have on the forum.

28 - End

All right. So, look, that ends the motivation and sort of what computer vision is and why you might do it and how we're going to do it in this class. Arvid, you have anything else you want to add? Yeah. I just hope that you're excited about the course by now. To make sure that you don't listen to the entire course like a podcast, I'm going to step in with some annoying little quizzes. Annoying, huh? Yes. Oh, okay. They we'll find out. Okay. Arvid has been practicing all his life about how to be annoying, that's what his mother told me. So I think he's got it down. But anyway, it's time we get started, it's a bit of work but I think if if you do it, you'll learn a lot of computer vision, so let's strap ourselves in and just get going.

2A-L1 Images as functions

1 - Images as Functions Intro

All right, well welcome to computer vision, you know hopefully, from the introduction, you feel, inspired to charge through here. And what we're going to do is we're going to start of easy, and and frankly, we'll get kind of hard towards the end, which is okay, because, by the time you get there you'll be, ready to do that. So today, we're going to talk about images, the most of you think about images as, something you see on a screen, or if you're old school, like me, something that's actually printed in a magazine, you know, it's something you can look at.

2 - Images as Functions Part 1

So here's an image of an old and I think now expired comedian who's, therefore, cannot sue me. That's Phyllis Diller, by the way, in case you remember. And by the way, we're going to start with black and white because black and white just makes everything easier because it's just a single channel. We, we'll do color on and off, but pretty much everything we do for black and white do, you do for color, hold for black and white, and, and it's just easier. So when I show this to you, you actually think of it as a picture or something to look at. But what actually is, is a function. In fact, we can just call it a function of I of x y, all right, where the I has something to do with the image intensity. So, if I think of this as a function, then I can just plop this as a surface and MATLAB makes this incredibly easy. And if I did, it would look something like this. Okay? Now this is the exact same function, but instead of showing you as a picture where, you know, sort of straight on, and by the way, the way MATLAB does it it's really cool, the, the higher the thing is it also makes it brighter, so you can see. So if you take a look at like the, the, the checkers pattern on that awful shirt she was wearing, right, so the bright spots are here, and the dark spots are down there. Okay, that function is the same function as the image that I was showing you before. Computer vision and especially image processing, we'll be talking mostly about the image processing side of computer vision today and the next few are about taking these functions and computing something from them. Often, we're just going to computer another image-like function, so images in, images out. And sometimes, we'll be getting some sorts of information. So here's a very simple example. Suppose I took that previous function, and I just smoothed it. All right, so now you see, I have the same surface I had before, but it's now, you know, it blends smoother, and the peaks and the valleys of that shirt are, are much smoother. They're not as steep as they were before. Okay. So that's the function. Now, of course, I can show that to you as an image again. What's that going to look like? Well, you've probably figured this out because you're all so smart. It's just going to be a blurry version of that image, okay. And I'm showing it here side by side with the blurred function, oh, sorry, the smooth function, right? Because there is this direct analogy between what we call blurring in the image and smoothing of that function. It's exactly the same thing.

3 - Image Quiz

So here's a quiz, all right? So an image can be thought of as A, a two-dimensional array of numbers ranging from some minimum to some maximum. B, a function I of x and y. C, something generated by a camera. And D, all of the above.

4 - Image Solution

Okay. So probably pretty obvious, it's all of the above. Now, I'm going to tell you, most Georgia Tech tests are a little bit harder than that, okay? But it's just to get you to pay attention.

5 - Images as Functions Part 2

So, let's talk a little bit more about images as functions, all right? So, we can think of an image as a function, f, sometimes we'll say f, sometimes we'll say I. That maps, you know, to, from R squared to R. That is, it goes from an x y to some pure intensity or value at that position x y. But we're not going to have, sort of, arbitrary functions, we're going to limit them in certain ways. For us, an image is going to be defined to be over some bound. So x ranges from a to b. And y ranges from c to d. And the intensity ranges from some min to some max. Now, raise your hands out there if you think that images value go from zero to 255. Come on, be honest, you're raising your hands, okay. That is a pure accident, zero to one probably would have made a lot more sense. Zero be black, white be white, oh sorry, one be white. Where did the 255. Well, you computer nerds we all know. Well there are 8 bits in a byte, right. So if they're all on they're all ones we'll call that 255. There is nothing special about 255, okay. And in fact, later we're going to even have to have images that can have negatives in them. So the thing that you just have to remember, is we're going to allow our images to go from some min to some max. The min will be sort of the blackest black, the max will be the whitest white, if we're actually thinking of them as intensities. But later if we think of them as just pure functions, like image derivatives, they just going to take on some real value. By the way, we can do color images the same way. Now, instead of having one function that maps x,y to an intensity, we just have three functions, often called r, g and b, sometimes called l, u and v. We're going to talk about that way later. All right? But they basically, you can think of this as what we call a vector-valued function so every pixel, the function is a vector of three numbers. Like I said, most of the time we'll be sticking to gray level images.

6 - Define an Image as a Function Quiz

Now how is this a function? Well, you can think of an image as a collection of light intensities at different locations. For instance, this area on the shirt is pretty bright, here this shadow is dark, and this grass somewhere in between. Now how would you identify these different locations? Note that these locations are laid out in a two-dimensional space. We can characterize it using, say an x-axis, which is the horizontal dimension, and a y-axis, which is the vertical dimension. Any location on the image can thus be specified using an x-axis value and a y-axis value. Notice that this x axis value can be any real number. Similarly for y. Now the image intensity at the position x, y can be written as f of x, y. And that's how you can think of the image as a function. Now what about the intensity values in the image? If we assume that they are real numbers than we can say that f is a mapping from R cross R, or R square, to R. Now this definition seems to indicate that images are infinitely large, but practically speaking, images have a finite size, they have a certain width and a certain height. Assuming that our coordinate origin is here, we can define numerical bounds for the image. For instance, 10 to 210 n the x-axis, and say, 15 to 165 along the y-axis. We also know that image intensity values have a finite range. Say in this image, they range from zero through ten. Given these finite ranges. How would you define this image as a function? Complete the function definition above by filling in the boxes. Note that the first pair of boxes is the range of column

values. The second set of boxes represents the range of rows, or y values. And finally, the third set of boxes represent the image in density range.

7 - Define an Image as a Function Solution

The answer is fairly straightforward. X values can range from 10 through 210. Y values can range from 15 through 165. And we define the image intensity values to be in the range 0 through 10.

8 - Define a Color Image as a Function Quiz

So a color image is also a function. It's just that the value at each location is no longer a single number that represents the light intensity. Instead, it can be a vector. A vector that holds three different intensities for three color components. For instance, each location in an RGB has values for Red, Green, and Blue. These values can be separated into three different channels or planes. If you take all of the red intensity values in an image, you have your red plane. Similarly, if you take all the green intensity values, you get your green plane, likewise, for blue. Given this scheme where each location is associated with a tuple of three values and assuming that each of these intensity values is a real number, and so are the values for x and y. How would you represent this image as a functional mapping? Mark the expressions that correctly represent the desired functional mapping.

9 - Define a Color Image as a Function

Let's first look at the range of the function. Since each location has a tuple of three values, hence the right-hand side of the mapping should have R cross R cross R, or R cubed. The first expression is therefore clearly wrong. Note that a color image is still a mapping over a domain of two dimensional x and y values. Hence the third expression is also incorrect. Therefore, we can represent an RGB image as a mapping from R cross R to R cross R cross R. This is the same as writing R cross R maps to R cubed. In fact, you could also shorten it to R squared to R cubed

10 - The Real Phyllis

So far we've talked about functions from a mathematical perspective, but this is a computer and in a computer everything has to be digital and that gives us even some more restrictions okay. So let's take a look at the grid that was a little chunk taken out of the Phyllis Diller picture. So here you see, in fact I had a little Matlab code right, it says pd. Pd Phyllis Diller. That was my array. And this says rows 40 to 60. Columns 30 to 40. And that's the middle of Phyllis's face. You might not have known it, but that's the middle of Phyllis's face. By the way, one of the things you should realize is this is exactly the same representation as the picture on the screen. But you happen to have a vision system that will look at bright dots and dark dots and see things. And when you look at these numbers, you don't immediately see those things. Mathematically, these are identical. Oh, and by the way, something that I just mentioned, which is going to bite us in some place. These are rows and these are columns rows go down columns go over so x and y okay remember that we're going to do that now.

11 - Digital Images

In, digital images in computer vision, we typically operate on discrete images, right, and that means we have to do two types of discretizations. First of all, we have to sample, the 2D space on a regular grid, that is we have discrete pixels at locations, you know what pixels stand for, picture elements, okay, in the old television world that was called pells also for picture elements, but for computers we have to be special so we call them pixels. So we have to pick them at specific locations, the other thing is we have to quantize, each value, we don't get to have a continuous real value, we have some finite number of bits to represent that, so like we said, maybe have 8 bits, so it

will go from zero to 255. These days, you tend to have 16-bit images, or 12-bit, or depending upon the device, but the idea is that it's quantized, to some level. Even though, it's quantized, later we're going to tend to think of these things as floating point, and I'll tell you now, that if you compute with integer images like unsigned integer 8, 8 bits your code will just break, so, use floating point images. So in general, especially in MATLAB, which we'll be doing a lot of, images are represented as a matrix of values, typically integer values to start with [INAUDIBLE] so here's Phyllis, looking as delightful as ever, and, we index our matrices by, again, i and j, row and column, sometimes, x going over this way and y going that way, if I say some pixel i j it means row i column j, if I say some pixel at x y, all right, x y, x is horizontal so I have to go get the column that, so you'll have to swap them, and part of the problem is our math is always determined by x and y, and our computing is always determined by i and j, row and column, and, and that's a, a tension that we'll have. Sometimes we use 1D signals, 1D signals will just be an array of, of numbers as well. All right.

12 - Compute Image Size Quiz

Let's take a moment to talk about image size. Here is an image defined over a two-dimensional space. Since it is a finite image, it must have certain bounds. Let's say the limits along the x-axis are ten and 330. And similarly, along the y-axis, 20 to 278. Can you tell me what is the width and height of the image? How about the area that it occupies? Type in your answers in the boxes provided.

13 - Compute Image Size Solution

That's right. The width is the difference between the two x-axis limits, 330 minus 10 equals 320. Similarly, the height is 258. And the area is simply width times height. And that comes out to 82,560. For digital images, the height is the same as the number of rows. And width is the number of columns. Thus, the area is the total number of picture elements, or pixels. Now, if each pixel has three color values for red, green and blue, a color image, specifically an RGB image, has three different values at each pixel. This means a color image of this size has 82,560 times 3 total color values. If each color value is represented by one byte, then you need as many bytes to represent the entire color image. This should give you a sense of how much memory you need to store an image on a computer and how it depends on the width, height and number of color channels.

14 - Matlab Images are Matrices

Most of you, I hope, will use either MATLAB or Octave for, for the work in this class. We talked about how you can use also Python and OpenCV, etcetera. MATLAB or the open source version of it of Octave makes it easy. If you're an actually student somewhere, you know, really registered somewhere, there is a student edition of MATLAB. It is less than most textbooks and a great thing for you to purchase. So, in MATLAB, images and matrices just work really well. So, you know, here's all it takes to read an image. Right? So we've got this function, imread, and we're going to read in as a file peppers.png. And by the way, if you don't put these semicolons in there, MATLAB spits out all the numbers, which is incredibly painful. And then what we're going to do here is, I'm going to take just the green channel, and the green channel can be indexed because in, in MATLAB, images have a certain number of rows, certain number of columns, and you can think of this as the color planes or the layers. So when I say im of colon that means all the rows comma colon all the columns. And then 2 that's red, green, blue. So that's green. Now some of you are screaming, no, Professor Bobic, you messed up. I will certainly mess up in this class. Although I will mess up less frequently in this class then in my in class, class. Why? Because Megan hates to see me mess up. That's actually not really true, she likes to put it in there, but anyway. No, unfortunately, I did not mess up, or fortunately? I don't know. MATLAB indexing starts at one. In, sort of, normal computer, zero would be red, one would be green, two would be blue. In MATLAB indices, indexing of arrays, it starts with one. So one is red, two is green, three is blue.

That means that we all the time have to be subtracting one off of indices in order to be able to multiply them to get into other locations, the computer scientists out there will know exactly what I'm talking about. Just remember that MATLAab is one based indexing. So we've got our green channel, which is just a single channel. Well, I can just show that. And if I can imshow, we'll talk about that in a minute. Well, green has just got a single layer, so it thinks of it as a grayscale image, and I am showing this case would just display it. And then you'll notice, MATLAB makes it really easy to plot things also. So it says draw a line, and it, can you see that red line in there? It just drew a line right across there. Ain't MATLAB great? In fact, I can also just call plot. So what this is, is this says, give me the 256th row. Give me all the columns and then plot it, and you'll just get a plot. By the way, I recommend highly getting used to sort of exploring your image. Plotting the values and being able to see is what's going on, what really should be going on?

15 - Quantize Quiz

Since a digital image is sampled at discrete locations in space, it can be written down as a two-dimensional array or matrix of values. Here is an example. Note that this matrix has fractional values, both positive as well as negative. But what if we could only represent a small set of integer values, say between 0 and 5? How would you quantize this matrix so that the result consists of only the integers 0 through 5? Enter the converted values in the corresponding boxes. Assume that we always round down. For example, 1.8 becomes 1. Also, be careful about the limits. Anything less than the lower limit 0, should be converted to 0, and anything greater than the upper limit 5, should be converted to 5.

16 - Quantize Solution

To get the quantized matrix, convert each number, always rounding down. Integer values within the given range remain the same. Anything above the upper limit becomes 5. Similarly, anything that is less than 0, in this case -1.3 becomes 0, and so on. Note how quantization results in a loss of detail. Extreme values beyond the range are lost as well.

17 - Load and Display an Image

All right, let's try to load and display an image. To load an image, we want to use the imread command. The image will be stored in this variable img. To display it, we want to use the imshow command. Hit Test Run and scroll down to view the output. A bottlenose dolphin surfing the waves. All right, what more can we find out about this image? What if we wanted to find out the size of the image? You guessed it, we'll use the size function. That returns the size of the image. To display it, we'll use the disp function. We can also find out the class or data type for the image. Go ahead and type out these commands, run the program, and note down the results.

18 - Image Size and Data Type Quiz

So what was the size of the image? Note, that Octave prints out the height first and then the width. What was class or data type of the the image? Type in your answers.

19 - Image Size and Data Type Solution

All right, let's find out the size and class. Octave prints out the height of the image first, and then the width, height is 320 and width is 500. On the next line we see that the class of the image is uint8. If you type these values in correctly good job. So the height and width turned out to be 320 and 500 respectively and the class was uint8. Now what does uint8 mean? Some of you may know this already, u stands for unsigned, which means this data type cannot represent negative numbers. Int

stands for integer and eight refers to eight bits or one byte. This is sometimes known as the bit depth. It indicates the number of bits allocated to store each intensity value.

20 - Inspect Image Values

Let's look at some values from our dolphin image. As before, we load the image with imread and let's also display the image with imshow. How about we print out the size as well? And let's run this to make sure everything's okay. All right. Dolphin's still there, and the size is still 320 by 500. Let's say we want to find out the image value at a particular location. We specify this location with a row and column number. We learned earlier that an image is a function over the two dimensions. Octave uses a notation similar to a function to access values at a particular location. Let's say we want to find out the value at row 50, column 100. We write img, that's our image variable. Followed by the row column coordinates in parenthesis. And let's display this value. All right, let's see what we have. So the value at 50, 100 is 208. Similarly we can find out the values for an entire row. We'll use a similar notation as before, but we'll do something different for the column. Putting a colon tells Octave to return values for all columns. Let's see what the output is. As you can see, Octave dumps values from the entire row, all 500 columns. Obviously, this isn't a useful way to look at values from an entire row. What else can we do? We can plot these values. This makes more sense, doesn't it? You can clearly see the relatively higher values where the white wave is, and then the other values are comparatively lower. Can you find out the values from this three by three slice of the image? Rows 101 to 103. Columns 201 through 203.

21 - Inspect Image Values Quiz

Fill in the values in the corresponding boxes.

22 - Inspect Image Values Solution

All right, to select the desired slice, we specific a range of rows and a range of columns. In this case, the range of rows is 101 through 103. And the range of columns, 201 through 203. And let's display these values, so the numbers are 81,77, 77, 81, 78, and so on.

23 - Crop an Image

Let's use this method of selecting a range of rows and columns to extract a larger portion from an image. By the way, this is also known as cropping an image. Let's use a different picture this time. Let's see what it looks like. All right, a classic two-wheeler there. Let's check the size of the image first. All right, 320 by 500. When cropping this image, we'd want our limits to be within this range. Let's say we want to select rows 110 through 310, and columns 10 through 160. I wonder what we'll find there. That's the front wheel. No points for that. So what is the size of the cropped image?

24 - Crop an Image Quiz

So what is the size of the cropped image?

25 - Crop an Image Solution

As before, let's use the size function. So it turns out to be 201 cross 151. Is that surprising? Why isn't it 200 cross 150? Let's look back at the range we selected, 110 through 310 and 10 through 160. Note that in both these ranges, the limits are inclusive. Which is why the first range, 110 through 310, includes 201 different rows. Similarly, 10 through 160 includes 151 different columns. Try to extract different parts of the image. What happens when your selected range goes out of bounds

26 - Color Planes

How about we look at a color image? What do you think is the size of this image? Let's find out. 258, 320, and 3. Why are there three numbers? The first two are the height and width. The third one is a number of color planes or channels in the image, which is three. So how would you select a single color plane? We'll use the same indexing notation we used to crop an image. Say we want to select the red channel, which is at index one. We want all the rows, all the columns, but only the first plane. Now what does this look like? All right. So that's what the red channel looks like. Brighter areas indicate higher red values and darker areas vice versa. The apples are not as bright as you'd expect. Why do you think that is? Let's also figure out the size of this color channel. As expected, the width and height of the color channel are the same as the original image, but it doesn't have a third dimension. Well, that makes sense, doesn't it? This color plane is one of three, which are stacked together to create the color image. Each one of them is a two-dimensional array. Note that the extracted color channel is an image by itself, so you can apply the same operations as you've seen before. Let's try plotting the values from a row of the image. And that's what the plot looks like. Play with this image in the code editor, try out different operations, try selecting other colored channels.

27 - Add 2 Images Demo

So what do arithmetic operations on images look like? Let's start by adding two images. Like before, we load up the images. Let's display them and make sure their sizes are equal. Here are the images, and as we can see in the program output, their sizes are equal. This is important because addition is an element by element operation. This means pixels in one image get added with corresponding pixels in the other image, and so on. Since the two images are of the same size, we can add them. Octave is intelligent enough to figure out that these two are matrices of the same size and hence performs an element-wise addition. The result is, as you would expect, an image that has elements of both source images. You can clearly see the bicycle and the white surf. You can also see the dolphin faintly visible. Notice how this image is exceptionally bright. Many areas are washed out. Why do you think the image is so bright? To find out, let's look back at our code. Note that we're directly adding values from both images. Areas where both images are bright turn out to be doubly bright. This indicates that we should perhaps scale down the image intensity values. By how much? We'll think of a pixel, where both these images have the maximum intensity value. That pixel in the summed image will have twice the maximum value. So if we want the maximum possible value in the summed image to be the same as the maximum possible value in each of these source images, then we want to divide the intensities by 2. Does this look familiar? Yes, this is the average of the two images. Let's see what this looks like. All right, much better. Compare this with the direct sum. You can clearly see the difference in brightness. Also notice that there are no longer any washed out areas. Let's rewrite the expression for average and see what happens. Since both bicycle and dolphin are being divided by 2, we should be able to add them first and divide the result by 2, right? And let's call it average_alt. Let's see what this looks like. Wait a second, this is not right. Shouldn't the two results be the same?

28 - Add 2 Images Quiz

To understand the result we are seeing, let's take a closer look at the computation that is happening behind the scenes. Here are the two images. The first image is the result of dividing each image by 2 and then adding them. And the second is the result of adding the two images first and then dividing by 2. The first image has better detail and is also slightly brighter than the second one. Now, we know that these images are just collections of intensity values. And values at each corresponding location are being added together. Let's take two sample values, say, 183 from one image and 152 from the other. Now, in the first case, we divide these numbers by 2 and then add the results. In the second case, we add the two numbers first. What do you think is the result in these two cases?

29 - Add 2 Images Quiz

The key to solving this problem is to know that both these images are of type uint8. This means that all pixel values are integers in the range 0 to 255. So here's what happens. Since the images are unsigned integers, Octave tries to retain the same data type throughout the arithmetic operation. So 183 by 2 comes out to be 92. Note that Octave rounds to the nearest integer. In this case, it is rounding up. Similarly, 152 by 2 is 76, and their sum comes out to be 168. In the second case, the addition is performed first, so 183 plus 152 is 335. But note that this number cannot fit in the unsigned int 8-bit range. The maximum value possible is 255. So this number gets truncated to the upper limit. The division proceeds as expected, and the result is 128. You can imagine that in a number of places, the pixel values add up to more than 255. In all these locations, you will only get 128 as the result. This is often less than the actual average of the two numbers. Hence we see that the first method better preserves pixel values. You will certainly come across these odd arithmetic errors. Just be mindful of the image data type you are using, and the order in which you perform arithmetic operations.

30 - Multiply by a Scalar Demo

In the previous example, we saw how we can divide an image by a number. Dividing by 2 is the same as multiplying by 0.5. And the order of writing these two doesn't matter either. The constant 0.5 is known as a scalar. This potentially comes from the fact that it scales the image values. Let's see what the result looks like compared to the original image. Halve the intensity values, clearly darker. Note that we can potentially multiply by any number, even greater than 1. Multiplying the intensity values by 1.5 makes the image brighter. And we see the same washed out effect in certain areas. This is due to the image values above 255 getting truncated at that limit. In Octave, we can write a function to perform a common operation. Let's turn the scaling into a function. We write a function by typing in the word function, followed by a variable name for the return value. Then an equal sign, the name of the function, and parameters in parentheses. This is followed by the body of the function. In this case, we want the result to be the product of value and image. To ensure that we are performing element-wise multiplication, let's change the star to a dot star. This doesn't make any difference when one of the values is scalar, but when the two quantities being multiplied are vectors or matrices, then star and dot star produce different results. We end the function by typing endfunction. Let us load an image and try out this function. And there is the scaled image.

31 - Blend 2 Images Quiz

Now that we know how to add two images together, and multiply image intensity values by a scaler, let's revisit our example of averaging two images. We saw that division by two can be rewritten as multiplication by 0.5. Now, this results in an image which has equal parts dolphin and equal parts bicycle. What if we wanted to change these ratios? Say we want more of dolphin. But note that, in order to keep the maximum intensity value the same as that of the original images, we should ensure that these weights sum to one. In general, this is known as blending two images. Let's see what it looks like. Yes, we do see a little bit more surf from the dolphin image, but it's a little hard to tell. How about we change the way it's a little farther? More dolphin. I wish we had a function to do this, which we could call like this. Can you write this function for me? Let me get you started. Put your code inside the function body. Remember, to return something, assign it to the output variable. Also note that a and b are the two images to be blended, and alpha is the weight to be applied to a. Once you have implemented the function, test it out with different values of alpha.

32 - Blend 2 Images Solution

We know that we want to multiply A by alpha and since the some of weights needs to be one we multiply B by one minus alpha. And finally we assign this to the output variable. That's it. Let's see

what we get. Alright, same image as before. Now see how easy it is to change the blending weights. For example, I want little less dolphin and more of bicycle. And, there we go. It almost looks like there is water on the street. This method of obtaining a weighted sum of two images is the basis of alpha blending.

33 - Common Types of Noise

So if images are just functions, then we can do things to images that we can do to functions. Like we can just add them, right? You can add two functions, right? Well, then we can add two images. And to introduce this a little bit, we're going to introduce the concept of noise, okay? So noise in an image. Is just another function that, combined with the original image, gives us a new function. So, we'll just write this, this way as our new image. We'll call it I prime. It's just I of x, y plus this noise function. You know, well what does that mean? Well, we have to take a look at what this noise function would be. Okay, so there are lots of different kinds of noise functions. Here's one, and this stuff's courtesy of Steve Sites, there's a type of noise called salt and pepper noise. Which doesn't take a rocket scientist for you to figure out that probably what it does is, it takes your original picture and it sprinkles occasional white spots and occasional dark spots. And that's called salt and pepper noise for the, for the obvious reason. A, a relative to that is something called an impulse noise, where you just get little white specks now and then. Different kind of imaging systems might give you that kind of noise. But by far, the noise that you're most familiar with is typically Gaussian noise, or normally distributed noise. Where we basically assume that at every pixel we take the original image and we stick on here some value that is independent identically distributed from some normal or some Gaussian distribution. All right, and that's Gaussian noise. And most of the time when we talk about noise we'll talk about that function. Okay? We can actually have Matlab make us a noise function. It's real easy. So here we say, look we're going to make a noise array, which is just, I take the size of my image, random n, randn generates a noise signal that has a mean of zero and a standard deviation of one, and if we scale that up by some sigma. Okay? That will spread that out and make it bigger so that's essentially the noise with mean of zero and a, a standard deviation of sigma. And because functions are just functions and images are functions, I can just add them. I can say let my output just be the image plus the noise. And if I were to plot that, you would see what's here, right? And on the right you can see that there's all this noise in our peppers. And if we plot this, you can see here we get this nice clean plot and here we have all this extra noise that's been added. And so that's our noise function.

34 - Image Difference Demo

If you can add to images, you can subtract them as well. The difference between two images is simply one image minus the other. It might be hard to understand at first what’s going on. Greater values in the difference image signify greater difference between the two images. Brighter areas in this result indicate where the two images differ more. Note that this is dolphin minus bicycle. Here the order mattered. Bicycle minus dolphin gives us a different result. This makes sense, as what the difference operation is doing is simply subtracting pixels in corresponding locations. If two such pixel values are a and b, then a minus b is different from b minus a. But when thinking about the difference between two images, we often don't care about which one is greater, and which one is less? Note that b minus a is simply a minus b negated. When thinking about the difference between two images, we often don't care about the sign of this difference, only the magnitude. That is, we're interested in the absolute difference between two images. For that you use the Octave ABS, or ABS function. Let's see how different the two results are. Wait a second. These two don't look different. In fact, they're exactly the same. What's going on? Let's take a closer look at our code. Especially this line. Let's say two values being subtracted are 20 from bicycle and 56 from dolphin. Theoretically the result should be minus 36. But remember uint8? These images can only represent numbers between zero and 255. So what happens here? It gets truncated to zero. Notice that even in the absolute difference case, the subtraction is performed first. This intermediate result is the

same as the original difference. The numbers here are already between zero and 255. So the absolute value operator doesn't make any difference. So what can we do about this.

35 - Image Difference Quiz

So we're losing out all the negative values in the result. How do we ensure that we can preserve image difference? Let's say a and b are two images of type uint8. Check the following options, which you think will give the desired result. For instance, should we compute the absolute values of the two images first and then compute their difference? What about this expression? Or converting to a different type? Would that help?

36 - Image Difference Solution

The first expression doesn't make any difference. A and b contain only positive integers. So this will give us the same incorrect result as before. The second expression is interesting. A minus b would give correct difference values where a is greater than b. And zero, where b is greater than a. Similarly, b minus a will give you correct difference values where b is greater than a, and zero where a is greater than b. Hence, there's sum is in fact the absolute difference that we want. Converting the images to uint16. Does increase the range of values that they can store. But remember that u signifies unsigned, which means uint16 cannot represent negative numbers, and we'll end up getting the same result. Floating point images can inherently store negative values. Hence, converting to floating point would help. Fortunately, there is a built in function to compute image difference that preserves values. We don't have to explicitly convert the data type or use any funky expressions. This function is contained in the image package in Octave or image processing toolkit in Matlab. You can load a package by typing pkg load followed by the package name. The function we want is called imabsdiff. It takes two parameters. The images to be subtracted. And the order doesn't matter. Let's see how this compares with our previous attempt. As you can see, this preserves the magnitude of image difference throughout the image. The image package provides many more functions to carry out common operations. Feel free to explore them.

37 - Generate Gaussian Noise

So we know that randn generates Gaussian noise. Let's see how it actually works. If you call randn without any parameters, then it returns a random number. Here we get 0.76388. Run it again. A different number, 1.3958. You can pass in dimensions to randn to generate a vector or matrix filled with random numbers. Let’s say we want a row vector of five columns. So one row, five columns. Each time we run this, we get different sets of numbers. As you might have guessed, we can generate a two dimensional matrix of random numbers as well. Say we want two rows and three columns. Since these are a bunch of random numbers, we call this noise. What is interesting is that randn draws these numbers from a Gaussian or a random normal distribution. Hence, the n in randn. A Gaussian distribution has a probability distribution function that looks like this. The center, or mean, for randn is zero, and the standard deviation is one. The standard deviation is a measure of how spread out the distribution is. I mentioned this is a probability distribution, which means getting back numbers that are close to zero is highly likely, whereas numbers far away from zero are less likely. How do we do know for sure that randn is actually sampling from a Gaussian distribution? Well, if we had enough samples and distributed them among bins and we counted how many numbers landed in each bin, then we would see a pattern similar to the probability distribution function. Let's try that. How about we start with a vector of hundred numbers? Instead of displaying the numbers directly, let's compute a histogram. Hist accepts a vector or matrix of numbers as a first argument and as an optional second argument, you can pass in bin centers. Let's say we want the centers to be integers, from minus three to plus three. Hist returns two values. One is the count of elements, which we want, and the second is the bin centers. Let us display the bin centers and the columns in a tabular form. We will create a small, temporary matrix, with the first row being the bin centers and the second row being the counts. As expected, the center has a high

count, and the ends have low, in fact, zero counts. You see the same behavior no matter how many times you run it. For a visual representation of what's going on, how about we plot these numbers? X-axis will contain our bin centers, and the counts will be on the y-axis. We see something that vaguely resembles the Gaussian probability distribution. To get a better picture, we need more bins. You can generate a sequence of uniformly spaced numbers using the lint space function. Here we can replace this vector by writing minus three to plus three, seven different numbers. That is including zero. Let's make sure this is the same as before. Note here that the bin centers are same, as expected. Now we can easily increase the number of events. Say, we want 21 one of them. I'm going for odd numbers because I want to include the zero in the middle. Displaying so many numbers wouldn't be useful, so let's comment that out and see what the plot looks like. Clearly, we have better resolution along the x-axis, but what's going on with these spikes? I think we need more data, let's bump up the vector to 1,000 numbers. Now you see the familiar bell curve slowly emerging. Let's increase the number of samples further. There you go. In addition to randn, you can find other random number generation functions in Octave or MATLAB such as just rand. This samples numbers from a uniform distribution. Randi generates random integers. Feel free to play with these functions.

38 - Effect of Sigma on Gaussian Noise

So I just snuck one past you there. Okay. We said the magnitude of the noise is determined by sigma. Fine that's great. In fact, we could just look at the noise function itself, right. So don't add in the original image. Just look at the noise function. But we can't do that just yet until we make a decision, and the reason is this. What's the mean of the noise. Megan? Zero. Very good. So that means some of the values are going to be what? Positive, some of the values are going to be negative. How do we look at a picture that has positives and negatives in it? Right? If we said zero was black and one was white, or zero was black and 255 was white, so how do we do this? Well, the mistake is saying that zero is black. Okay? We're going to say, look, we'll map some minimum value to black, some maximum value to white, and we'll distribute them in between. In particular, zero should be what color? What color do you think zero should be, between black and white? What comes between black and white in the universe? Grey. All right? So let's suppose we have values that you know, go from minus 20 to plus 20 in our image, well we can make minus 20 black, plus 20 white and, and zero would be grey. And if we did that, it would look like this. So here we're showing you images of Gaussian noise. Just the noise, So if there's a very small sigma, so remember sigma's up here, so a very small sigma. All right? You can barely see that this is anything but a constant grey. As we let sigma get bigger and bigger and bigger, you start to see more and more speckle. And that's the effect of sigma so it's just a noise function being added to an image.

39 - Effect of Sigma on Gaussian Noise Quiz

So likes these quizzes, so here we have a quiz, and if you can't do this with three eyes closed, I don't know. It says, look, there are four different sigmas here, you know, what are those sigmas? Well, they're 2, 8, 32, and 64. Can you label 'em for me?

40 - Effect of Sigma on Gaussian Noise Solution

Two, small, okay, right there I see something I can't. Oh, by the way, the reason we're using all these slides, my handwriting is awful, so you're going to be very happy that we have all these slides. So that's two. Let's see, I guess the eight is over here. The 32, yeah, I guess the one on the top left is less, and 64 is there. Okay, I passed my quiz. I hope you did too.

41 - Apply Gaussian Noise Quiz

Given an image img you know that its size is size of img and you can generate a noise image of the same size by passing this to randn. The values in this noise image will be normally distributed around zero. With standard deviation of one. What happens when you multiply each of these generated values by two? How does this affect the resulting distribution? Does it increase the counts? Does it increase the spread? Or both? Mark the right answer.

42 - Apply Gaussian Noise Solution

The correct answer is that it only increases the spread. Note that we are only multiplying the values. The number of values, or their counts, are unchanged. Multiplying a set of normally distributed numbers by a value effectively changes the standard deviation of the distribution they were drawn from. Now why is this important to know? Remember that randn generates values with standard deviation one, whereas the images we've been using are of type uint8 and range from zero to 255. What do you think happens when you add the results of randn directly to an image? Let's find out. Time to use a new image. If you look carefully you'll be able to see three moons and a shadow. Now we generate our noise image and add it to the original. Not really very different, is it? This is because the values that randn generated are really small compared to the image. Let's scale up the values. Now you can see the noise affecting the image. How about we increase this further? And more. Now it's really hard to see the moons, isn't it?

43 - Displaying Images in Matlab

I didn't say in the previous image what the range of our pictures would tend to be, okay? Remember I told you an image might go from, you know, 0 to 1. And that would be from the darkest black to the brightest white. Well, if I did that and I had a sigma of two. You would get black and white all over the place. And yet, when I go back here, sigma of two is just a small variation. Whereas sigma of 64 was a big variation. Why? In this image, we have this notion of maybe minus 127 was black and plus 128 was white. When we talk about the amount of noise in an image in terms of the intensity. It has to be with respect to sort of what's the overall range. So another reason to use doubles in your images. And to think of them as going from 0 to 1, then we can talk about a sigma of, you know, 0.1. Well, that's a tenth of sort of going from black to white. If you want to use something arbitrary like 0 to 255. You can do that. First of all, use 0.0 to 255.0, so use a floating-point number. And then you're going to have to say, okay, I guess a sigma of 0.1 in one case would be a sigma of, like, 25 in another case. Right, because I've stretched the whole thing out. So you have to worry about the magnitude of sigma with respect to the overall range of your image. This will c, catch you number, numerous times when you go to display an image. All right, because now you have to tell the machine, okay I've got this image. How do you want it displayed? Matlab has a large number of ways of displaying images. If you have the imshow function which I think actually comes from the Image Processing Toolbox. You can show it this way where you'd tell it low and high. And it will display, anything with the value low or lower as black, anything higher than high as white, okay? You can also do imshow and just give it this empty array. And it will scale the image for you automatically. That is, it'll find the minimum value in the image and say, okay, that's going to be black. It'll find the maximum. It'll say, that'll be white, and it'll scale you. There's another function called imagesc, for image scale. It's a much older function. It's not in Image Processing Toolbox, which will also display it. Don't get caught between this question of how I display an image versus how I use an image. Just I just finished teaching part of this course here at Georgia Tech. I had some people doing, computing some gradients. They computed some gradients, which involves substructions and derivatives and all that stuff. And then one guy, he normalized his picture to go from 0 to 255, before he computed with it. You would only normalize it in order to display it, not in order to compute with it.

44 - Adding Noise Quiz

So here's a question where we start to talk about noise. When I add noise, to images, as an arithmetic operation, which of the things do I have to worry about? A, the speed of the addition operation. B, the magnitude of the noise compared to the range of the image. C, whether we add the noise to the image or we add the image to the noise. That is, the order of the operation. Or D, none of the above.

45 - Adding Noise Solution

Well, I'll tell you most of the time when I give a quiz, the answer is all of the above or none of the above, but not really in this case. The speed of the addition operation, you know, back in the Dark Ages, you had to worry about the speed of addition, now addition is instantaneous. Okay? And by the way, last time I checked, 4 plus 2 equals 2 plus 4. Addition is commutative, so we don't have to worry about whether we add the noise to the image or the image to the noise. It's kind of weird to think if I start with some noise and add an image to it, same value. But what you do have to worry about, as I said before, is the magnitude of the noise compared to the range in value of the image.

46 - Images as Functions End

So that ends our first lesson on, images as functions and image processing, our first technical lesson. I hope you didn't find it too pedantic or too boring, and you'll come back for the next ones because as we go forward, there'll be more and more cool stuff to do. I can hardly wait, and I hope you can too.

47 - What Did You Learn Today

So what did you learn today? List any new concepts, terms, or commands that you picked up in the lesson. Comma separated, or one on each line. You can use this, and similar notes throughout the course as a reminder for yourself. This also gives us a sense of how you learn, and will help us improve this course.

2A-L2 Filtering

1 - Intro

All right, welcome back to Computer Vision. Last time we talked about images as functions. An image could be thought of as a function of two dimensional position like i of xy. Or a two dimensional array we'd say f of ij row column. And sometimes I'll try to distinguish between whether I've got rows and columns and x and y. And we talked about how x and y are not the same as rows and columns. In fact, a row is typically going down so if we go row, column, that's y, x. The other thing is, and we're going to talk about this in a little bit, there's going to be this question about whether y gets bigger as you go up in an image, which is like what you learned in grade school, right? Y positive goes up. Or maybe a lot of you think of rows going down, and so we'll, we'll have to wrestle with that. And, we also talked about that, even though indexing in a computer world should start at 0, in MATLAB, indexing of arrays starts at 1, and you're sure to screw that up somewhere in your code, but eventually, you'll learn to figure it out.

2 - Gaussian Noise

All right, let's do some work, not much work, but a little bit of work. So last time we talked about, adding a noise function to an image function, so here we have our noise, defined as just this

random things scaled by the sigma, our output was just the image plus the noise. And we said, remember, you have to worry about the size of the sigma in respect to the range of the image, so if your image is 0 to 255, a sigma five might be plausible, if your image goes 0 to 1, size of sigma five is not plausible, so you have worry about how those come together. Now, suppose there was noise in your image, and you wanted to remove the noise, how might you think about doing that? Now I'm sure, bunch of you have suggestions that are kind of similar. Here, here's the typical one, right? Let me replace the value of each pixel, with sort of an average of the pixels around it. Okay? So. Let's think about what that would look like in 1D, and then we'll go to 2D, and we'll talk about why is this the right thing to do or when is this not the right thing to do, all right. So, here's our first attempt, so, we're going to replace each pixel with an average of the values of the pixels neighborhood, and this is referred to as a moving average, so here I have some location here, and I just take the average value and I put it down there, okay? And then I would move my little, what's called window as I take my average, and I get a new value, and I get another one, and then eventually, I would get this new somewhat smoothed version of the original, and it's smoothed meaning that we sort of averaged of the things locally.

3 - Averaging Assumptions

All right look, that was not rocket science, that seemed reasonable. But the important question to ask, really important question to ask is, why was that solution intuitive for you? Why did you say, you know what, I'm going to take a little average in order to get rid of some of the noise? What assumptions are you making about pictures, and by the way, about noise, in coming up with that answer. All right, so a little thought we come up with, you're making a couple of key assumptions. The first assumption is that the real value of a pixel is probably similar to the values nearby. Okay. Otherwise, why would I use the pixels nearby in order to try to figure out what my real value was? So okay that was assumption number one. The other thing that you're assuming. And for those of you who know a little, just a little about that problem, though the math etc. You're basically assuming that each noise that's been added to each pixel is independent of the noise that's been added to all the other pixels. And so what that means is if I take the average of the noise, that's going to tend to, I'm going to assume zero. That is that sometimes it'll be up, and sometimes it'll be down, and if I take the average around them, that that will be zero. So if I have pixels nearby having about the same value are related and the noise is independent, then the average is the right thing to do.

4 - Noise Quiz

So here's a quiz just to sort of keep you on your toes, or at least to keep you mildly awake. If noise is just a function added to an image, we could remove the noise by subtracting that noise again. That is, the operation's reversible, okay? A, true, yeah, we could do that. B, true because we don't know the noise function so we wouldn't know what function to remove. C, false. When you put noise into something, it destroys information in the image and there is no way to recover it.

5 - Noise Solution

Well, this is one of these funny things. Hm. Well, these certainly true. Okay. Yes, I could remove the noise if you told me what the noise is. But nobody ever tells you what the noise is. Because if they did, they would have removed it in the first place. So you don't know what the noise is. You just know the statistics of the noise. And then, here is one that's interesting. False, because additive noise destroys information. Well, if you have just plain addition, and the addition works, then that statement is not true. And so, C would be wrong. But, suppose you have a limit between 0 and 255 or even 0 to 1. If your noise pushes you beyond the limits and you clip. You've lost that information. So sometimes when you add noise you get these nonlinearities. But in general, we can't remove the noise because we don't know what the noise was. If we did, we could just subtract it.

6 - Weighted Moving Average

Let's revisit our weighted average. Okay. So, instead of just thinking about averaging the local pixels, let's think of this as a set of weights, okay, and what we're going to do is we're going to weight all of these pixels by some set of weights, and we're going to combine them using those weights to come up with the new value. Now when we were just doing the moving box, our weights were just, let's say it was five long, they would be one fifth, one fifth, one fifth, one fifth, one fifth, five of them, right? So our weights was uniformly distributed, okay. But wait a minute, does that make sense? Remember our assumption that nearby pixels are related to my value? And there's another related assumption to that. The more nearby you are, the more related you are. So that means that pixels that are closer to me should contribute more to the average. So let's change our moving average, so that that's true, okay? Before, if you remember, we had these uniform weights. One, one, one, one, one, all the way across, divided by five. What if we wanted to use some nonuniform weights? So, so here's a set. One, four, six, four, one divided by sixteen. And what I'm going to do is I'm going to change those weights from being uniform to being nonuniform, and I want you to watch what happens to this bottom picture. So that's the uniform weight, and that's the nonuniform, centered weighted, right. Uniform. Nonuniform. And you'll notice that the nonuniform is a smoother rendition. In fact, you know, take a look here. You know, this area right there, we get a smoother range of, of what's going on. And the nonuniform weights corresponds to this assumption that the more nearby a pixel is, the more weight it should have.

7 - Moving Average Quiz

So, question, if you're going to do the moving average computation, the number of weights that you have should be what? A, odd, because it's easier to have a middle pixel. B, even, that way it can be symmetric around me. Or c, obviously either even or odd.

8 - Moving Average Solution

For most of us, we tend to use odd sized sets of weights. And what that lets me do is put my set of weights centered over the pixel that I'm changing. All right? Otherwise, what I have to do is sort of put the center between the set of weights and average them that way. And that what, and then, so we tend to use a odd but symmetric weight mask.

9 - Compare Filter Results Quiz

All right. Let's talk about comparing filter results. I took a vector of ten random integers and plotted them as a line plot. I then filtered the vector with a uniform filter of length five and plotted the results. I also did the same with a nonuniform weighted filter. Can you figure out which is which? Type in the filter that you think produced the line plot in the corresponding box. Use the same notation as you would to write a vector in Octave or MATLAB. That is, including the two squared brackets for the original set of numbers you can type in the word, original.

10 - Compare Filter Results Solution

The blue line has the biggest extremes. It's got to be the original. The green line is smoother, but has some uncharacteristic segments. For instance, here. While the original sequence decreased significantly, the green line actually increased a little. Similarly here, the original sequence increased, but the green line dipped. This indicates that values other than the central value at each point are affecting the results too much. Looking at the first unusual spike, it seems that the peaks on both sides are contributing as much as the central value. The green line is a result of a blindly uniform filter. The pink line is also smooth, but more importantly its peaks and troughs are well-aligned with the original. This seems to indicate that the central value had more weightage

compared to the neighboring values. So yes, the pink line is a result of applying the center weighted filter. One minor detail, when applying these filters, we would want to scale the weights so that they sum to one. For the uniform filter this would be dividing by 5. And for the non uniform filter we'd divide by 16.

11 - Moving Average In 2D

So what about images? Well images are two dimensional, and doing this in 2D is pretty straight forward extension of 1D with a, with a little bit of a flip. That's going to be a joke. Alright, so let's do a moving average in 2D. I hope you can see that there's a whole bunch of zeroes in this picture over here. You see all those zeroes? And then in the middle on the bright area are these 90's. So this maybe is an image that goes from zero to 100, or whatever. But the idea is that zeroes are dark, and 90's are bright. And what I'm going to do is I'm going to take a moving average, so here I have my little average right? It's a three by three average, and I'm going to take the middle pixel, over here and I'm going to put that average over there. Okay? So if I put my average overall at zeros and I take an average of all zeros. I get, Meagan, what's the average of all zeros. Zero. Every now and then I'm going to ask her some questions she has no idea what the answer is. And she's not even sure she's supposed to know the answer, so that's kind of cool. Okay, it's zero. So now we move our square over one. Right so we, we move it over here, we get zero, zero, zero, zero, zero, zero, 90, okay. What's the average of that going to be? Well, you can think of each of these as one ninth, so one ninth times zero summed up eight times plus 90. That average is going to be ten. Okay? So it turns ten, and then I slide it over one more, and now I get 290, so its average is going to be what? 20. And keep going. I get 30, and keep going further to 30, so I'm just moving this along, replacing the average. And finally, I get this total averaged value, right? And you can see that by putting a three by three, I get this nice bright area right around here which is where my three by three can sit over all the 90 values. And everywhere else it sort of falls off. And you'll notice the user zeroes here until we get to these tens. Where do those 10s come from? They come from this one little, bright spot there. Okay? And one of the things, by the way, you may have noticed is, by the boundaries we get these fun, kind of dark values. And we're going to talk more about, boundary conditions in a minute. All right.

12 - Correlation Filtering

So the math of what we just did is referred to as correlation or correlation filtering. And what we just did is we did it with uniform weights. So let's assume that our averaging window is size 2k plus 1 by 2k plus 1. So remember it's odd right, so if k were, were a three that would be a seven by seven. If k was one that'd be a three by three, right? So we have a odd sized window. So what we do is we're going to loop over all of our pixels around some location F of i,j, i,j being the row and column. And we loop over all the u's and all the v's going from minus k to plus k, summing up all of the pixels. And then, since we have the uniform weight right, so you have a weight of one there, we divide the whole thing by the number of weights in the filter, right? So if I had a 3 by 3 that would be 9. So if k was one, two, three. Yeah, it would be one-ninth. But that was uniform weights, and I don't know about you but I don't look any good in a uniform anymore. So, we're going to do something totally different here. No, all we're going to do is we're going to do what are called non-uniform weights. And what that means is instead of having 1 over 2k plus 1 squared we're going to have a different weight at each location. And so these non-uniform weights are written here as H of u, v. And those are simply the weights that you're going to, still some over the whole thing, but now these different weights are going to be applied as you move around with inside that mask. This operation is referred to as correlation or the cross-correlation and I've written this here. We would say that G was the correlation or cross-correlation of H with F. So this H by the way is referred to by many things, okay? It's called a kernel, it's called the mask, it's all, it's called the coefficients. It is just the matrix of the linear weights that get used. By the way, in case any of you have taken a machine learning class, and you've turk, talked about machine learning

kernels and the kernel trick. There's a sli, very very slight relation here but really it's, it's a totally different thing. We're thinking about these kernels as these masks that we do the computation.

13 - Averaging Filter

So then the question arises, what makes a good kernel? Well obviously a very successful major. That was joke. So let's consider a uniform filter. Did you get that one Megan? Yes, you got it, good. All right, so you remember the averaging filter? Right, so the averaging filter was just this box filter of all one-ninths. And as we moved it around, we took this image on the left and we created this image on the right, all right? So the question actually arises. What if I had some image like this, and I apply that box filter. What would I get? Well, I would get this, and that is known as really, really ugly. Okay, that does least not to me, as a nice smooth version of the original. Now if you're looking at an itty-bitty monitor, it might look nice. But trust me it's really ugly. And the question is, what went wrong? Well, really what went wrong is squares are not smooth. Which is probably a statement from the 60s, but it's, it's what I mean. Basically filtering with something that's not smooth to try to think about blurring or filtering an image to make it smoother doesn't seem right. And in fact we'll be able to say a lot more about what smooth means in terms of mathematics in a few more lectures when we talk about [INAUDIBLE] analysis. You know, so what was the problem? To get sense of what's wrong, imagine that you had a single point of light that you're looking at very far away, right? And you've blurred the camera, so it was out of focus. What would the image look like? Well, it would look something like that. Okay? Brighter in the middle, falling off towards blackness at the edges. Very, very deep. All right. And in fact, if we think of images as functions, I could take this image, and I could plot it as this function. If I've taught at that blurry spot, what kind of function would I get? Well it would look something like here on the right.

14 - Blur Quiz

Thinking about this, if we wanted to blur a single pixel into a blurry spot, we would need to filter that spot with what? Is it A, 3x3 uniform weights? Is it B, an 11x11 square of uniform weights? C, is something that looks like that blurry spot?

15 - Blur Solution

Could we use a 3x3 u, uniform weights? No we did that and it looked awful. How about, an 11x11 square of uniform weights, that would be better, right, because it's bigger. Well, sort of yes sort of no, you still have this problem that you get these very hard boundaries. Clearly what you want, is something that looks like that blurry spot, higher values in the middle, lower at the edges

16 - Gaussian Filter

So what would such a function look like? Well, it would look something like this and it would be, what's referred to as a Gaussian filter. So here's our original image, here is our filter. Now this is not a great Gaussian because it's only three by three. But you'll notice it's higher in the middle, then falls off at the edges and even falls off more at the corners. And what that is is, that's a very sort of low representation, very coarse representation, of a Gaussian function. And our Gaussian function is written here. This is the two-dimensional Gaussian in the u, v. 1 over 2 pi sigma squared, e to the minus u squared plus v squared over sigma squared. That's a circularly symmetric Gaussian function. Get to know and love your circularly symmetric Gaussian functions. All right, so here I have, that's in fact what that plot was, so we're going to do smoothing proportional. So this exponentially to the minus x squared plus y squared over 2 sigma squared. And the technical description of this, is that it's a circularly symmetric fuzzy blob. That's what the Gaussian is going to be. So if we take that image, that same high textured image that we had before, and we filter that, with this now Gaussian blurring function, what are we going to get? We get something that looks

like that. So much better, so much nicer, right? I mean, compare smoothing with a Gaussian to the non-Gaussian. Gaussian, non-Gaussian, right? And you can see the difference. In fact, in the non-Gaussian you see all these sharp edges, now you really seem them because they're red, all right? But when you do it with a Gaussian you get that nice smooth blurring.

17 - Gaussian Quiz

Ok so now we're going to get to a quiz that's actually hard. It's not really hard, it just kind of feels hard, because I always screw it up. And that is, the Gaussian filter that we're talking about is referred to as a, as a exponential, and the complete formula here is what? So h of u, v is, by the way you see the exponents? They're the same everywhere. The only thing we're looking at that's constant. So, the question is, what's the right constant? One over 2 pi sigma squared, 1 over the square root of 2 pi sigma squared, 1 over the square root of 2 pi sigma.

18 - Gaussian Solution

Okay, going back to here, it's just two pi sigma squared. I always forget where everything falls. Those of you who know your normal distributions, you'll remember that that's the constant that's in front of the Gaussian.

19 - Variance or Standard Deviation

So the Gaussian we just talked about is what's referred to as isotropic, a fancy word for circularly symmetric. And so it basically had one parameter, just sigma, right? Bigger sigma is, the more the blur. But we have to implement these in digital computers. We actually have to have a matrix, and so then we have to worry about two things. The size of the matrix, is the matrix three by three, five by five, eleven by eleven? And then the sigma that's represented within that. So let me show you that first. In Gaussian filters we talk about the variance, sigma squared of the standard deviation of sigma, and that's the amount of smoothing. So these two filters are the same size, they're both 30 by 30s, but the sigma inside one of them is two, the sigma inside the other one is five, so the five has a larger sigma. Now sometimes we'll make a mistake and say it's a larger kernel. When we say larger, what we actually are talking about is the size of the sigma inside. And we can distinguish that. So here's an example of the size of the kernel. So both of these kernels have a sigma of five, but one of them is represented within a 10 by 10, and the other is represented within a 30 by 30. Which one do you think is going to work better? The bigger one, it's smoother, it's going to work better, okay. And by the way, if somebody says, oh I, I smooth that with a bigger kernel. They actually mean a bigger sigma. That's what you care about, is the size of the sigma, not the actual kernel. The kernel has to be big enough and then the sigma is, is within that.

20 - Matlab

As I mentioned at the start of this class, I'll be showing my examples mostly using Matlab. I know we've looked at the course developer. Arpin has worked on being able to do it in Octave or also in Python using various image manipulation OpenCV. But what I'm going to do is show you filtering in Matlab. And basically Matlab makes it trivial to build filters and apply filters. So Matlab, what we're going to do is define two things. We're going to define the size of the kernel. Remember that's what we were talking about before. So in this case it's going to be a 31 by 31. Again, odd, so I can put a center pixel down. I'm going to a sigma of five, and Matlab has this really great little function called fspecial. And obviously it's special, or they would call it something else. In fspecial, you can give it parameters, one of which is the type of filter you'd like. You can give it the size and the sigma. You can also give it rectangular size and multiple sigmas. It will build these filters for you. And in fact, Matlab has this beautiful little function called surf. Right, which will plot for you as a surface and if you do it with the right color map you would see this. You could also show it as an

image. That's what this little picture is right here, all right? But even more importantly, you can take your image, and that's our image here of a panda, I can filter it by this h, which was the filter we just built, and then I can show that. What's that going to look like? It's going to be a blurry panda. Okay. This code is all it takes to build your filters and apply them to images in Matlab. It makes it very easy. Again, depending upon the size of the sigma we get different amounts of smoothing. So here we're using three different sigmas of 1, 3, and 10. We build our Gaussians using the different sized sigmas. We filter them and show them, and you see that we get, you know, hardly any blurring, little more blurring, and a little more blurring. That's all it takes to build these filters in Matlab.

21 - Remove Noise

So we talked about removing noise using a filter, let's see how well that really works. Let us load a perfectly good image. Spoil it by adding some noise. I should really name this sigma to avoid confusion later. Finally, we know how to create a Gaussian filter. We define a size and a sigma. And then we can use the fspecial function from the image package. So load up the package first. And then create the filter. Now, we can apply this filter to remove noise. Note the order of parameters in imfilter. First is the image and second is the filter. Notice how the filter has smoothed, or rather, blurred the image. The fine particle like appearance of the noise is now smudged. But the filter has also affected the original image a great deal. So, noise removal is no magic. You don't get back exactly what you started with. Visually, it might not seem very impressive, but image processing routines further down the road behave quite differently given a noisy image versus a smooth image. Go ahead and run this code yourself. Try out different parameters for noise generation and smoothing.

22 - Gaussian Filter Quiz

So when filtering with a Gaussian, which is true? Is it A? The sigma is most important. It defines the blur kernel's scale with respect to the image. Is it B? The kernel's size is most important because it defines the scale. C, altering the normalization coefficient, that constant on the front, has no effect on the blurring, it only effects the total brightness. And D, A and C.

23 - Gaussian Filter Solution

Okay, well, so it should be pretty clear that it's A and C. Why? because the sigma is what defines the blurring. And then the constant in the front, remember the one, and one over two pi sigma squared, okay? That just multiplies everything, so that's just going to change the brightness. So fundamentally, what matters is the size of the sigma or sigma squared.

24 - Keeping the Two Gaussians Straight

Finally, a word of warning or clarification at least. We just talked about sigma as being the width of a Gaussian, where that was the variance of the, of the smoo, smoothing of the blurring. Last time we talked about sigma or sigma squared as the variance of a noise function, how much noise was being added. So in one case, where we share might the filter, the sigma is in space, okay? Where as in the noise case it's in an intensity or it’s in the value. The bigger the noise signal is, the more noise you added, the bigger the blurring filter sigma, the more you're going to blur. So, you have to be an in a reasonable called sigma as they're both using a normal distribution. But one is over a space, and one is over intensity. We can show those two sigmas here, and by the way, here I'm using I'm using images that go from zero to one. Partially because that I know how sigma varies with respect to the range of the image, and partially because the slides that I stole off the Internet did this. All right? So in the top row, there's no smoothing going on, so we have a sigma of 0.2 in the noise. That's a lot of noise if the range is only going from 0 to 1 or, or let's say minus 0.5 to 0.5. A sigma

0.1 is less noise. A sigma 0.05 is less noise yet. But then we can smooth it, all right, with a Gaussian. Right, this is the smoothing, with the Gaussian kernel. And the more we smooth, the blurrier these things get. And so, for the same amount of smoothing, the thing with the smaller amount of noise, with the same amount of smoothing becomes even smoother, right? So this image here is even smoother than that image there. Okay? But this is showing you those two sigmas, the two Gaussians. Almost always it'll be clear from context, but since I've had situations where students say, wait a minute, I thought the bigger signal was, the more noise we got. Now you're telling me the more sigma is, the more blurring is, the less noise we have, and I say yep, two sigmas.

25 - End

All right. That ends this lesson on filtering and noise. And we'll start going down further image processing next time.

2A-L3 Linearity and convolution

1 - Intro

Welcome back to computer vision. today, we're going to continue talking more about filtering and sort of finish up the basics about that, so then next time, we can apply that to more complicated uses of filtering. And we're going to start by developing some linear intuition. And the reason linearity is important will become clear in a minute. Let's do a little intuition. So an operator, we'll call it H, or a system, is called linear if two properties hold. And for what I'm going to show now both f1 and f2 are going to be functions, and a is going to be a constant. So the first property is called additivity, which is basically just that the things sum. So if I have some operator, and I apply that to the sum of the two functions, f1 plus f2, I just get the sum of the operator applied to each of the function, looks a lot like the distributive law of addition and multiplication, but that's additivity. And the other one, which is sometimes called scaling property, but is actually technical term I think is homogeneity of degree 1, is just that if you multiply your function, in this case, f again, by a constant a, and then apply H, what you get is a times H applied to f1, and so when you multiply by a constant, that constant just propagates through in a linear way. We do some multiplication, and then we sum them, and because multiplication and addition both have these properties basically, the filtering operations we're going to do are going to be linear. And we're going to take advantage of that later.

2 - Linear Quiz

Question. Which of these are not linear? a) the sum, b) the max of a function, c) an average, d) a square root, e) (b) and (d)

3 - Linear Solution

Okay, well that's pretty straight forward. A sum, well a sums are sums are sums. You know that's linear it's going to do the right thing. A max of course doesn't change, right? If I've got two functions and I'm taking the max it's determined by, sort of, the single biggest value and the rest of the function doesn't matter. And of course square root, well that's not going to be linear because square roots are never linear. I think you can figure that out.

4 - Impulse Function and Response

What linearity is going to allow us to do is to build up a signal or a function, remember an image, a piece at a time and then be able to say how a linear operator affects that whole image. Right?

Because, basically the way both linearity allows us to say is if I could sum up a bunch of things to make an image, then if I apply a linear operator to that whole image, it's going to be the same as the sum of applying that linear operator to each of the pieces. And it let's us talk about things in an effective way. So, to do that we need sort of these building blocks of a function. And the building block of functions is what's referred to as an impulse. And the impulse function in a discrete world is very easy to talk about. It's a single point with a value 1, and it looks just like an impulse. In the continuous world, an impulse is a little bit trickier because what does it mean to be sort of at a single point width, you know, it would have some amount 1, and that's where you have to turn on your calculus hats just a little bit, no, I guess you put on your calculus hats, right? And an impulse is actually a small, little signal whose area is 1. Okay? So, and as the thing gets narrower and narrower in width, it has to get taller in height in order to maintain that same area. And in the limit what you get is an impulse. So it's got 0 width and infinite height, but it's, it's integral, it's area is 1, and that's what's referred to as an impulse. We're going to mostly stay in the discrete world, so we won't have to worry too much about that. So the, the cool thing about an impulse is the following. Suppose I've got some sort of a system, a black box. So it's a function H, right? So, sorry, it's an operator H. And we don't know, actually, anything about it. But if I put in an impulse, okay, so I put an impulse into the system, I can see what comes out. By the way, that response is called the impulse response. Not, not all that clever, right? So what's really cool is, if I know what the impulse response of some black box H is, maybe we'll call that h of x, okay. I can describe what this operators going to do by h of x. The reason that works is the following. Since any sequence of pulses here, and we're going to do this in 2D in a minute, can be described by just adding in a shifted set and scaled set of those single impulses. If I know how this black box effects just the single impulse. I'll be able to say how it affects the entire image. Okay? And that's why impulse responses matter.

5 - Filtering an Impulse Signal

So, let's take a look at what an impulse response looks like in an image. So here we have an impulse. Okay, and so, here's our impulse, right? It's just a single pixel with a value of one, okay? And we're going to filter it with some kernel, h, okay? Remember kernel mask, thing that we're moving around in order to make our filter, so we have our H. So the question is, what is the output? And again, I want to thank Christin Gramen for having made these slides so I can change them, steal them, move them around. So first I put down my filter, and you see my filter's over here like this, right? So it doesn't hit that 1 at all, and so its value is just going to be a 0. Fine, no big deal. All right. Now, move the filter over a little bit. Well, it just pulls out the f here, right, so this pixel right there pulls out the f so an f goes right there, in, in my result. Move it over one more time, what do I get, I get the e. Move it over one more time again, what do I get, I get the d. So, you notice that even though the filter goes D-E-F, left to right. In the image, what comes out is F-E-D, going the other way, and that's just because of how this correlation pulled it up. And it won't surprise you that it, not only does it flip it left and right, it also flips it up and down. And that's what happens when you do a correlational filter of a impulse. Just because of the way you slide it over, you'll end up flipping that entire response. By the way, something that I was being implicit about is I was assuming that this center of that filter was what we were calling the reference point, right. So wherever the center was located, wherever that center pixel. That was the one that was being entered into on the result. Okay. You could use a different point, but we're going to assume that the center pixel is the reference pintal, pixel of the filter.

6 - Kernel Quiz

So here's a quick little quiz for you. Suppose our kernel, our filter, the thing we're moving around, was size M by M, right? So maybe it was a five by five or a three by three, so M would be three or five in that case. And suppose our picture was N by N, maybe our picture is 100 by 100, or 1000 by 1000. How many multiplies would it take to filter the whole image, with that filter, all right?

Two M N. M times M times N times 2. M times N times N. M times M times N times N. Hallelujah singing something. I don’t know it just seems like it had a rhythm to it there

7 - Kernel Solution

So let's think about it. Well actually in fact, the answer, and we would normally write it this way, M squared N squared right? Because every pixel I have to multiply all of the coefficients times the pixels that are underneath. So there are M times N coefficients. And I have to do that over all of the N times N pixels. So the number of operations is on the order of M squared N squared, which can get pretty large if M and N are moderate size. Later we'll talk just a little bit about what are called linearly separable filters. We don't use them a lot anymore if you've got really fast computers, but it, it can make things be much quicker.

8 - Correlation vs Convolution

We had this problem, of when we put in through a correlation, we got back out sort of this flip thing, all right. And, if you remember, here's the equation of correlation, we have this kernel H and we sum over it, going from minus k to plus k, multiplying it times our image, and what that did was it caused us to end up with that flipped result, all right. The right way of thinking about this, is that, when an impulse comes in and it hits the filter, what comes back out is sort of this, this reverse signal. So, the, the right way of thinking about the operator is there's something called convolution, and when we do convolution, that's what we actually mean when we say we're going to apply this filter or this kernel, and what convolution does, is it, flips both the left-right and the up-down direction, you could have either flipped the kernel or flipped the axis to the pixels, it doesn't matter, you would get the, the same value, and, so that flipping gives you what's referred to as convolution. So by the way, if I was using a Gaussian or a box filter, how will the outputs be different for correlation and convolution, that is what happens if I flip my Gaussian, answer, nothing, okay? For a circularly symmetric or for a symmetric filter, whether I do convolution or correlation doesn't matter, it is going to matter to us in the next lecture, the one after that when we take derivatives in one direction or the other, and that's when you have to be careful, but if you have a symmetric filter, it doesn't matter. So this can be illustrated nicely in the following way, so here we have the, equation for the convolution operator, and we're going to illustrate it like this, all right. So here we have our filter and there's this little asterisk up here and the asterisk is to show you sort of what the top right hand corner is. When we do correlation, we just pick that up and then we sort of slide it around, when we do convolution, what we do is we rotate, the thing, and, because it essentially flips it left, right and up, down, right, you see that the little asterisk is now down here in the bottom left hand corner. Okay? And then, thank you, Kristin. We zoom that all over the image and that gives you your output. So, that's our convolution operator. Again, the difference between correlation and convolution only matters if you have an asymmetric filter but now you know the difference. Like I say, convolution is actually sort of the physics, so what's going on when you put an impulse through this response.

9 - Convolution Quiz

When convolving a filter with an impulse image, we get back the filter as a result, right? So in other words, by doing the convolution, we, we flipped it, and then we rotate it and it flips again, so I take an impulse, and I convolve it with H, I get back H. All right, well then, how about the following? If we convolve an image with an impulse, what would we get? Well, A, a blurred version of the image. B, the original image. C, a shifted version of the original image. And D, you have no idea.

10 - Convolution Solution

So you don't get any credit for d. You have, even if d is true, you have no idea. You get no credit for d, all right? The answer, actually, is the original image. And the way to think about that is, when I was doing the convolution or the correlation. There's really no distinction of what's the image and what's the filter, right? It's just an operation where I combine these things. So, before when I was convolving an impulse with a filter and getting back the filter. We were thinking of the impulses, the image, and the, and the filter as the filter. But you could have just thought of as the filter as the image, and the impulse is the thing you're doing the convolute, convolution with. And basically all the impulse is doing out, is pulling out the single pixel and sticking it in the results. So when you can convolve something with an impulse, you get back just the original image.

11 - Properties of Convolution

All right. One more thing. In order for all of this to work, we need a pr-,. a, a property called shift invariance. And shift invariance is basically that your operator behaves everywhere the same way. So I have a couple of pixels over here on the left and I apply my operator. If I had the same pixels over here on the right. Oh, this is my left, my right, your right, turn around, okay, or, look, never mind. If I have them up and down, right. If I have a couple pixels up here, and I apply a filter I get the same values if I have a couple pixels down here and I apply a filter. That means that I can shift things around and do the addition, and get my entire image back. As we said, because convolution or correlation are built on multiplication and addition, these are linear operators, making the whole notion of filtering a linear operation. And this means that convolution has some very useful properties. For example, it's commutative. All right. So, f convolved with g is the same as g convolved with f. Remember that whole idea about which is the impulse and which is the filter? It's also associative. So that here we, we wrote the associative property of convolution. We're going to take advantage of that in just a minute. It has an, a, a unit impulse. It's the identity. So that's what we talk about, that if you convolve any function with the identity you get back that function. And then here's a cool one. Differentiation of course is just the limit of subtraction and then a division, and a division is same as multiplication it's 1 over in that, in that case. So differentiation is a linear operator. Now you may have remembered that from calculus, right, so the derivative of a times f, where a is a constant, was just a times the derivative of f. And the derivative of f plus g was the derivative of f plus the derivative of g. So, differentiation is also a linear operator. And because of associativity we get this cool property here. That the derivative of a convolution is the same as taking the derivative of one of the elements and then convolving it with the second. And we're going to make use of that when we do edge detection and gradient finding in a little bit.

12 - Computational Complexity and Separability

I want to talk just a quick second about computational complexity. Mentioned it before. As we said, if your image is N by N and your kernel is M by M or we will call it W by W for sort of width. And it's easier to say W being different than N because [INAUDIBLE]. Exactly, okay? So the question is how many multiplies do we need? And we said before that we need N times N times W times W, or N squared W squared. Doesn't N squared W squared sound better than N squared M squared? Because when I say N squared M squared you have no idea what I'm saying if I say it quickly. N squared W squared works. And that can get sort of big. So there's a cute little property. Sometimes your main kernel, your, your filter,. Can be created by convolving a single row with a single column. And when that's true, you can take advantage of the associative property, in terms of how we do this. And this is what's refered to as a linearly separable kernel. So let me show you an example. So here we have a single column, and here we have a single row. And just think of 0s being around it. If I convolve this column by this row I would get out this new H which has 1 2 1 2 4 2 1 2 1 in it. Okay? So c convolved with r equals H. See, this is why we don't use my handwriting. Let's suppose we wanted to filter something with H. All right? So that would look like this. We have a function G that we're going to create by convolving F with H. But we said that C

convolved R is the same as H, so that convolves F. And because of the associative property, C convolved with R then convolved with F. In order to get our new function, is the same as C convolved with R convolve with F. And the reason that's better is that I can do two column convolutions instead of one square. So now instead of being W squared N squared it's 2 times W times N squared. And that can be, that used to be very important when computers were not quite as fast. But it's still reasonably important, because for example, if W is say a 31 by 31, that's a factor of 15 difference, all right? So that's more than an order of magnitude. Anytime you can do anything that buys you an order of magnitude for not a lot of money. You should go ahead and make that purchase. Because orders of magnitude are hard to come by. So here's a nice way of doing this. So when we do various kinds of smoothing, etc, often we use linerally separable filters and you can just apply them.

13 - Linear Operation Quiz

Here's another quick quiz. True or false? Division is a linear operator. Okay? A, false because X divided by Y plus Z does not equal X over Y plus X over Z. True, because X plus Y divided by Z is the same as X divided by Z plus Y divided by Z. And c, I have no idea.

14 - Linear Operation Solution

Okay, again, C might be true, but it's false. No, C is true, but you get no credit. The answer is true, in this case, because it's division by a constant. All right, so in this case, if I'm dividing by Z, so if I sum up X plus Y divided by the Z, it's the same thing as X divided by Z plus Y divided by Z. Much later we're going to be doing perspective projection, and you're going to end up having to normalize the X and Y values, by the Z value of a point. And then the Z is actually going to be a component of the element. And since that component can change, that's not going to be a linear operation. But in general, division is a fine linear operation because the last time I checked, dividing by two is the same as multiplying by point five, and multiplication is a part of the linear operation.

15 - Boundary Issues

One thing that comes up when doing filtering is what to do about the boundaries, because you might ask what happens when your filter falls off the edge. What happens when your filter falls off the edge? Meghan has been rehearsing this all week, outstanding. Well, basically it's undefined until you define it. All right? So you have to think about what size operation you want, so that can be illustrated like this. Okay? And here we're going to use a little bit of old MatLab nomenclature, they've changed it, because the old MatLab nomenclature actually makes it clear. So, here I've got a function f and I'm filtering it with g. And you might try to think about well, what's the size output that I want. And there's sort of three different possible sizes. One is when g just touches that corner, I start to get a response. So if I think of the center point of g as the reference, I would actually get a box that's bigger than the original function. Okay. The flip side is if I want to make sure that all of g is actually touching f, then I'm going to, again using the center point, I end up with a smaller output than the original. And that, that's used to be referred to as valid, since all those points were in fact correct. But the problem is when I filter a 55 by 55 image, I'd really want to get back a 55 by 55. I don't want to get back a 58 by 58 or a 52 by 52. What I want is referred to as the same. So here you see that we put that filter with its middle at the corners and we get back the same size. So the problem of course, is what's underneath these pixels here that are sticking out? And basically when you do filtering, you have to tell the system what you want there.

16 - Methods

MATLAB has a couple of ways of thinking about how to do this, right? And we're going to use, again, old MATLAB nomenclature, and then I'll tell you about the new one. So there are a couple

of different methods. The first method is called clip. Clip basically means I assume that that outside boundary is black. I then apply my filter, so here I can see my filter. And when I pull out the image, you can notice that this thing has gotten kind of dark at the edges. And that makes sense because that black has leaked in, all right. But that's referred to as clipping. Another method is called wrap around. It's kind of a weird method. It has to do with some Fourier analysis that we'll talk about later. Basically it says that I assume my picture continues and wraps around. Which means that you see this stuff filled in here? That's the stuff that comes from this side of the picture. It's easier to see up here, right? These, these red peppers came from down here. And the, the yellow straw comes from over there. All right? And that was under assumption that what you were actually looking at was a periodic signal. So if you were looking at a periodic signal. The next thing that would happen would be the stuff that was at the start of it. Well, in filtering, in image filtering, that doesn't work so well. Here, I'm going to apply the filter, and then I'm going to cut back to the original size image. And you'll notice that there's some red stuff here. Where did that red stuff come from? Well, it actually came from the bottom because it got wrapped around. There is a method that's called copy edge or replicate where you just basically extend out, right? So I just extend out the same value. And then I run my filter and then I pull out my picture and that's reasonable, okay? The replicate method is a, is an easy one, and it gives you a reasonable result. It basically says that the image sort of stays the same. Now the problem is is that the statistics, of course, are different, right? Because you had this nice varying image, and then you make everything be sort of the same going out. So, another method is called reflection. Okay, are sometimes called symmetric. And what that does is you, you reflect the image out. And here, I just did that. In fact, here let me draw right here. Here's the reflected edge. It's actually, if I erase it, hard to see that edge. Right? Because I basically re, I, I take the image and I fold it back out again. All right. So, then I take that. I apply my filter. And then I pull out the image, and it actually does a pretty good job. So, often you either want to do what's called replicate, or copy, the edge or reflect across the image. In new MATLAB these are expressed using this function called imfilter, which comes from the Image Processing Toolbox. And you can do you put in a value. Puts that value in, so we put in 0, it's like clipping. You can wrap around with circular and then the copy edge is replicate and reflection is symmetric. And typically, you'll use either replicate or symmetric.

17 - Explore Edge Options

So which is your favorite option to deal with edge issues? Load up the image package, read an image, create a gaussian filter. Remember gaussians are special. Now when you apply it, specify an edge parameter. Passing in zero is equivalent to the default. You can see the black seeping in along all the edges. This is because we passed in zero. What happens if we put in some other number? Try it out. What about circular? If you look closely, you'll see the green seeping in on this side, and a little bit of red on the opposite side. Replicate. Replicate is not too bad. No obvious effects. And lastly, symmetric, or reflect across edge. Not bad either, actually not too different from replicate. Feel free to experiment with these different options, filter sizes, and sigmas.

18 - Reflection Method Quiz

The reflection method of handling boundary conditions is, in filtering is good because. A) the created imagery has the same statistics as the original image. B) the computation is the least expensive of the methods. C) setting pixels to zero is quick. Or d) none of the above.

19 - Reflection Method Solution

Well, C is true but irrelevant. B is false but irrelevant. A is the correcting thing. The imagery that you create by doing that reflection has the same statistics. And I guess I should have also written, no obvious edge induced. Right? When I did that reflection, it was actually hard to see where those edges were.

20 - Practicing with Linear Filters

So the very last part of this lesson, we'll do some simple analysis about what different filters we'll do, and then we'll do more sophisticated filtering later. Here we have a picture, there's the original one on the left, somebody's eye, maybe you know whose eye it is, and I convolve it with that filter that's an impulse, what do you get back? Well what do you get when you convolve an image with an impulse, you get the original, and there it is, it's no change, 'kay so when you filter with an impulse, all right. So here's a slightly interesting one, what happens if I've got an impulse but it's actually not centered at the reference point but it's shifted over right, by one. Well what's going to happen is, when I place my filter down the, in the middle, it's going to go get the fill, the pixel from the right, and put that down at the reference point, then I move it over, and I get the right, now whether I get the one on the right or the left will depend on whether I'm doing correlation or convolution, remember correlation I just move it around, convolution I flip it over and then move it around, but what you're going to end up with is a shifted image, so in this case you'll shift to the left if you're doing correlation, so by shifting the impulse you get a shifted function, and that's because the idea is that the center coordinate here is 0,0. All right, what about this? So now I've got all ones divided by 9 so it's so it sums to 1, what is that going to look like? Well, we've already seen that, right, that's just a, kind of a crummy smoothing filter, it's a blur, all right, so I just get out the blurred eye. All right, now comes something really cool. All right? What if my filter, is actually the combination of these two? Okay? This is essentially twice the impulse, minus the blur. All right. Now, this is still all linear, so, the output of applying this kernel, where the kernel is made up of sums and multiplies, can just be done by taking the sums of the original two outputs, and that would look like this. And you'll notice, I'll show you a better example in a minute, that it's kind of sharpened up the image, it's accentuated the differences, and this is, drawing something like this, so here, this is filter is called, a sharpening filter, and it's got these little parts that have to do with the, the minus, and then the part in the middle. And if I show this applied to the whole filter, okay, you'll see that, and you, so, now you know whose eye it is, there is Einstein's eye and you'll, and if you take a look on your screen, you'll see that this one is a good deal sharper than the, than the previous one.

21 - Unsharp Mask

You have time for a quick story? Okay, sure. Anybody here ever use Photoshop? Raise your hand. In the older Photoshop or even currently, they have something called the unsharp mask. Okay? And when you apply the unsharp mask, what does it do? It makes it sharper. Now you might ask, why would you call an unsharp mask something that makes it sharper? Don't ask. Okay. In the old days, like when you did things in a darkroom, you were playing with light and you, chemicals, etc. But you still actually able to add and subtract light. So those of you have ever seen film knows that film were made out of negatives, right? And so that where it was dark, it would let out hardly any light and where it was light it would let a lot and then the, it would reflect on, it would be projected onto paper. And then whenever a lot of light hit the paper, it would become black and that's why you had to make it be a negative. Some very clever photographer figured out the following. Let's suppose I take a negative and then I put a piece of wax paper over it and I put another piece of film underneath that. Okay? And I turn on the light for just a split [SOUND] second. I will get a blurry negative of the original negative. Okay? Right? Because it, it becomes a negative, it's a piece of film, so I, I put the, the good negative, the sharp negative piece of wax paper and then another negative and there are other ways of making it. But the idea is that I would get a blurry negative of the negative, okay? So if the negative were all white, this negative would now become all black. All right? It was the negative value. Then what you could do is you'd develop that film, et cetera. You could put that blurry negative of the negative in with the original film and expose it at one shot. And the math that was being done was exactly this math. We had our original and because it's a negative of it, we're subtracting off a slightly blurry version of the picture and that would give you a sharper picture. So that's how you did image processing with chemicals and film to make a sharper picture. And what do you think that extra negative that you made, okay? That was blurrier was

called, it was less sharp. It was the unsharp mask and it would be added to the original picture. And the unsharp mask would yield for you sharper picture. So now you learned something totally irrelevant, because you probably haven't even seen a piece of film. But it is why in software today, unsharp masks make pictures sharper. And now you actually know why, why the math works. Right? You're essentially subtracting off a slightly blurry version of the image.

22 - Filter Quiz

Here's a quick quiz. If a filter's coefficients don't add up to 1, they can be corrected by multiplying by the necessary scale. Right, you can just multiply them through by whatever it takes, all right? Or, the resulting image could be multiplied by the square root of that number after the operation to compensate for the horizontal and vertical application of the filter. A, true. B, false.

23 - Filter Solution

Of course it's false! Look, if the thing needs to be multiplied by, 2.5 to make everything sum to 1, then that 2.5 can be applied to the filter or it can be applied to the image. Not 2.5 squared, or something like that. because remember, linear scaling, linear multiplication, just passes right through these linear operations.

24 - Different Kinds of Noise

So we said, and we talked about this two times ago that Gaussian averaging was a reasonable thing to do. If the noise was independent in each pixel and was centered about 0. So that it like, the noise is created by a Gaussian process. So some were positive, some were negative, etc. And now we've talked a little bit about how they're doing the filtering. But there are other kinds of noise as well, and we need different kinds of filtering. Maybe not a linear filter. That's why it's at the end of this lesson. Just to show you something that's not linear, but an effective filter. Okay. So on the left here, we have that pepper image with a little bit of Gaussian noise added. And on the right, we have the pepper image with the salt and pepper noise. Different peppers. Peppers as in good peppers to eat with salt and pepper noise, which spicy pepper black. The question is what kind of filtering might you need to use on the right hand side, okay? So the way to approach this is to remember that our other assumption about images, right? Is that the underlying image change slowly around the pixel. So the idea is that if there's some sort of arbitrary value. That was put in some location as what happens in salt and pepper noise. How could we go about finding the value that we should replace that value with to make a better picture? Now remember, when we were doing the filtering, this blurring. We were essentially replacing the pixel values by the local average. And it was either a box average or a Gaussian average. And that was fine when the noise was not a huge amount of noise and it tended to go to 0. So by averaging, you tend to average away the noise and get about the right value of pix, of, of the pixel. But if a few sort of totally random values are being injected into your image. You really need to do something different. And as some of you know is when I have sort of these very spurious values. An interesting way of, of, coming up with sort of the appropriate middle value, is not an average, but a median.

25 - Median Filter

Here I'm going to show you a median filter and image processing. At the top here, we have a chunk of our image and notice that in the middle is a pixel that probably is not right. Probably is a piece of salt. All right. Because there are all these other lower numbers and then poof a 90. So the question is, what should we replace that with? If we just do the regular filter, then we would take the local average of everything, including the 90, and stick that there. A better idea is to do a median filtering. So we sort all those pixels. Whoops, I left out the 10. There we go. And we pull out the middle value, the 27. And we replace that here. So now you see the 90 is totally gone. The 27

happens to be replicated. We don't care. But the idea is that we replace that pixel with this median value. So before you're gone, we ask some questions about that. First of all, what's kind of nice is every value we put in the picture was present locally. Especially if it's odd. If it's even, then it's the average of the two, but if it's an odd number of pixels, the median is one of those pixels whose value is somewhere around me. So I'm not introducing any sort of really weird values. So that's good for when I have these weird spikes, like this salt and pepper, but here's a question for you. Is it a linear operation? What do you think Megan? No? Good. Yeah. Right. No, look median is not a linear thing right. I add in a couple different things. I summed them up and the median can move however the median going to move. It's not going to behave nicely. So what is this look like in a real picture? So here we have our pepper image. So here is the salt and pepper version of it. Again, different peppers. And here it is with the median applied. If you zoom on that you'll see that that's a really well cleaned up image. And you can see, this was a scan line, the plot of a single row across the image. And you can see all the salt and no pepper noise. And when I replace that with the median, I get a much nicer signal over here. And that's why median filtering works so much better for salt and pepper noise. So I'm showing you this because the median is really an example of a non-linear thing. It's also sometimes refered to as edge preserving. And the reason for that is shown on this simple drawing here. Suppose I have, I've got a signal that's supposed to be like this but there was a single spike added to that. If I take the median, I get this nice effect. This thing comes here, this, et cetera. If I were to take the mean or the average or a blur. I would tend to blur across that nice sharp edge and that's why median filtering is referred to as edge preserving. And so this is class of nonlinear filters that are useful but they're a little less well behaved in terms of the mathematics.

26 - Apply a Median Filter

Let's apply a median filter. As usual, load an image. Let's add some salt and pepper noise. Octave and Matlab have some great utility functions, like imnoise. You pass in an image, and then the type of noise you want, and a parameter. Here's some salt and pepper on the moon. This time we won't use imfilter. Remember that median filtering is a non-linear technique. We use the function medfilt2, which stands for median filtering in two dimensions. As you can see, a median filtering is really effective in removing salt and pepper noise. A little bit of blurring has occurred, but not too much. Try it for yourself.

27 - End

So this is the end of the, the lesson. Mostly what we've just talked about is so you understand more about correlation and convolution. Remember, correlation we take the filter straight, convolution is it's the 180 degree rotation so it's both left, right and up and down. Most of the time if we're using symmetric, when we're using symmetric filters it doesn't matter. But if we're taking derivatives, right, so the derivative in the x direction, there's actually a positive direction versus negative direction. That's when you do have to worry a little bit about correlation versus convolution. And most importantly, filtering is going to be an important tool, sort of, in your toolbox for how to deal with images and get out the elements that you need. And we'll do more of that over the next time. And then again, much later, when we're trying to compute features over images.

2A-L4 Filters as templates

1 - Intro

Welcome to Computer Vision. The last couple lessons we introduced the notion of filtering. We started from a notion of how to do like, noise removal as a way of applying a, a linear operator type. Mostly time being linear. And it was being able to do a local averaging. We also talked about the difference between correlation and convolution. Which is sort of just keeping a little bit of

bookkeeping in terms of keeping track of appropriate orientation of the filters. Now we are going to shift a little bit. And talk about the notion of filters as some way of getting at an intermediate representation of an image. What I mean by that is, you could imagine you take some image, and you convert the pixels into some, maybe an array again. Still an image as a function. But the idea is that that function now, is no longer just about the intensity. But it's about some property of the pixels locally in the image. And then we'll get to something that might be a very useful property in terms of representing our image. Before we do this, I need to put in one little caveat. Remember before, we talked about things being linear, so if I was correlating an image, filter with an image. If I just multiplied that correlation filter by some constant. Okay? Then the whole value would get bigger. So if I were comparing the output of an image correlated with filter one versus an image correlated with filter two. And I want to be talking about sort of, how much filter one responds to the image versus filter two. It's important that in some sense, the scale, the multiplicative scale of those filters be the same. Otherwise the correlation will just be scaled by their difference. So by the way, from what I'm about to tell you, it's okay to gloss over the details. For those of you who are doing the problem sets, you're going to learn a lot more about this method that's referred to as normalized correlation. Here I'll just introduce it. Here's what we're going to do, we're going to assume when we do our correlation. First, we're gon, we're going to normalize our filters. By making them that, say, the standard deviation of all the pixels is equal to one, let's say. That would be one way of doing it, because as I scale that filter up and down, its standard deviation grows and shrinks. So I'm going to make standard deviation be no more, be exactly one. That's a problem if you've got a constant. Don't worry about that problem. We're going to try to make it so that its standard deviation is one. Then the other thing that we're going to do is, suppose I've got two images that are actually the same. Except one has just been multiplied by some constant. Well then if I correlate, even my normalized kernel, the one that's been multiplied by the constant, the total value is going to get multiplied also. So when we do it's called normalized correlation. We do two things. We normalize the filter, like we said. And then when we stick the filter down over a little patch of pixels, we scale that patch so its standard deviation is also one. And then we compute the correlation. And as we move the filter around, we change that image scale because the, the pixels are different. And that's what's referred to as normalized correlation. So for the rest of what I'm going to show you, I'm going to be doing normalized correlation.

2 - 1D Correlation

All right, back to our regularly scheduled program. All right, we're going to introduce this using one-dimensional signal and then we'll go to 2D. So here we have a signal, and it's a 1D signal, and here's a filter. And that filter, if you take a look, actually this filter here looks a lot like, oh yeah, something like that. And that's because I made this and I actually pulled out that chunk. Okay? But the system doesn't know that yet. And what we're going to do is we're going to do the normalized correlation of this filter with that signal. When you do that, you get a result that looks like this. And you'll notice its maximum is at this peak, this peak here in red, right? Right here, okay? And that's the location of where this filter came from out of that signal, okay? So the highest value is exactly when these things match. There are a couple of reasons to explain that, here's the one that I like best. Pretend for a moment that both of our signals were normalized, or were sort of shifted, so that they were about zero. So sometimes they went positive and sometimes they went negative, okay? And remember we've got the scale being approximately the same. because they scale up, so their standard deviation is one. If I take one filter and I lay it over the image and I'm going to multiply them and then sum them up. Remember, some of my values are positive and some of them might be negative. I realize this is the first time we've talked about negative signals. In fact, on the drawing here you'll notice it's zero in the middle. But if I've got positive numbers and negative numbers and I've got positive numbers and negative numbers in my filter, if I want to do a multiplication and have it come up as high as possible, when does that happen? Well the obvious intuition is, well whenever my pixels in the image are positive, I want my coefficients in my filter to be positive. Because positive times positive is what negative, Megan? Positive times positive? Yeah. Positive. Positive. And what do I want where the image is negative? I want the filter coefficients to

be negative also, right? Because negative times negative is what, Megan? Positive. Positive, very good. Yeah, she went to film school, but she's doing great. Anyway, and so if you think of sort of a filter as having, because we've scaled it, it can't go arbitrarily high and arbitrarily low. If you get that signal to be lined up exactly, so wherever it's positive the other is positive and where one is negative the other is negative, you get the maximum value. There are a lot more sophisticated ways of showing this involving differences in distributions and things like that. They all basically will show you that the maximum value you can get. The product of these is when they're exactly the same.

3 - Matlab Cross Correlation Doc

We can do the same thing in two dimensions also. So, here is a different pepper image. Right? This is a little chunk of the pepper image. Happens to have some garlic in it too. All right. Taken right sort of from here. I could do normalized correlation of this small patch on the whole image. And I can plot that. And when you do you get this answer. All right. You see this little spike here? That's what happens when the filter lands right on the spot in the image that it was taken from. You get this, again, the positives line up with the positives, the negative line up with the negatives.

4 - Find Template 1D Quiz

Okay. Let's write some code. We'll use one dimensional data to begin with. Given a discrete signal s as a vector and a template t our job is to find where that template occurs in the signal. This is just like finding a substring in a longer string. Here we are playing with numerical values which has some implications. Note that here we want to find the starting index where the template occurs. Let's print out the signal and template to see what we mean. Here we've used colon to generate the sequence of column numbers. Size of s, 2 is the number of columns in s. Let's see what the output looks like. The top row here is the column numbers. Can you find the template? Yes. It is located here. Starting at index 5 in the signal. So this is what I want you to do. I want you to write a function find*template*1D. That takes two parameters, t and s, the template and signal. It should return the index, where template is found in s. Once the index is found we should be able to display it. Let me get you started with some template code. By the way, you most likely need to use the image package. Make sure you test out your function with different signal and template values. When you submit your code your function will be evaluated against other test sets.

5 - Find Template 1D Solution

As you might have guessed, we want to use normalized cross correlation. This correlates the template t over the signal s and returns a normalized set of values. You can find out the maximum value in this result set using the max function. This corresponds to the best match location. When given two output arguments max can actually return the index of the max value as well. Isn't that what we wanted? Let's look at the result. So we see that the returned index is seven. But the correct answer should be five. What happened? The correlation function actually starts comparing from the first point of overlap between the template and the signal. Here the template has three elements. So comparison starts where the third element overlaps with the first element of the signal. And the window moves from that point forward til the end and beyond. The operation stops at the last point of overlap. Let's look at the values of c to figure out if this is really the case. We use the same trick here to display column numbers, and let's comment out other output code for clarity. As we can see here, we have 16 values. Our signal had 14. The first two coefficients are from positions where the template was partially outside the signal. Therefore, index three, which is the length of the template corresponds to the first position in the signal. We have to account for this offset when returning index value. One way to think about this is that the index value returned by max is the rawIndex. In order to get the correct index, we need to subtract the size of the template and add one. Let us restore our original output lines and look at the output. Now we have the correct index, five. This is the starting position of the template found in the signal. Let's try a different template. All right. As

expected, the template is found at position seven. Note that we are not constrained by templates that have exact matches. For instance, we can try a template that only partially occurs in the signal. Quite impressive. The template is found in the correct position, even though there is a mismatch. A partial mismatch would result in a correlation coefficient that is less than one. But it might still be the maximum across all other elements.

6 - Template Matching

Let's continue exploring this idea of using filters as templates of what we want to find in two dimensions. Okay? So a template is a thing that you give me and you say I want you to find something else that looks like this and we're going to do that through a filtering. And here's a nice example courtesy of Christian Grumman as well I believe. Although I'm not sure she made, if that's one these, by the way this is early in the class. You're going to see a lot of slides that have somebody's name next to them, because I've got them, the. And then what I later discovered is, many of these slides came from other people, and from other people. So the name that I put down is the name that I was able to track it towards. And the rest of us on the internet we're just making our way. All right. So, here's a very simple example. So here we have a scene on the left and we've got this template on the right. All right? So this is the think that we want to find. And you look over here and you say, oh I see where it is it's, it's, it's, it's right over here. So what we would like is our system to spit out something like this. Say aha, here's my little green box. Well what we can do is we can actually do this through a normalized correlation. If I take that masked kernel template and I do a normalized correlation with the image on the left, what I get is a map or correlation image that looks like this. Okay? And you'll notice. That, where black lies over black and white lies over white, you can think of that as positive and negative, so I get some brightness over here and some brightness over here. But my really bright spot is right there because that's the template having landed in the right place. So that's how you use correlation in order to do this detection because now we're thinking of filters not so much as doing an averaging or a blurring but actually being a template that we're looking for.

7 - Find Template 2D Quiz

I bet you're itching to do this yourself. Here's the function I want you to define, find*template*2D. Takes a template and an image as input and returns the x and y coordinates where the template is found in the image. I have just the image to test this out. I'll cut out a glyph from this tablet, which will serve as our template. Remember, you want to find where the top left corner of the template occurs in the image. Also note the order of the return coordinates, y x.

8 - Find Template 2D Solution

As before, we use normalized cross correlation. Since the image is a two-dimensional matrix, we need to be careful when finding the maximum. Simply writing max of c would compute the column wise maximums. Indexing c with a single colon flattens it. Therefore, max of c colon would compute the image wide maximum. We use a slightly different trick to find the location where this maximum occurs. The find function directly returns the row and column, that is the y and x coordinate of the desired location. Remember that these are raw indices into the matrix of correlation coefficients. We need to account for the offset by subtracting the size of the template along the appropriate dimension and adding one. And that's it. Let's see what y and x values we get back. All right, 75, 150. Isn't that where we cropped out the glyph from? Indeed, those were the top and left coordinates. We can do some additional plotting to make the result more intuitive. Here we first show the original image, and then we plot a red plus on top of it where the template was found. Since this is not a fully interactive environment, we have limited plotting capability. In the local installation of MATLAB, or Octave, you can a lot more, like draw a rectangle where the template is found.

9 - Wheres Waldo

So those of you that are old enough will remember, and especially if you're from the United States. You'll remember that for a while there was this whole bunch of books out there called Where's Waldo. And they would give you a scene that looked like this on the left, and your goal was to find. Here's Waldo right here, kind of blurred out, hard to see, that would be my template maybe. And I want to find Waldo. And if you take a look at this, can every body see where Waldo is? Well, he's right there, okay. We go back there, okay, you can see him. There he is, all right? So, the question is, how could you do this doing the template matching? Well, we would do the same thing before. We take our image, we take our little template, and maybe you can see over here on this Correlation map, all right? There's that bright spot right there. And you'll notice, that everywhere else it's hardly bright at all. But right there, that's that bright spot.

10 - What is it Good for Quiz

Besides looking for Waldo what is template matching good for? Lets look at some cases. Can you find lines in an image with templates? Including vertical horizontal and diagonal lines. How about faces? Some small, and some large. Icons on a computer screen. Can we use template matching to find cracks and fissures in metal components? This is actually something that computer vision is successfully used for, in real assembly lines. Mark the situations in which you think template matching can be fairly successful.

11 - What is it Good for Solution

Since lines in a image can be in different lengths and orientations. You cannot come up with a single template to find them. Although you can use a bunch of templates at different orientations to find parts of lines. There are much better options for this purpose. Template matching is in general not suitable for finding arbitrary lines. The case for finding faces is arguable. Again, due to the possible size differences, you cannot come up with a single template. Moreover faces can be front-facing or side-facing or rotated at any arbitrary angle. Either answer is correct, although as we will see, there are much better ways of doing this as well. Finding and recognzing icons on a computer screen is a fairly predictable task. This is because icons are usually shown upright and their size is from a limited set. Therefore, template matching is suitable in this case. Cracks and fissures hardly have a predictable shape. Therefore, they cannot be detected by template matching. Are there any other good uses you can think of? Or cases where you are sure that template matching is bound to fail? Share with your peers on the forum.

12 - Non Identical Template Matching

And in fact that's shown here. So it says, you know, what if the template in the scene is not identical to some sub image. So here we have a scene and we have a template, well it turns out that the best location of this template in this scene is right there. It actually finds the same car. Now you have to worry about scale and orientation and all that stuff, stuff that we're going to talk about later in the class. But even though the cars are quite different the fact of the matter is it has sort of bright stuff here and dark brown things there. This has bright stuff there and dark brown things there. So as a template it's not awful. So that's using these filters as templates

13 - Normalized Correlation Quiz

Here's a question for you. Would this method actually work for finding Waldo in most pictures? A) yes, normalized correlation is a powerful thing. B) no, we don't actually have the right template. C) partially, explain how.

14 - Normalized Correlation Solution

Whenever I ask you that the answer is always partially explain how and here's why. Yes, normalize correlation is powerful if you happen to have the template. And in here for Waldo, we have the exact template. How do I know? I actually cut it out of this picture to begin with, then I made a template out of it. But if I don't know what Waldo's going to look like. All right? So he's got that same dumb hat and the striped shirt, but he could be bent over, et cetera, whatever. I don't have the exact right template. But now think about stepping back and blurring it just a little, so there's kind of red stuff up here and reddish, whitish, pinkish stuff down below. So your template is sort of close. So you might actually be able to get some candidate examples.

15 - End

And actually that ends this lesson. Essentially we're using filters to localize interesting areas in the image that have some sort of property that makes the filter respond well. So what we're going to do going forward is we'll use some special filters to compute some functions, like smoothing, but also to sort of find strong responses just like the templates. And we'll do that actually in the next couple of lessons.

2A-L5 Edge detection: Gradients

1 - Intro

Welcome back to Computer Vision. The last lesson, we started talking about filtering as a way of finding a specific pattern in an image. We talked about template finding. And the idea was that given some pattern, correlation or convolution would be some way of finding that pattern in the image. And the assume, we assumed that the, that the goal thing that we were looking for was given to us. And then there was also this idea that, well, maybe we could just take the locations where this pattern was found. And that became sort of a description of what we were looking for. You could even think of that as describing the image, although we didn't talk too much about that. But again, that's the idea that you have, you know what the important thing is that you're looking at. But what about generic images? Ones that you don't know in advance. What's going to be in the picture. What are some good, the word we sometimes use is features, what are some good things to try to find in the picture, that might capture a lot of the information there, and sort of give you the essence of what you need to work on.

2 - Reduced Images

So to get some insight into this question, I want to show you some very very greatly reduced images which are still going to be meaningful, that is, you know, what is going on to me in use, so normally when I do this in class, what I do is I say raise your hand when you can tell me, if you can tell me, what's in the picture. So I would show then this. And everybody would raise their hand, and they would say it's some goofy looking lion. All right? And I would show them a picture like this, okay, there, there's an elephant. There's a really cool, old car. There is a, something I'm never going to own, a private jet. But what you can see about these pictures is there's actually not a lot of pixels being used in this, in these pictures of different grayscale values. Instead, somebody has just sort of drawn in the edges. And yet, that conveys to you a lot of the information. So the idea is that, in some sense, edges in an image seem to be important, they seem to convey something about what's going on.

3 - Edges

Let's talk about edges in just a generic image, and how it is that edges come to be within a picture. So what you can see here is this is some ancient picture somebody drew out, that there are edges that happen for a variety of reasons, right? So here's one right there, okay. And that's at a depth discontinuity. First we're looking at the bottle, then we're looking at the background. Here's a shadow edge. Of course, shadows aren't really that harsh but, go with me further. There's idea that there's a discontinuity in the illumination. Up here, from this top to there, is a discontinuity in the surface normal. That is the shape in one way, and then the [INAUDIBLE] the shape changes and it's pointing the different direction. And then finally, edges here can do to be to essentially color discontinuity, so we go from white to black on the same surface, same illumination, same shape and everything, but the reflectance is actually different. So that's what edges look like. We can look at them in a real picture. I think this is UT Texas, which would mean that again this was stolen from Crystal Robin as well. Borrowed. All right, so in a real image we have depth discontinuity. So the side of the building, we have shadows. So there you can see the shadow being cast by the sun on the front. What's interesting is we have textures. So if you were to blow up this area. And we'll, maybe we'll take a look at some texture later. Just from the, the texture or you can think of this as paint things are on the surface, you get different edges that show up. And then finally, the top ridge of that little cement thing, the edge around the bridge on top, that's like the top of the bottle cap. Same stuff, same color, but I get a discontinuity in surface orientation. So, our goal for today and, and next time also, is to find the edges in image. Or to put this another way, we want to convert an image, which is a function of x y, into a reduced set of pixels or curves et cetera that are the important elements of the picture somehow. And the idea is so, you know, that it encodes the changes that matter to you.

4 - Change Boundaries Quiz

So here's a simple question. Edges seem to occur at "change boundaries" that are related to the shape or the illumination. Which is not such a boundary? An occlusion between two people. So I'm here and somebody else is there, there's an occlusion between me. A cast shadow on the sidewalk. A crease in a piece of paper or a stripe on a sign.

5 - Change Boundaries Solution

Well as we said before, in the picture of that bottle, this last one, a stripe on a sign, that's not actually due to a change in any of the physics of the, the shape or the illumination. It's actually due to a change in the reflectance function. Okay. And we'll talk about that later when we talk about reflectance functions. But the idea is that it, it's not part of the nature of the geometry of the shape of the object or of the scene. But of having to do with the, the reflected texture. Or a reflected color

6 - Edge Detection

So the question becomes what do we do? How do we compute? To figure out that a pixel in an image. The pixel is some location x, y is actually an edge pixel. That is that it lands on an edge or is part of an edge somewhere. To think about this, we have to return to our idea of image as function. So you remember here's our comedienne. And this was two different representations of the same function. One just happens to be shown to you as a picture, so you're used to seeing that although it's pretty scary. And the other is shown as this height map, where the height was proportional to the intensity. And the thing that's clear is that these edges, right? So this edge by the side of her face there. Is really a steep cliff by the side in, in that function. Okay. And what we're going to look for fundamentally for edges are going to be these steep changes somewhere in that image function. So it sounds pretty simple. Okay. In some little local neighborhood. Can I determine that I'm in a, at a location where there's going to be a very strong change? Doesn't seem so hard. Well, there are two

things that we have to sort of answer in order to be able to do this. If we're trying to look at a neighborhood with strong signs of change. We had this question of well, how big is a neighborhood? And then what do you mean detecting change? How much of change is change? And, and how are we going to determine exactly where that change is? So, so here's an example of a function and the idea is that, you know, probably the edge might be thought of as like that. Our first problem of course, is we don't get to do that. We have to pick the pixels that we say are the edge. Right? We don't get to say it's between, well, we could say, it's between pixels but we, we tend not to do that, right? So the idea here is that the 80s are on one side and these 20s and 30s are on another side, so we have to try to go about finding those.

7 - Derivatives and Edges

Change is going to be about what when we talk about functions? Well change is going to be about derivatives, all right? And this is why we're going to be able to talk about derivatives with respect to our functions. So here we have a very simple image, and I'm going to show you a plot here of a horizontal row, so if we just take a row. And we would just plot it. You'd see it starts out high and then it comes down smoothly and then it goes back up again. All right, so this is a nice smooth image. Well as you remember, when you were taking a look at change back in calculus class, you would take the derivative. If I apply the derivative to that function, I would get a function that looks like this. So this is the first derivative of that scan line, and these edges, which are these edges, correspond to these extrema of the derivative. That is, here's, somewhere down here is where the maximum negative slope is. And somewhere over here is where the maximum positive slope. This is the maximum negative. That's the maximum positive. So, finding edges is going to have something to do with finding peaks in the derivative. So how are we going to find these extreme in the derivatives? Well, the same way we found other things, right. How do we find Waldo? We ran some filter, we applied some filter and looked for its response. So we're going to filter our image with the appropriate word is operator in order to find these peaks in the derivatives.

8 - What is a Gradient

So let's talk about differential operators. Okay, so a differential operator, when applied to an image, it returns some derivative, okay. And we're going to model these operators as filters, as masks, as kernels, that are going to compute that image gradient function, all right? And then we're going to threshold this gradient function to select the edge pixels. All of which brings us to the question, what is a gradient? All right, so, here you gotta go way back in your mind there somewhere, for those of you, and I guess I, I make to little, I mean, many of you are heavily involved in the, in the, getting your degree, or your courses and you've been taking all these math courses. If so, I apologize for being so pedantic. For those of you who haven't looked at a calculus course in a while, we're going to have to remind you a little bit about multivariate calculus, okay? And multivariate is when you have a function that was a function of more than one variable. So for example an image is a function f or i of two variables, x and y. And when I have functions that are a function of more than one variable, I can take what's referred to as a partial derivative. The derivative in the x direction, or the derivative in the y direction. And the gradient is the vector that's made up of those derivatives. So let's take a look at this for images. So I'm going to write this as the gradient of an image. And this is that operator that I'm going to use to show the gradient. And an image is just a two vector, made up of the partial of f with respect to x, the gradient of the image in the x direction, and the partial of f with respect to y, okay? So here's an image. And that, you can see in this image, it's only changing in the x direction. So its gradient would be whatever the change is in the x direction and 0 in the y direction. Likewise, here's an image that only changes in the y direction. Now in this case, I'm talking about, I was drawing the arrow in this way. We could either think of y as going, more positive as you go down, we talked about this, or you can just say that this is the direction of the gradient in the sense that in this direction is where it's getting brighter, okay. So in this image my gradient would be 0 partial of f respect to x, but I would have partial of f respect to y. And normally of course, I have change in both direction and that's the gradient itself.

It's partial of f respect to x and the partial of f respect to y. And it has both a magnitude, how quickly things are getting brighter. And also you can see there's this angle, which is sort of the direction of that gradient. And the gradient of a function is the direction of most rapid increase in intensity, and the magnitude of that vector is how much it's changing as a function of a unit step in that direction. Here we're just expressing all of this mathematically. The gradient is given by the two partials. The direction can be computed by just computing the arc tangent. That is, the arc tangent of the change in y over the change in x. Computer people amongst you, you should probably use a tan 2. So, that if the partial of f with respect to x is zero, that your machine doesn't explode. And then we can also talk about the magnitude of the gradient which is just the magnitude of these sums, and that's telling you how rapidly the function is changing. Which, of course, is clearly going to be related to something to do with edges because remember we said that edges, we get steep cliffs where things change tremendously, so we're going to look for large magnitudes of gradients when we're finding edges.

9 - Magnitude Quiz

So, quick quiz, what does it mean when the magnitude of the image gradient is zero? The image is constant over the entire neighborhood. B, the underlying function is at a maximum. C, the underlying function is at a minimum. And D, either A, B, or C.

10 - Magnitude Solution

It's a, b, or c. If you remember from calculus, right, when you have a function, when it gets to the maximum it's neither going up nor down right there, right? So the gradient, the slope is zero. So I have a function, so I have an image that's got a nice bright spot and I'm at the spot that's the brightest. Right? The gradient is going to to be zero. Likewise, if I'm in a hole, and I'm at the part that's darkest, the gradient is going to be zero. And of course if the entire image is 118, okay? Constant, there's no change anywhere, or just over a neighborhood, also the gradient would be zero. So the gradient is zero in any of these three cases.

11 - Finite Differences

So look, so what what we were just talking about is fine for calculus class, where you got all those fancy little Greek symbols. And you probably, by now, wish you were paying a little more attention to those Greek symbols. But the question is, how do we do these things in a computer where we don't get to do continuous math? So to do that, we have to talk about discrete gradients. So, in continuous math land, the partial of f with respect to x is just this limit, right? So we, we move a little bit in the x direction, epsilon, subtract off the original one and divide by epsilon, and when we let limit go to zero, that becomes our derivative. But in discrete world, we can't move arbitrarily close. We have to take what are called finite differences. So what we're going to do is we're going to talk about approximating our partial by some finite difference. So the simplest finite difference is this one, right? I take one step in the x direction, and I see what the value of the function is, and I subtract off the original, and I divide how big was the step. Just one. So that just becomes this value. So we say it's approximately f of x plus 1 minus f of x, okay? And that's called the right derivative. You get it? because it's, takes step one to the right. Oh, wait. One to the right. Right, left, for those of you who aren't looking at this thing in a mirror. Okay. In order to talk about the right way to think about these derivatives, let's take a look at our finite differences a little bit more. So here we have a picture of the striped duck that David puts in his book very often, David Forsyth. And you can see here we have this, it's a zebra, okay? So we have this picture of this zebra which has all these stripes on it. And over here, we have one of those finite difference images. First question, is this the finite difference in x or in y? Well, let's take a look. Here I have these nice kind of almost vertical stripes, and as I go across, you can see that I get a bright value and a dark value and a bright value and a dark value, so in others words, as I'm seeing changes going horizontally, in x, I'm getting different values of whatever this finite difference is, but across here, I have hardly any

changes in x, and only changes in y, and you'll notice this is not a very strong signal there. Okay, so this is going to be a finite difference in x. One of the things that might be bugging you about this, is on the right-hand side here, I'm showing you an image that has positive numbers and negative numbers in it. So here's the deal. Normally, when we display an image, we make zero black and some other number white. Or we say some minimum value's black, and some maximum value is white. Oh, I didn't say it has to be zero, it has to be some minimum. So I could make like minus 128 be black and plus 127 be white. Or minus 20. And I could make zero be gray. So what you're seeing here on the right is what's referred to as a gradient image where gray is zero, white means positive, black means negative. This is sure to mess you up, in some of, doing some of the problem sets, et cetera, but remember, that the display of an image is just for convenience for me and you when it's a mathematical function like this.

12 - Partial Derivatives of an Image

Here's another example of partial derivatives. So here we have another animal that tends to have stripes. Although, I would not get as close to this one. So first question. Which one is the x and which one is the y? Well, let's see. Again, we have some vertical stripes here. And we see them right there really strongly. So that changes as you go to the x. And they don't show up over here so much. But here we have horizontal stripes, all right, that show up nicely. You probably figured out by now that on the left hand side we have the partial with respect to x. On the right hand side, the partial respect to y. And in fact if you take a look at this, you'll see that, and I'm going to make sure I put these in the right way, is that when I get to this, this edge right there, which is this edge right there. That as it gets brighter, that value goes high. So the filter that I'm using here is this minus 1, plus 1 correlation filter. I take that value at my right, subtract off the one from my left and that's the value. And now, correlation versus convolution matters. Because if I swapped it around, things would be different. On the y side it's also going to be plus 1 or minus 1, but I write it like this. And the reason that I do that is, one of these is the filter that you would use for computing the y gradient. But whether it's the one on the left or the one on the right will depend if whether in your universe you make y go up positively or you make it go down positively. And that's a choice that you get to make. You can put the origin of your image in the top left hand corner like computer scientists tend to think about them or in the bottom left hand corner like mathematicians for thousands of years have done that. Only computer scientists would screw with that, but we did. So you, you'll probably have to figure it out which way y is in your world and then apply the right filter. By the way, I can take the magnitude of this. That is just take the, the sum of the squares and take the square root, and I would end up with a picture that looks like that. And that should start to give you a hint as to where we're headed because that looks an awful lot like an edge image.

13 - The Discrete Gradient

So those are our discreet gradients, but now we said we wanted an operator. Right, we want to mask our kernel that can implement these gradients. So here's an example of an operator H. And I'm going to use three rows, and I'll tell you why in just a minute. For now we only have these two columns. So the question is, is this a good operator? No. No, it's not a good operator. Why is it not a good operator? Well there's a couple reasons. One of which is there is no middle pixel. So it's hard for me to say that this pixels value because the question is what's special about the right. It's should be, I could also be able to go to the left. So, if I wanted to do that what might I do? Well, I might do an operation that looks like that. I'm going to go to the right here, and the left there. Now, you might ask yourself why is it a plus a half and minus a half. Why is it? No, ask yourself, don't ask me. Well, okay, here's why. It's the average of the right derivative and the left derivative. And what I mean by that is, the right derivative would be a plus one here, and a minus one there, okay. The left derivative would be a plus one here, and a minus one there. If I wanted to average them, I would add them and divide by two. So, I add them I get minus 1, 0, 1, and then when I divide by 2. I get minus one-half and plus one-half, with a 0 in the middle. So that's why this thing that goes

plus a half to the right minus a half to the left is the average of the left and right derivatives. Get it? Cool.

14 - Sobel Operator

People have been doing derivatives and edges for a very long time. The most classic one is referred to as a Sobel operator, named after Leonard. I have no idea. It's named after some guy named Sobel. And the Sobel operator looks just like what I was showing you. But instead of one-half and minus one-half, it's got this weird thing where it's doing these eighths. And you can see that it does, not only plus 2 minus 2, which you would then divide by 4 and you get the same value. But it also does a plus 1 minus 1 on the row above me, and a plus 1 minus 1 on the row below me. And the idea here is, that if I'm taking a derivative. Remember how we said before we assume that our images are sort of locally this smooth, that is they change similarly? So the idea here is that if I'm going to compute a derivative at a pixel, I won't look just left, right at me, but also look nearby me. And then to normalize this thing, you have to divide it by 8. By the way, having just finished, almost finished teaching the course here on campus, we had a problem set where people were having a heck of a time that they hadn't had a problem with before. Turned out, somebody had discovered that MATLAB has a built in function called imgradientxy. And they said, could we use that function to compute the gradient? Since it was for part of a problem set that was more advanced and they had already done gradients, I said sure. And all the sudden people are having all sorts of problems. And the reason was, if you look up imgradient, it tells you that it applies the Sobel operator by default. Well here's the Sobel operator. Looks great, but you know what? It doesn't divide by 8. So all their gradients were scaled by a factor of 8 and it totally screwed up the entire class. A hundred people got screwed up because of this. So, now we tell them you can use im gradient, but you better divide by 8. Oh, by the way, the y one is here as well, and in this case y is positive going up. Remember, it can go in either direction. Alright, so the Sobel gradient would just be made up of the application of this sx and sy to get you these values. And the magnitude is just the square root of the sum of the squares. I should have said that gx is the application of sx, gy is the application of sy. So the magnitude is just what we did before. And here is the arctan2, the atan2 that we were talking about in order to get the gradient. So here's an ancient example. It's so ancient I know because that's X Windows, so that even predates when most of you were born. And here you have a picture on the left that's an image. Here is a gradient magnitude, so you just apply the Sobel Operator, take the sum of the squares, square root of. And then by the way you could just threshold it. And you'll notice two things. One, it's not an awful edge image and two, it's not a great edge image. All right? So we're like part way there. We're partly towards getting that done.

15 - Well Known Gradients

There are lots of well known edge operators. There's the so, Sobel. Then Prewitt and Roberts which you can see does these different kinds of ways. Matlab basically understands all of these. And in fact, in Matlab there's this cool little function now called fspecial, which will make filters for you depending upon the name that's in, might be in the imfilter toolbox only or, or basic Matlab. But the idea is that you can give it Gaussian, you can give it thing, you can give it Sobel. And if you give it Sobel, what it'll do is it will reply back this operator, okay? And in this case, by the way, turn everything into doubles in Matlab, if I apply that to some image, all right? So here's the double of my image, I apply the filter, and then I display it, and I use a gray color map. And you can see it gives me that gradient image, okay, and that's the y gradient only. It also will give you back the x gradient.

16 - Compute Gradients Quiz

Quick quiz. It is better to compute gradients using, convolution since that's the right way to model filtering so you don't get flipped results. B) correlation because it's easier to know which way is

which which way the derivatives are being computed. C) it doesn't really matter. Or, d) neither since I can just write a for-loop and it'll compute the derivatives.

17 - Compute Gradients Solution

Well, this is a tough one. The answer is either b or c. Look, as long as you keep track of what's going on, it doesn't matter. You can do whatever you want, which case it would be c. Remember before I showed you those correlation operators that I'm doing correlations, because I want to know which way is the right way? So the right versus to the left. So I think when you're doing filtering for gradients, doing correlation probably works better for you. And by the way, so in the old days, we used to have to either call correlation or convolution explicitly in Matlab. Now we tend to use i m filter, as we've talked about before. I m filter by default does correlation. So if you want convolution, make sure you tell it. All right? So you just have to keep track of that.

18 - Gradient Direction Quiz

Let's take a closer look at gradient directions, especially how they're computed and represented. How about we use an image that has clearly defined edges and distinct angles, like this? Note that we convert the image to double type after reading and scale it down to a 0 to 1 range. This makes it easier to track down numerical issues that may crop up. It also makes it convenient to display, because imshow assumes a 0 to 1 range for double images. MATLAB and Octave have a direct way of computing image gradients. Imgradientxy is the function, it returns a pair of matrices, the first one being the gradient in x direction and the second one in the y direction. I'm pretty sure that the default filter used is Sobel, but you can also mention it explicitly. Let's try to visualize the gradient images. But note that imgradientxy does not normalize the gradient images therefore we must scale them to the appropriate range. Now why do we need to add 4 and divide by 8? To understand this, let us consider this a bell operator which looks like this. Now what happens when this filter interacts with an image, or an image region, that is black on the left hand side and white on the right? All these negative coefficients, they get multiplied by 0 and cancel out. Whereas these positive coefficients get multiplied by 1 and some to 4. So the total response of the filter at that position would be 4. Similarly, an image region which has white on the left hand side and black on the right hand side would result in minus 4. So our gradient values are in the range minus 4 to plus 4. If we add 4 to each of these values, the range shifts to 0 to 8. Hence, dividing by 8 gives us a 0, 1 range. Enough with the math. Now what does this look like? As expected, in the x gradient image, you see the vertical edges show up. Similarly, you can view the y gradient. This time the horizontal edges show up more clearly. Note that on your local machines, you should be able to simply pass in the expected range and let imshow do the scaling. If you leave out the limit values and pass in an empty vector, then imshow will scale the image based on the actual minimum and maximum values found in the image. This can give slightly different results. For instance, say the minimum magnitude found was minus 3.5. This behavior is similar to what imgsc does, although that has other options for setting the color map, et cetera. Okay, now that we have the x and y gradients, we are interested in computing the overall gradient magnitude and direction. Fortunately there is a function to do this in one step. The magnitude return is the Euclidian norm of the gradient in x and y directions. As we saw earlier, each of those could have an absolute value of 4. So the total magnitude values can go from 0 up to 4 times root 2. And that's what we used to scale it by. This is what the gradient magnitude looks like. The edge pixels are not super bright, which kind of indicates that the edges were not as sharp, but that's okay. The edges are still distinctly visible. Remember that the gradient direction is an angle computed as the tan inverse of y by x gradient values. The result is returned in degrees ranging from minus 180 to plus 180. Where 0 degree corresponds with the positive x-axis, and increasing angle rotates counterclockwise. The right edge has gradient pointing at 0 degrees. Gradient values in the plain areas of the image are undefined. Since they have to be set to some number, they're also 0, which is why you can't see the gradient values along the right edge. As I mentioned, angles go counter clockwise, so this is 45 degrees, 90, 135 interestingly, gradient pointing to the left is at 180 degrees, which is the same as minus 180.

And here are the other angles for good measure. Now I want you to write a function that finds pixels with desired gradient direction. The function should be called select_gdir. The first couple of arguments are the gradient magnitude and direction images. The third is the minimum magnitude value we want to consider. This is to filter out noisy pixels that can have random gradient directions, but usually low magnitude. The last two arguments specify the desired gradient range, low to high. For instance, here we want to look at gradients in the 45 degree angle, with a plus minus 15 margin. Note that you need to return a binary image that can be displayed with imshow. Here's some code to get you started.

19 - Gradient Direction Solution

Okay, the solution is actually a one-liner. Firstly, we know that we want pixels that have at least the minimum magnitude specified. We can use relational operators to directly compare matrices with scalar values. This is essentially a thresholding operation. Next, we use a pair of comparisons to say that gdir should be between angle*low and angle*high. Notice that the results of the three comparisons are combined using the element-wise and operator. And that's it. Let's see which pixels have a 45 degree gradient. Seems to work as expected. Notice that the line is thinner than expected. Not all the pixels with a gradient direction of 45 degrees have been included because the ones with low magnitude have been filtered out. Similarly, we can look at 0 degrees. Here we specify the range as minus 15 to plus 15, 90. Here's a negative angle, minus 135. Note the low and high values. 180 degrees can be tricky. Something for you to think about.

20 - But in the Real World

So far, so good. Except that what I just showed you won't work. [LAUGH] And let me show you why. So in the real world. So here I have this little function, okay? And so this is f of x, right? And it's got some stuff, and then it takes a jump and it's got some more stuff over there. So that's intensity as a function of x. If I actually take the gradient of that function by just doing, say, a right difference. All right? What I'll do, I'll apply my derivative operator and I'll get this. And one might ask, where is the edge. Well, we can see here, the edge is really, whoops, I missed, the edge is right there. But in this mess, it's sort of hard to tell. And the problem, of course, is that we've added noise. And that noise has caused us to have positive and negative derivatives all over the place. So let's look a little bit more about the effect of noise on some of our derivatives and what we might do about them, okay? So, here we have again more of that striped duck, and here we have the gradient image. And what you can start to see is that as I add noise to the image, my gradient starts to fall apart. That is I'm starting to see this, I had this little bit of Gaussian noise and now I'm turning into these big salt and pepper noise. Oh it feels like salt and pepper noise, just these large spikes are happening within my derivative. So we have to handle that noise somehow. You probably know how to handle, how do we handle noise? We filter, we smooth, we get rid of it, right? We did that before. Come on, are you paying attention? Alright, so here's my image f. So now let me apply a smoothing kernel, h. Okay, so I'm just going to smooth it. So when I smooth it, I get this is h convolved. Doesn't matter whether you do convolution or correlation. H convolved with f, and you see now I get a nice, smooth function. So now that we've smoothed the signal, we can take the derivative and what you see is we get this nice, smooth peak. So, where is the edge? The edge is going to be at this peak right here, corresponding to that edge right there. So basically in order to find our edges, we're going to have to basically apply smooth gradients somehow and look for some peaks. Before we go ahead and do that, remember this whole notion of associate property and linearity etc. We can save ourselves an operation, because the derivative of h convolved with f, is the same as the derivative of h convolved with f. So what does that look like when we're doing this edge detection? It's the following. So here we have the same f that we had before. Now, I take my h, and I take the derivative. That now looks like this. When I apply that function directly to f, I get out the same value that I got out last time. Which has to be because of the associative property of linear operators, and derivatives and filtering being these linear operators. That also saves us an operation, because we can just take this filter and apply it and no more derivatives have to be done.

All right, so last question, okay. So, we have these nice peaks. We have to find the peaks. We have to find the maximum of that derivative. How do we find maxima? We take more derivatives. So we're going to have to take another derivative. So now, instead of just doing a single derivative, we're going to take a second derivative. So, before h was just a Gaussian, this was our first derivative of a Gaussian. If we take the derivative of that again we get this, what's sometimes called inverted mexican hat. And you'll see that, when we do it in 2D, right, because it's kind of like this sombrero shape but up and down, right? When we apply that operator to the image, we get this nice zero crossing with this strong slope. And that's what corresponds to our edge. All right. And this idea of a stong slope zero crossing. We don't have to find general maxima, we just have to find any place where the value is zero and that nearby has a strong gradient.

21 - Linearity Property Quiz

Alright little quiz. So which linearity property or properties did we take advantage of to first take the derivative of the kernel and then apply that and not have to do it the two steps before? a) associative, b) commutative, c) differentiation, d) (a) and (c).

22 - Linearity Property Solution

Actually well it's d. We, we took the associative and the differentiation is linear and associative can be applied and that's what allowed us to take the derivative of the operator and apply that to the whole image. We'll talk a little bit, I guess next time, why that's a good thing to do sometimes. But its simplest explanation, let's suppose my image is 1,000 by 1,000. And pose my derivative, my, my smoother is 31 by 31. I could take the derivative of a 31 by 31, which is small, and then apply that to my 1,000 by 1,000. Or, I could apply my 31 by 31 to my 1,000 by 1,000 and then take a derivative of the whole 1,000 by 1,000 which would be a lot bigger operation than taking the derivative of a smaller filter to begin with. That's one of the reasons why, we do that.

23 - End

So that ends the lesson on gradients in 1D, and now what we'll have to do is do it in two dimensions.

2A-L6 Edge detection: 2D operators

1 - Intro

All right, welcome back to Computer Vision. Today, we're going to sort of finish up our edge detection unit. Which will then be used for things that you're going to be doing later. Our last lesson we talked about the notion of edges and how they related to magnitudes of gradients and derivatives of functions. And we remembered what a gradient was in case that part of your brain had fallen off. And we talked about developing some operators that you could apply to the image that would be computing these gradients. We showed how they were sensitive to noise, and we had to do worry about filtering in order to smooth these things out. And we talked, we could sort of first filter the image. And then apply the operator. Or we could filter the operate. We could, sorry, smooth the operator and apply that to the image. And here is a, a picture just reminding that by doing this, saved us that one operation. So here is our function, f. Here is the smoothing filter. We take the derivative of that. And from then on in, we could just use that operator to find these peaks and edges.

2 - Derivative of Gaussian Filter 2D

In 2D of course, it's not just a derivative but we have to talk about the direction of the derivative. In gradients, we have derivatives in the x direction and derivatives in the y direction. So when you do this same trick of taking the derivative of our filter, we have to say, in what direction are we taking that derivative? So let's write this like this. So what we want to do is, let's suppose we want to take the derivative in the x direction. And what this little h of x is meant to imply is that that's a little filter that just takes our derivative in the x direction. Maybe it's a Sobel operator, maybe it's one of the others. But it's a small filter taking a derivative. And g, here, is going to be our Gaussian. And you know, as we said before, because of the associative property, besides smoothing and then taking the derivative, we can apply an operator that is the Gaussian with its derivative. So that's what's shown here. Okay? And the minus 1 1, that's a trivial one, it could be minus 1 0 1, it's my generic derivative operator. And the g, this is my Gaussian, my nice smooth, and I just pulled out a chunk of the matrix that's, that's kind of like that. And what does that give us? Well that gives us this function like that, okay? And so that is, it's first derivative of the Gaussian, which when I apply to an image gives me the derivative of the Gaussian smooth image for that associative property from before. So the question is, is it preferable to do this, right, to apply this. And we should think about this as a quiz, all right. Because it is preferable.

3 - Smoothing Quiz

So here is the question. Why is it preferable to apply h to the smoothing function, h here is our derivative, and apply that result? Well, it's not. They're mathematically equivalent. B, since h is typically smaller, we take a fewer derivatives so it's faster. C, the smoothed derivative operator is computed once and you have it to use repeatedly, or B and C.

4 - Smoothing Solution

A is true, but not relevant here. Yes, they're mathematically equivalent, but I just say, why would you prefer one or the other? Well, both B and C are reasons why you might, right? They're just basically filters you have around. I want the gradient and the x direction. Here's my filter, I'll just apply it. And it, it just makes it a easier way, to do that computation. You know, going forward, you may decide that you've got different smooth images already, and you're just going to take the gradient, so you just do that directly. Fine, you know, be contrary, that's okay, because A is true, they are mathematically equivalent.

5 - Effect of Sigma on Derivatives

So like I said, when you do this, you end up with this gradient function. And of course, we do this both in the x direction and the y direction. And, you know, we now have this problem of correlation versus convolution. Well, if this is the x direction, if x goes positive this way, this is a correlation filter. By the way, this picture is what this looks like as an image. Okay? So these zero values on the outside, that's these gray, and where it goes negative is negative and where it goes positive is positive. In the y direction, which way is positive is always a problem because, like we said, y can go up or y can go down. You know, the main issue of course, is that it, it, it goes vertically. So those are our two operators. So that's how you would use a Gaussian in order to get some smooth derivatives. But there is this question of how big a Gaussian should we use? You might remember from filtering that we could pick different size Gaussians, and here again is the fspecial being used, to get different sigmas. And what happens is, you get more or less smooth versions of the image. The same is true when we compute derivatives, right? We can change the sigma, the size of the Gaussian. And when you do that, you'll get sort of enhancing the, the magnitudes of these derivatives as a function of sort of how quickly the image varies over space. All right? So with a small sigma, all of this fine texture in here shows up everywhere in here. In

fact, even this little texture in here shows up in there. But if I use a bigger sigma, okay? So this area here you see there's a whole lot less going on there, because we smoothed away most of that small variation. So one way of thinking about this, smaller values, finer features are going to be detected and larger values, only the larger scale edges will be detected.

6 - Canny Edge Operator

Now that we know how to compute smooth derivatives and gradients, we can return to the question of how do we actually find the edges. And fundamentally it's a multi-step process, right? You're going to create smooth derivatives to suppress sort of little bit noise and we're going to compute sort of the main gradients, then we're going to threshold that gradient somehow to find sort of significant areas of change, and then we're going to take those areas and then we're going to take those areas and we're going to do what's called, thinning and there's an example of that shown down here, we'll talk more about that, so that a fat edge becomes just a single contour, and say that's the edge, and then finally, we're going to have to somehow connect or link those pixels if you actually want to have a connected contour. So the edge operator I want to talk about is done by John Canny, actually as his masters, thesis, when I was, still at school there as well, went on to do some really cool stuff in his PhD, but everybody knows it because it was called the Canny, edge operator. And the Canny operator works the following way, you first filter the image with derivative of a Gaussian, you find the magnitude and orientation, then you do what's called non-maximum suppression, which is this thinning, and we'll talk about that in a minute. And then, there's a linking operation, we're going to talk about that in detail, that was really the cool insight, where you're going to define not just one threshold, but two, and you're going to use the, the high threshold to start to edges, but you'll use the lower threshold to continue. By the way, MATLAB does Canny edges, so you can just take the Canny edges by calling the edge function, and I encourage you to do doc edge or, or help if, if your doc thing's not set up, to learn all about the edge detection in MATLAB, it's actually a great source of image processing information, the documentation of the image processing tool box in MATLAB.

7 - Canny Edge Detector

So, to illustrate the Canny edge detector, I'm going to use this picture of Lena. Now, this was a picture that was used for image processing a lot. Some guy, he cut out just the top part of a picture of 1972, that, of Lena Soderberg, I think was her name. Because it appeared in a a men's magazine. So he used just the picture. And by the way, this is Lena recently. So we all change. But here is our Lena image. And if we were to take just the magnitude of the gradient, we get a value that looks like this. You see it's close to edges, but actually, you and I see edges, right? It's just a gradient image, all right? So then we can threshold that gradient. So now you can see a bunch of that stuff has gone away. So anything with a gradient that's not high enough has been removed. And then we do an operation called thinning and we'll talk more about this in a minute. The fancy name is non-maximal suppression. Basically saying if I've got a bunch of points that exceed a threshold locally, let me only pull out the point that exceeds it the most. We'll talk about that in a second. When you do that, you get something that looks like that, all right? So that you go back and you can see there's the thinned version of that. Just a quick note about why you have to do the thinning, all right? So let's take a look at this little area here. You can see that there is this kind of thick part that exceeds the threshold. And what we would like to say is that there's an edge running through the middle. So you think about it, if you're coming across here, you might see a profile that starts low and then goes up high, all right? And if you take the derivative of that, you'll see that the derivative over this area is above this threshold. So that gives you this thick edge. And we want to turn it into a thin edge. So the way the non-maximal suppression, the thinning is done in a canny operator is as follows. And you don't act, you're not going to have to implement this, I just want you to be aware of what's going on. Basically it finds areas of high gradient, and it looks across in the direction of the gradient, and it finds just the peak there. And then over here same thing, you would find the peak here. Here the gradient is in that direction, so would find the peak there, right? Sometimes,

and that's what this picture's showing here, sometimes you have to interpolate, right? You find the gradient, you say okay, I think it's somewhere in between two pixels, so you can actually get sub-pixel accuracy. But the point is, that it basically looks perpendicular to the gradient, in order to find, the maximum. So that's how you do the, the thinning part, and now there's one very clever detail to look at, all right? If you take a look at this point sort of under the chin here, you'll see that some of the pixels did not survive the thresholding, okay? And it's a problem because we, maybe we, you can say we had too high a threshold. But the problem is if we make the threshold too low. Then a whole bunch of stuff is going to show up that we don't actually care about. So the question is how to deal with that. So this is what John did in the, the hysteresis. The first thing we do is we apply a high threshold to detect edge, strong edge pixels. So the threshold pulls out a bunch of pixels. Then what we do is we can link those strong edge pixels to form strong edges. Okay. So far not so clever. Here's where the clever part becomes. We now apply a low threshold to find weak, but possible edge pixels. And then, we extend the strong edges following the weak pixels. So what that means is that if an edge only has weak pixels on it, it doesn't get found to be an edge. An edge can only be found if some of the pixels are strong-edge pixels. The assumption here is that all edges that I care about will have some strong pixels in them. And then, I might have to continue through a weak area, but the edge got initiated by the strong detections.

8 - For Your Eyes Only Demo

This one's for your eyes only. Don't tell anyone you found these in a graduate computer vision course. All right, I have two images for you, frizzy and froomer. I want you to find edges in these images, and then display the edge pixels that are common across both. Go ahead and do that, note what you see, and move on to the following quiz.

9 - For Your Eyes Only Quiz

So what's the secret code?

10 - For Your Eyes Only Solution

Hope this wasn't too difficult. Yes, the code is 007. Now let's see how we got that. First, convert the images from color to grayscale. And then use the edge function to compute the edges individually for each image. Note that I've explicitly passed in canny. The default is I think sobel, which doesn't work as well. Canny accepts additional arguments, the hysteresis values and smoothing sigma. Feel free to play with them. Here are frizzy's edges. This is almost perfect. Note that around the mouth and nose, where we had thick outlines, we have double edges. This is expected. And here's froomer's edges, also pretty good. Okay. Now what? Note that edge images are binary images. Which means you can and them together to find common edge pixels. And there you go. You never know what's hiding in an image or two.

11 - Canny Results

So here's a result. And, you know, one of the interesting things is, I mean, it looks pretty good, it's a good edge image. But my very first computer vision courses from Berthold Horn, who was one of the creators of computer vision. And, you know, he expressed the concern that it's really hard to know when and edge image is good. Right? Because the real question is what are you going to use those edges for? So I'll just say that the canny edge operator gives us better edges than other operators. Meaning that they tend to pull out the edges that you would want for future processing. Now one question, of course, is since we're doing these smooth derivatives is what size Gaussian kernel do we use in order to compute our edges? And as we said before, the different sigmas, so here we have sigma one, sigma two applied to this clown image, you can see what it does to the

edge image. Large sigmas detect large scale images, small ones detect small. And the choice of sigma really depends upon the behavior that you want.

12 - Canny Edge Quiz

So a quiz. The Canny edge operator is probably quite sensitive to noise. A, true, derivatives accentuate noise. B, false, the gradient is computed using a derivative of Gaussian operator which removes noise. Or C, mostly false, it depends upon the sigma you chose.

13 - Canny Edge Solution

Okay, well it's c, it's mostly false. It's pretty good with spec to noise, but you do have to worry about the sigma that you've picked.

14 - Single 2D Edge Detection Filter

Finally, one last thing to show you just sort of in completeness. So, you remember one dimensional case of the second derivative of the Gaussian. So, what we have here is, this was our original Gaussian. This second derivative was this inverted Mexican hat operator, and when we applied that to f. It's the same thing as taking the second derivative of the smoothed version. We were looking for that spot there and those what are called zero-crossings that, on that bottom graph, that corresponded to where the edges were, okay? But to do this in two, in two dimension is a little bit harder, and the reason is. There's more than one direction to take our derivative. So here we have our Gaussian, the formula there, and it, you know, is this nice mountain in the middle. But we have to take a derivative in one direction, and so, you know, that's the one that's saying x, and there'd by one and Y. And then on our second derivatives there'd be three choices. I could take the partial of x again, so that's a partial of f squared is partial of x twice. I can do the partial of y, twice. I can also do the partial of f with respect to y. Which one am I supposed to use? Well, the correct answer is, you use what's referred to as a Laplacian of Gaussian. The Laplacian operator, shown here, is this second derivative of x of f with respect to x squared plus the secondary of f with respect to y squared. And that's what actually gives you this Mexican hat operator symmetrically. And if you apply that to the image and you take the 0 crossings, you will get your edges. And in fact, if you run the some demonstration code in matlab for edges. You can take Canny edges, or you can take difference of Gaussians or Laplacian of Gaussians, they're, they're almost identical. And you can see, what that does is it's looking for the 0 crossings and it's another way of getting the edges. One of the challenges there is those tend to be closed counters all the time whereas the Canny will only find you the, the contours that have some magnitude support. I would tell you most people, more people probably use canny for just doing regular edge detection.

15 - Edge Demo

Here is a quick demo in MATLAB showing you a few different ways of computing edges. Note that you can do the same in Octave. Just remember to load the image package. All right, let's read an image and display it. Here we use figure to open a new window, imshow to show the image in the window, and title to set a title, all in one line. We will use this idiom frequently. Let's now convert the image to monochrome or grayscale. We'll use the rgb2gray function for this purpose. And here's what it looks like. Now let's create a smoothed version of the image. First, create a Gaussian filter using the fspecial function. Here's what the filter looks like. Whoa, plotting it as a surface might help. Okay. Now apply this filter to the image. Compare this with the original. All right. For the first method, we will shift the image left by one pixel, right by one pixel, and compute their difference. Let's make a copy of the smooth version to create the left image. To shift the image to the left, we copy all the pixels from the second column onwards til the end to positions first column to n minus 1. Note that the last and second last columns here would be identical. Similarly

for right. Now we compute the difference, remembering to convert to double type. Note that the difference may contain negative numbers. So to display it correctly, we pass in an empty vector as the second argument to imshow. Notice how object boundaries are highlighted as brighter or darker areas, indicating greater positive or negative differences. Other areas are almost gray, indicating close to zero difference. Our second method is to use the canny edge detector. We use the edge function, passing in canny as the method argument. Remember that the Canny algorithm performs non-maximal suppression? That, along with some other tricks, result in a much more meaningful edge image. Let's also run the Canny edge detector on the smooth version of the image. Notice how a lot of the detail features are now gone. Here is the original set of edges for comparison. Our last method uses Laplacian of Gaussian. You simply need to pass in log as the method argument. Again, the edges found by Canny are shown for comparison. You can use doc edge to find out more options that might be available. As you can see, there are other edge operators that you can apply, as well as pass in custom parameters. By this time, you should have MATLAB or Octave installed on your local machine. You will need it to solve the problem sets. Please refer back to the introduction lesson for instructions.

16 - End

So, that ends the lesson on edge detection and in fact, the first subunit all together on image processing. Hopefully you've learned a little bit about filtering with convolution and correlation, and the idea about taking these derivatives. We've also talked about filters being used as templates. next, what we're going to do is we're going to take a little detour, and do some real computer vision. That is, computing some data structure that tells us what's in the picture from our images, and will actually use these edges and then we'll come back to doing more about image processing. But first, you'll get to go find some lines, circles, coins, and cool stuff in pictures.

2B-L1 Hough transform: Lines

1 - Intro

Up until now, we've really been doing really mostly image processing where you take an image in, sum of I of x,y and you apply some function to it and you get out a new image what I'm labelling is I prime of x,y. And that's great and there's whole courses, in fact, entire careers. Hundreds of thousands of PhDs written on image processing. But that's not why we're here. We're here to talk about real computer vision. In real vision you still take an image in, but what you get out is good stuff. All right, the whole idea is that we're putting images in and we're getting stuff out. So what are some examples of stuff? Well, maybe just a line. So here we have an image on the left and some edge description of it in the middle, and you can see a whole bunch of lines that have been found within this image. Or maybe you want to find circles. Actually it's a little hard to see, but there's a bowling ball there. And the way the bowling ball was found was find all the circles in the image. Well guess what? There's only one circle and that's the bowling ball. Or, here's some more circles. Just find the coins. And in fact I'll tell you that this particular picture was put together by a TI TA of mine a huge number of years ago. He's now a pretty famous researcher doing stuff on at I can't say where. But this was in order to show how to do a Hough transform on circles. And you might even find a good stuff as like. Find me this car. So you see on the, on the left here just some examples of where we might locate the car. But actually, the car is found right in the middle. Actually, this is a notion of an arbitrary shape. And in fact, we're going to get to how to do finding an arbitrary shape. Actually not this lecture, not this lesson, not the one after it. But the one after that, where we talk about a generalized Hough transform.

2 - Parametric Model

Today we're going to focus on finding parametric models. And what we mean by a parametric model is that it's a, it's a class. It's a class, it's a model that represents a set of instances where each instance can be represented by a particular setting of the parameter or the parameters. So for example, a line is a parametric model. A circle is a parametric model. And, like I said, we'll even be able to talk about a parameterized template, where you have a template but you change it's shape according to the value of some parameters. So, when you're trying to fit a parametric model, or another way of saying is you're trying to find a parametric model inside your image. They're essentially a cup, there are a couple of things you have to keep in mind. The first is, what parametric model are you going to use? That is, you know, how are you going to represent, how do you go from the parameters to the thing you're representing. And in fact, today, we're going to show you that even for something as simple as a line, how do your parameterization matters. The other thing about this model, and this actually works to our advantage, is that the membership, or the notion that a little point in the image or an edge or a template belongs to a model, is not determined by just looking at that edge. You actually have to look at many different edges or look at the whole model. And that notion of extended support is something we're going to take advantage of. And then finally whenever you go looking for parametric model. You could just say well, if computers were infinitely fast, and every time we use one they feel like they're becoming infinitely fast. Well I would just check for every possible line in all possible places. Well, the reality is that even with really, really, really, fast computers. The number of possible models is really, really, really big. So it turns out you're really, really, really, I'm losing track here. Turns out you're a really fast computer, still, you have to worry about the combinatorics of your computation

3 - Line Fitting

Here's a simple example, okay? Line fitting. And you might wonder, you know, what's, why might I want to find lines? Well, you know, as the example shows here on the left, you might want to know that your chips are seated correctly, so you see the lines on the chips. Or this is the tower again for the University of Texas, and you find the edges, and you might just want to see whether or not you might be in the State Legislature, and you want to make sure the tower at the University of Texas is not leaning to the left. That's a joke for some of you. But we're going to assume, for now, that finding lines is something we want to do. So you might be wondering, well, wait a minute, we just spent a long time learning about how to find edges. Aren't we done? Can't we just find the edges? Well, if that were true, I wouldn't be standing here. So let's talk about the difficulty of line finding. So here's that same University of Texas tower, again, courtesy of Christian Gromen, and we've run an edge operator. And what I've done here is, I've pulled up a part of it just so we can talk about some of the phenomenon. So problem number one is that there's lots of points here that have nothing to do with the lines. So all these little edge points down here, these have nothing to do with the lines, so we have to be able to dealt with them efficiently and rapidly. Second, only parts of the lines are actually detected. So for example, here you can see that there's this big gap in the line. Even though what we'd really like to do is say that we found the entire line. And finally, there's noise and corruption actually where we do find the edge. So over here, you'll notice that in these lines across here, there's all this stuff that doesn't exactly lie on the line. You can see that pixels are jumping around here, as well. So we have to be able to go from these noisy measurements to where the edges are in order to find the actual lines.

4 - Edge to Lines Quiz

All right, let's try to build some intuition about how to go from edges to lines. Remember that edge images are just collections of pixels, pixels like these, where possible lines could be found. How many lines can you identify in this image?

5 - Edge to Lines Solution

So you could identify lines in different ways. For instance, this could be a short segment, a longer segment to the right-hand side and a diagonal cutting across. Now this is a consistent set of line segments, which almost accounts for all of the edge points. But there are other possibilities. For instance, this long segment going right through and two shorter segments on either side. There's no single correct answer to this question. If you entered 2, 3, or 4, we would accept it as correct. Hopefully, this will give you an idea how finding line segments in an image is not a trivial task. This is especially true if you don't have any additional context and are looking at edge pixels individually.

6 - Voting

As I said before, it's not feasible to check every possible line, even with a really, really fast computer. What we have to do instead is somehow allow the data to speak to us, allow the data to decide where the lines are. And because this is democracy, what are we going to do? Well, we're going to do something called voting. So, voting is a general technique where we let the features vote for all the models that are compatible with it. And the way it works is pretty straightforward. We cycle through all the features. Now, today for most of the features what I'm going to need is little edge points, all right. And each edge point is going to cast a vote for the model parameters or the different sets of model parameters that it's happy with or consistent with. And when we're all done, we look for the model parameters that receive a lot of votes and those are the parameter models that we are going to instantiate. Why does voting work? Well, voting works for the same reason that Mickey Mouse, Donald Duck, and Lara Croft are never elected governor of California. Angelina Jolie maybe, but not Lara Croft. You see what happens is, in every election there are people, there could be millions of people who, who are unhappy with the main candidates, the real candidates, if you will. And they'll write in some silly name, Mickey Mouse or Lara Croft. But, even if there are millions of silly votes, as long as they don't all put down the same, silly name, then in some sense a real candidate will get elected. And this will work unless, Stephen Colbert, or somebody like that tries to orchestrate an overthrow. But, in general, the idea is that these silly votes are all distributed across, sort of, the not important candidate. So this works the same way for our computer vision. Noise and clutter features will cast votes too, just like the real features. But typically their fo, their votes should be inconsistent with the majority of the good features. Also another nice thing is, even if part of the circle, say, is occluded and there are no features there to vote for it, because the rest of the circle has gotten a lot of votes, we're able to find the circle. So the example we're going to use today to start with is just on fitting lines, all right? And to fit lines, we're going to have to ask a couple of questions. First of all, given points that belong to a line, what is the line? That is, you know, which, which line are they picking. How many lines are there? And which points belong to which lines? Now we're mostly going to focus on the first question for this lecture and in fact most of what we'll do, but extensions to what we're talking about make it easy to do questions two, and three as well. So the, the method that we're going to talk about today is called the Hough transform, and it's a voting technique that can be used to answer all these questions. And the main idea is just that voting principle. Each edge point is going to vote for compatible lines, that is, it's going to vote for any old line that would go through it, and then you're look for the lines that get many votes. So there might be two, three, four lines, and, you can find all of them. And by the way, if you keep track of which points voted for which lines, you're also able to go back and say which points belong to that line.

7 - Hough Space

So I'm going to take you through sort of how this works, and the key to the Hough Transform is Hough space. Here we have a line in image space. And you all remember the first equation, you solve a line somewhere in Mrs. Thompson's or Mrs. McGilacutti's or whoever your missus. Mine was Mr. Zebrow actually. He was also the football coach. I learned algebra from the football coach

in junior high school, which is a very strange thing. Anyway so you remember that the equation line is y equals m 0 x plus b 0. So what we're going to construct is what's called Hough space. And it's really the Hough parameter space. And in lines in this representation we have two parameters m and b. So for this line y equals m0 x plus 0. That's represented at a location in the Hough space at m0 b0. So there it is. There's the point in Hough's space. Now the key idea here is that the line in the image corresponds to a point in Hough space. Okay, you got that? A line in the image corresponds to a point in Hough space. Now we're going to do something a little different. Suppose we only have one point in the image space. And we'll put that point here at x0, y0. Well, what are the equations of the lines that might go through that point? Well, in image space we know that the line that goes through that point is going to satisfy for whatever the m and the b are. It's going to satisfy the equation y0 has to equal m x0 plus b, right? That, in order for it to go through the point x0, y0 it has to have a m and b such that this equation holds. Well, with some very simple algebraic rearrangement, that becomes b is equal to minus x0 m plus y0. That's the equation of a line in m b space. In fact, it's a line with slope negative x0 and intercept y0. Okay, so the idea here is that a point in image space is a line in Hough space. This is the duality. Well, if we have one point, what happens if we have a second point? X1, y1. Well, that's going to be another line. Right? That's going to be a line with b equals minus x1 m plus y1. And so now here's the really cool question. What line would be consistent with both points? Well it has to be the point where these two lines in Hough's space intersect. because that's the m and b that's consistent with being on a line that goes through x0 y0 and on a line that goes through m1 b1. And this ladies and gentlemen, boys and girls and all you interested parties, this is how we're going to find lines from points. So, now we have to reduce this to an algorithm. We'll first do it graphically and then actually show the algorithm. So basically, every point gives me a line in Hough space. So what I do is, I create a grid, here it's of m and b, we're going to change that in a minute, which, which are made up of a set of bins. And every point votes for a line's worth of bins, so it casts a vote in every bin that it goes through. You collect up all the votes. And after every point has voted, and whichever bin has the most votes, that's your line. So this, basically, we're going to be casting votes into bins and then finding the bins that have, sort of a large number of votes. Larger than, sort of the average noisy area. Before we implement this in real code and real math, we're going to have to rethink our representation of lines just a little bit. You might remember, that there were some issues with the m b representation of ver, or lines. In particular for example a vertical line is really painful, right? Because m is equal to infinity and in 7th or 8th grade or 9th grade or 12th grade algebra. The notion of having an infinite slope is kind of very painful. All right, so what we want to do is we want to use a more robust representation of lines. So that we don't have any sort of bad numerical problems. So what we're going to use is a polar representation for lines.

8 - Polar Representation for Lines

So in our polar representation, this purplish line here is going to be defined by two quantities. One of them is just this distance, d. This is the distance of a line to the origin, the perpendicular distance. It's the distant to the closest point on the line to the origin. And the second parameter, so that first parameter is d, and the second parameter's theta. which is just the angle, that this perpendicular makes with the x-axis, or if you want, you can have it be the angle that the line makes with the axis, doesn't matter. You just have to pick an angle. So basically what we have now is a polar representation of an angle. And a distance. In such a representation, by doing just a little bit of math, you can basically see that the dot product of any point on the line xy. Dotted with the cosine theta, sine theta location of this normal is equal to d. There's a lot of ways to do it but basically you can show that if you formulate it this way, that x cosine(theta) + y sine(theta) = d. So it's a lot, not a lot, it's a little bit ugly. Well, it's beautiful if you like trigonometry. It's ugly if you like algebra. It's a little bit more ugly than the y equals mx plus b formulation. But it doesn't have this problem that any of our lines are ill-defined. This is a perfectly fine way of representing any line. You can have any direction you want. Theta can go however it wants to go. And you can be, you can go right through the origin. D can be zero, or it can get as big as you need it to go. One of the interesting things now is if you take a look at this equation. If I know x and y, what I have left in terms of d

and theta. Is a sinusoid, all right? Which is why we say that a point in image space is now a sinusoid in Hough space, and we'll see an example of that in a minute. See, before, we had this beautiful duality of a point in image space was a line in Hough space, and a point in Hough space was a line in image space. Well, because we've introduced this cosines and sines, it's still a duality, but it's no longer simply between points and lines. One other comment about this, there's a redundancy or an ambiguity here. Let me draw it to you this way. Here's d, so if d can only be positive, this line has to be able to spin all the way around. So theta would have to go from zero to two pi, zero to 360 degrees. But If d could be positive or negative. Then theta only has to go from zero to pi. Or zero to minus pi and how you draw that. The idea is you'd only need a 180 degrees worth of coverage. Which way you do it doesn't really matter and our algorithm will do it in a particular way. But there's just this trade-off between if you let d go positive to negative top has it negative then only it has to go zero to pi. In fact, you can restrict things even more if you make this be the origin of your image, the top left-hand corner. If you're going to say that your line has to be within the image. Okay? Then that restricts even more because it has to cut off that quadrant, so theta and d are restricted even more. But these are just choices that you make in terms of how you code up the algorithm.

9 - Basic Hough Transform Algorithm

So speaking of the algorithm, let's take a look at it. So we're going to use the polar parameterization of the line, the polar representation. And something called a Hough Accumulator Array, which is just a fancy word for the thing that's going to collect the votes. You're basically going to have an array, two dimensions in this case, that where the bins represent different values of d and different values of theta. One of the things you'll have to decide, and by the way you'll be implementing this, is how big are the bins? How many of them are there? So if it goes from 0 to let's say 0 to pi, well if you have every one degree then there'd be 180 bins, if you had ever 10 degree there'd only be 18 so you have to decide how big the bins are. So given this equation and the Hough Accumulator Array, here is the algorithm. And I hope you appreciated that I tried to color code this. It looks garishly ugly but it tries to keep everything clean. So, the basic Hough algorithm is this. You initialize your accumulator rate to zero everywhere, then, for every edge point, aha, so we have to have a set of edge points. That is, at every place in x, y, we have to know whether it's an edge point, or we have a list of them. Somehow, we know which points are edges, in fact, how do we know we know that? Because we did that last time, right? So you know how to do this in MATLAB or Octave or your can write it in assembly code if you were really masochistic. So for each edge point x, y, what we do is, so here I use theta from 0 to 180, that's you know, supposed to be 1 degree increments. All right, solve for d. We just use that equation since we have x and y and we have theta we can solve for d. But notice, I'm not doing anything to restrict d being positive or negative here, so it could be positive, could be negative. And I have to then set H of d, theta, I have to increment its vote. So if d is negative, what that means is, what I really mean is, the bin of that d. So maybe my d goes from minus 100 to 100. So I would have 201 bins, if I do them by steps of 1, so if I got a minus 20, I'd have to add 100, that'd be 80. You, you get what I'm saying, right? The, the d value goes into its bin, so the bin of that D value gets incremented by one. After you've done all the voting you find the values of d and theta, where H of d of theta is a maximum. So you want to find these peaks. By the way, MATLAB there's a function called Hough Peaks, which by the way, you're not going to be allowed to use, because you're going to have to write your own peak finder. All right. Well anyway, let's assume you found just one d in theta, well the, the line itself would be d is equal to x cosine theta minus y sine theta. Get it? That's all there is to finding these lines given the edge points.

10 - Complexity of the Hough Transform

So we did an algorithm. Whenever you do an algorithm, you just need to think about things like how well does it work? Well, obviously it work great, because I'm telling you about it. No, but we'll, we'll see that in a minute. More importantly for algorithms we have to talk about things like

complexity. So the first question is, this is the easy one. What's the space complexity? That is how much memory do I have to use? Well, forgetting the image for a moment, the bottom line is I need k to the n bins. All right? So if I have k bins in each dimension, it's k to the n is the number of bins. So we were doing this in two dimensions, so it would be k squared. So, you know, if I have about 100, so it's about 10,000 maybe. That should tell you and this is going to come up in our next lesson, that adding the number of parameters, which increases n in this case can be very expensive in terms of memory. In fact, also in your problem set, you're going to try to do something and you're going to grow up MATLAB and then you're going to figure out how to fix it. What about time complexity in terms of the voting? Well, the nice thing is that the voting is linearly proportional to the number of edge points. You run through your edge points and they each vote. And the voting maybe might take a little bit of time, but it's constant. All right? Compare that, let's say, let's suppose you're trying to fit a circle. Okay? So a circle is defined by three points. So if you have some large number of edge points, the number of triples is that number, let's call it q. Q choose three would be a very big number, right? Whereas if you do something that's proportional to just q, you're in much better shape. So the idea here is that the time complexity is constant in the number of features or edge points that you've detected.

11 - Hough Example

Let's take a look at some examples, some toy examples and then some real examples. Here we have a cartoon example of an image on the left that just has a bunch of dots in it and those dots happen to all lie on a line, which is good. So this is a noise-free Hough example of which there are none in the real universe, but I made, I've got one here. No, I was about to say I made one, no, I stole this one. Others I made, these I stole. So on the left here, in image space, we have a bunch of points that all lie on the line. And they lie on a perfect line, so that's how you know this is a cartoon example, because that never actually happens for real. What we have on the right are the votes. And what you can see here is that each dot is creating a particular trace. And you'll notice that those are parts of sinusoids. Kay, here's another one. And that comes from that equation that I was saying before. And what's most important is all the votes over here line up. And that's how we would know that's where the line is. See? So cool. All right, suppose I showed you the picture of a square. All right? And I ran an edge detector on it. What would you expect to see in Hough space? Well let's see, how many lines do we have in a square? That you learned hopefully before seventh grade. There would be four lines, okay, and you would get a Hough accumulator array, a bonding thing that looks like this. And here you see that there are four peaks. It's a little hard to see sometime in the image, but the idea is that the values there are much higher than the values elsewhere and you can see how the votes overlap. So here you would know that there are four lines and you'd be able to pull them out. Okay? Here's another blocks world scene. You can see that there's a bunch of edges there, and here's the Hough array, and you see there's sort of sinusoids dancing all over the place. Here by the way you see one really big, bright spot. What's that spot going to be? Well, that's going to correspond to this nice, big, long edge. There are other spots. And that makes sense because there are other lines. But this big bright one is going to be the longest one. Because it has the most votes.

12 - Hough Demo Intro

We'll take a minute here to do a little demonstration of using a Hough transform that's built into MATLAB. And I'll repeat, on your problem set, you're going to be doing some Hough code. And you're not to use the Hough implementation that's already there. Nor anybody else's Hough implementation that you have from the, from the web. Because it turns out when you go and write your Hough implementation certain things are going to break. And it's in that experience of it breaking that you're going to learn what the important elements of it are.

13 - Hough Demo

To detect lines in an image, we first need to find edge pixels. So let's load an image, convert it to grayscale, and find edge pixels using the Canny Operator. Let's see what these look like. Here's the original image. As you can see, it has some lines in it. The grayscale version, and edge pixels. Now we'll apply the Hough transform method to find lines. For this we'll use the Hough function in MATLAB. Find out more about this function in the MATLAB documentation. The equivalent function in Octave is houghtf. The first returned value is the accumulator array. The second is a vector of theta values or angles and third is a vector of radius values or rho. Let's see what this looks like. We pass in the theta and rho values to properly label each axis. Rho or distance from origin is along the y axis. And the angle, theta, is on the x axis ranging from minus 90 to plus 90. Okay, so let's find the peaks in this accumulator array. We pass in 100 as the maximum number of peaks we are interested in. Note that a similar function called immaximize needs to be used in octave. Let's plot these peaks on the hough accumulator array. Note that we need to use the theta and rho values to plot the peaks correctly. The peaks are marked by small red boxes. The size of the peaks matrix is 13 by 2. 13 peaks were found. Each row contains the location of a peak. The first column has row values, or y values. And the second one has x values. Using these peaks, we can find line segments using the hough lines function in MATLAB. It looks like 28 line segments were found. Each element in line_segs is a structure where the two end points, the theta and the rho values. Let's plot these line segments. As you can see, most of the longer line segments have been detected, but a lot of spurious line segments also show up. Okay, so how can we get better results? Let's take a look at the edge pixels again. We notice that there are breaks along the longer lines in some areas. There is also this dense grouping of curves that could throw the Hough detector off. To find a set of cleaner, or more meaningful lines, we can do multiple things. For instance, we could increase the threshold parameter for hough peaks. To understand what these parameters mean, let's look at the documentation for this function. So threshold is the minimum value in the accumulator array that is the minimum number of pixels that support a line required for that line to be counted as a valid candidate. Any possible lines with less pixels will not be considered. Here we set it to be 0.6 times the maximum value in the accumulator array, the default being 0.5 times max. Neighborhood size defines the region over which local maxima will be computed. Note that this is not a region in the image. We are computing local maxima in the accumulator array, so the size of the neighborhood is defined in rho and theta dimensions. A neighborhood size of five degrees along the theta dimension means that a strong line will suppress other lines that are similar but slightly off in direction. Recall that we found 13 peaks in our last attempt. This time we have only seven peaks. Let's see where these peaks are. Looks like we might have cleaner results after all. Let's compare this with the previous accumulator peaks. We see that a lot of the previously found peaks in this dense region are now gone. The new peaks are clustered around three major locations. Okay, what more can we do? We could play with the parameters of houghlines. How about we increase the fill gap parameter to 50? This is the maximum number of pixels allowed between two segments for them to be counted as one if they lie along the same line. To focus on longer lines, let's increase the minimum length to be 100 pixels. For a better understanding of these parameters, take a look at the documentation for houghlines. Okay, let's see what the new segments look like. Compared with the previous results, we see that the false positives have been mostly removed. Some of the previously broken segments have also been joined. Obviously you can do a better job by playing with the parameters, especially here. So feel free to play with the huff transform functions.

14 - Hough on a Real Image

Here I'll show you an example of Hough running on a real image to show you that what it does well and what it doesn't do well. This is a picture of an American football field. This is American football, you know, played with a ball that's not round. We run an edge detector of some flavor, and we get out these edges, and those are some interesting looking edges. And then what we do is we run that through a Hough accumulator array. You can see the sinusoids here, sort of spread out, but you're seeing that these, these squares are all of these peaks that were found, and this is using some

code that tries to look for peaks. So you'll notice that a whole bunch of them are very close together, right? So being very close together would mean that they're about the same angle, and that they're approximately in the same location. Same angle, same location, well, yes, I mean, here are lines that are the same angle and about the same location in here, and here, and here. And even here, about the same angle, about the same location. So you would expect to find peaks that are nearby. So I'm showing you this peak image here. If I were to draw the lines associated with each one of these peaks, I would see something that looks like this. So, there's good news and bad news on here. The good news is that it found an awful lot of line segments. Okay, you'll notice it also missed a bunch. Okay, why is that? Well, that has something to do with the nature of the edges that were found in the bins in the voting. These are all details that are going to matter in terms of how well you find the edges. You'll also notice that there is this cyan line, here, and that's the longest line segment found. Now, somebody should be saying, wait a minute, line segments? What do you mean by line segments? Line segments? What do you mean by line segments? Well, in the Hough transform, when you find a theta and d, or an m and b, for that matter, that's an infinite line, right? That, that's a line that goes as far as you want to, certainly goes through the entire image. If you want to find the line segments, of the points that voted for that line, and just connect them or just connect the two that are furthermost away. Or do some other operation of running along that infinite line, seeing if there's an edge point near there or anywhere near there. So you have to do some other operation besides the voting we just showed in order to find the line segments. The good news for you is that in your problem set, I just want you to find the infinite line. You don't have to worry about finding the line segment that's actually supported in the image.

15 - Impact of Noise on Hough

Couple little things to look at remaining. The first is the impact of noise on the Hough transform. So here what you can see is an image on the left where we've taken that same cartoon set of dots and we perturbed them a little bit off the line. So we've added a little bit of noise to their location. On the right, you see the Hough votes. And you'll notice that now, the peak, not so precise. And in fact, if we had very, very fine bins, we might miss that peak altogether. So, in a minute we're going to talk a little bit about changing bin sizes, with respect to noise, as a, a way of making the thing work. But, what happens is that small amounts of noise can bump you off. So by the way, one really kind of cool thing you could do, if you wanted to find sort of the general peak over here, what might you do? You might smooth this image, right? You could actually take a, say a filtering of that as an image, then find the peaks. And now you know you've moved the peaks around a little bit because you blurred it. But now you at least found the peak, and you say, I'm going to do the Hough transform again, but this time I'm going to just focus on that area. Okay, so I'm going to build a new array with much finer bins but only there, and if d or theta are outside there, I'm not even going to count those votes and that would let you go from a noisy image to, to a better one. There's actually another problem. What happens if we have a lot of noise? Like, suppose all we have is noise. Here on the left, what you've got is a bunch of points that are put down, that are just randomly put there. And on the right, everybody gets the vote. Lara Croft or whoever you want. Go ahead and vote. Turns out there were no real candidates. But maybe we don't know that. You can accidentally find peaks. So sometimes you have to worry about, is the peak that I'm finding actually real or not? It's helpful if you already know, let's suppose you know there's six lines in the picture, well then you just find the six highest peaks, but if you don't know how many are there, you have this question of, when is a peak a real peak and when is it just the accidental alignment of votes?

16 - Extensions

To conclude this lesson, I want to talk about a couple of extensions to the Hough transform. And then some of these extensions are going to carry over, both to the next lesson and the one after. By far, sort of the extension that people leverage the most is the one shown here using the gradient. You'll notice that our algorithm is almost exactly the same. We initialize our accumulator array. We iterate over each point, but now instead of looping or iterating over all possible orientations, we

actually take the gradient at that point, and we take theta from that gradient. Now you could take a single theta, or you might even take a range of thetas, where you say, well, I know that it's, you know, approximately 45 degrees plus or minus 10, so I'll vote for minus 35 to 55, but I don't have to worry about voting for all the others. I've written it here as if you've got a single value. Now that you have theta, you can solve your equation directly, just as we did before, and we increment our accumulator array. The nice thing is, is that by using the gradient, you've reduced the voting time hugely, right? You don't have to worry, you just have a single point. Also, later, you can use it to reduce the dimensionality. So this is the extension that people make use of the most. In fact, the whole notion of orientation is something we'll talk about more going forward. A few other extensions. One is well, remember when we were doing edge detection, we said that some edges had stronger magnitude than others. And you have to set a threshold. Well, you might lower that threshold a bit to try to get more edges. But the idea is you might want to count the edges with a higher threshold more. Well, what would it mean to count them more? Well, you could imagine that stronger edges would get more votes. Another extension, and this is similar to what I was talking about before is, is playing with or changing the bin size of theta and d. As we said before, big bins make it easy to, first of all, vote, it's fast. But you sometimes can get similar lines landing in the same bin. To find a bin, and you have this problem that the real line because of noise votes for different bins, so extension is to do this hierarchically. First, do a coarse binning where you, you, have sort of larger bins, bins that capture more values. Once you find peaks there, you then go to finer arrays just within those areas, and you, you improve your recovering of the model. And finally, it's not just an extension, but it's a whole new way of doing things. We did this for lines, but you can do this easily. I shouldn't say easily. You can do this pretty straightforwardly for other parameterized types of shapes like circles, which, in fact, we will be doing in a little bit, oh, and you'll be doing as well. Or actually, any other shape including shapes that are defined by their templates.

17 - End

So that's the end of this lesson. You will undoubtedly go back and look at this pretty carefully since this algorithm and the manipulation of data sets are going to serve as the basis of the first part of the Hough problem set. And when you look up this stuff on the web, which many of you will you'll want to relate that back to what we've seen here. And I'll tell you this, implementing the Hough transform feels relatively straightforward. That's why people, when it was on a course where the problem set was due Sunday night at 11:55 p.m., many of them started at around 6:00 p.m. They later reported that it took them more like ten hours to do this problem set. And that's because the extensions and so, etc. Nothing quite works exactly like I say it does. That by the way, is going to be a lesson of this class. The papers, first, what we say in class never works. because it's a distillation of the simple stuff. What we say in papers hardly ever works, because we have to make it very you know, sound great for publication. In reality, stuff works, you take the theory and then you have to massage both the imagery and the al and the algorithms.

2B-L2 Hough transform: Circles

1 - Intro

All right, welcome back to Computer Vision. So having done lines, and remember lines are sort of the easiest version of a parametric model, now let's move to something just a little bit more complicated, namely circles. So, here's the equation of a circle. You remember this from, this is Mrs. McGillicutty's algebra class. X minus a squared plus y minus b squared equals r squared, where a and b are the center, and r is the radius. And for now we're going to assume the radius is known. How did you know it? You know, Megan sent you a postcard and said, the radius is 11, okay, fine, so you know the radius. Oh, and we're also not, not going to use any gradient information just yet. We're just going to find the location of the points. So here we have a circle, we don't actually know

where the circle is, but we've detected say, three points on that circle, the blue dots here, okay? So what is the Hough space? What is the accumulator erase space for the circle? Well since there are typically three unknowns, a, b, and r, but I told you the radius, Hough space is just a and b. Right, the center locations and the x and y direction. So, now let's consider the first point, say let's say, x0, y0, this point right here, okay. So, that point has to lie on the circle, and we know its radius. So one way of thinking about that, is that the circle is gotta be some radius r around that point, all right? And so what that does is it votes for a set of points, and that's what this green line represents, right? This set of points that are r around this location in the ab space, okay? So for a single point in the image space, we get a circle of radius r in the Hough space, all right? Let's move to the next point. Well, same thing, right? So it's gotta be, again, a radius r around here. And in Hough space, we're going to vote along this circle. And again the next point. And so if each point votes each point in the image votes for a circle, we collect all the votes. And so just as before, we get the majority of our votes here and that corresponds to this middle point, and that's what's drawn there. So it acts just like before where we were voting for lines, except now instead of using the sinusoids, we're voting in a circle in an ab Hough space.

2 - Detecting Circles with Hough

In fact, here's a nice little example of the thing running. And this is taken from a very ancient photo but you can see right in here, there's a bowling ball, and that's where the plus, the crosshairs are drawn. And just as a, sort of to show you some other way of trying to find this ball that's moving, you could image just looking for the stuff that's moving. After all, it hasn't hit the pins yet. But when you'd look for the stuff that's moving, they find this bounding box. How come we could find more than just the ball? Well actually if you take a look really carefully, you'll see that this lane is very shiny. Okay. In fact, in a little while we're going to talk about reflectance functions and specular reflectance functions, and you'll talk about why on this lane you actually see the image of the ball. And because of that, there's motion in here as well, so you get this bounding box. But that's okay, all of we're saying is basically, that it was able to find the, the circle. Here's another example taken. In fact, if you go and you type Hough transform circles or something like that into Google, in Google, and you look for images, you'll actually come up with this image. This image was actually taken by Vivek Kwatra back when he was a wee graduate student, TA in my class, back in 1811 or something like that. And what we did was we took a picture of of these coins on top of a textured background. We, he did it because he had to, because he was my TA. He's now a really famous guy doing cool research stuff, all right. So, basically, you can take this and you can compute the edges, and here's an edge image of that, and then we can look for circles. Now, we're going to use the known radius method. So, let's suppose we start with the radius for a penny. All right. And you can probably see that there's this nice bright spot right here in the middle that corresponds to the penny. Now what you may also notice is that there are these kind of blown ups areas. Not so much here. It's a little harder to see. We'll go back here. And that's because you could fit a penny. Sort of around the edge of the circle, and the centers would be rotating around the middle of the circle, and that's what this little circle of centers is. But at the real penny, they all align up, and so you get even a brighter spot. So how would we find the quarters? Well, we just use a bigger radius, the radius of a quarter, and we vote again, and again you'll see these spots here, here, and here. And now the penny edges, well they, they again vote for this bit of a circle, but not nearly as strong as the quarters do. So this was the original is, image, and these are the combined detections, and so that's pretty cool, you know, you can just find the circles.

3 - Hough Transform for Circles

But suppose you don't know the radius. Okay? Megan sent the postcard to the wrong place. You moved, because you wanted to upscale your apartment and now you know you shouldn't have, because you're left without a radius. So what do we do? Well, let's think about it this way. All right? Now our Hough space has three dimensions, a, b and r, because we don't know what the radius. All right? So now if I have single point, what happens? Well, now if we knew the radius, let's say,

the radius was, I don't know, seven over here, then it would be some sort of a circle centered about this point. Right? Just like that we did before. And if the radius was let's say, three, it would be another circle centered about the same point, but smaller. So I hope you can start to see that what we're getting in here is actually a cone, okay? So with unknown radius, each point votes for an entire cone in this 3D space. So the surface of the cone, right? Not a filled in cone, but the cone that's a surface. And the next point, what does it do? It votes for another cone and you can sum these all up. Now I will tell you this is pretty painful. And in fact, if any of you end up doing a problem set for some random course like this one, we're going to to tell you in terms of looking for circles. If you try to vote in a great big 3D space, it's not going to work so well. And in a whole bunch of lectures later when we teach you about ransack, it'll sort of overcome this dimensionality. But for now, just know that we have this little problem of growing the size of the voting space. So it can be done, it's, it's just a little bit painful. But remember before, one of the ways we got rid of the number of votes you had to do was by using the gradient direction? Okay? So because if like, I had a point and I knew the gradient, there was only one possible line it could be. Well, we can do the same thing with circles. All right? So now we have an unknown radius, but we have a gradient. All right? So again, our Hough space is a, b and r. But this time, our one point here, it has this gradient. Okay? Well, now if we knew the radius, we would just have one possible circle it could be. Right? So if you tell me that this is a point and that this is the gradient that I know the radius, that's the only place the center can be. But if the radius was half that, then the center would be there. Or the center, I guess it also could be on the other side. So it's just this single line of voting that can happen when you have the gradient and that's drawn like that. So in the Hough space, even though it's a three dimensional Hough space, you would only vote along one line. Okay? And so that's a little bit better, at least it makes the voting easier. You still have this problem when you have this three dimensional Hough space.

4 - Algorithm for Circles

So here's the algorithm sort of simply described for circles. Just like before, for every edge pixel, for each possible radius, right, so we don't know the radius, and then, either for every possible edge gradient, if we vote for the cone, or if we use the estimated gradient, then we compute a and b and we increment for a, b, and r, all right, and then that would just vote you in your Hough space. In general, even though 3D is only 50% more D than 2D, you'll find that voting in 3D is way more than 50% harder than 2D. And that's because, just one way of thinking about this is, let's suppose you had 100 buckets in each dimension. Okay. So, how many would that be if you had one dimension, Megan? 100? And if you had two? 10,000. And what if you had three? All right, that's a million, right, ten to the sixth. The number of cells that you have to consider is exponential in the number of dimensions, and that's a challenge with using the Hough transform.

5 - Voting Practical Tips

So a couple of practical tips with voting for the Hough transform. One thing you want to do is try to not do too many irrelevant votes. So try to minimize, sort of, edge elements that are not robust or are highly unlikely to be useful. Because remember, we don't have to find all the points along the line, just enough of the points. You want to choose a good grid discretization for your Hough accumulator array, right? How many bins you make. And there's no, sort of, magic answer to that. It's a little bit of an art. The problem is if things are too large, the bins are too large, then too many sort of false points vote for the same bin, and if the thing is too fine, then a little bit of noise will cause you to vote for the wrong bin. So it's, this, this trade off it's sort of the how course and fine the data structure is. The other things you can do, you can actually say, well I'm going to take my point, and I'm going to vote for the bin. But maybe I'll also vote for bins nearby, because maybe I'm not exactly where I need to be. It's a little bit like smoothing in the accumulator array. That actually can help. Using the gradient or the edge direction reduces the degrees of freedom of your voting by one, and sometimes that's very useful. And then the last thing is, once you've found, let's say, the circle or the line. You could go and draw it in the image, but if you actually wanted to find the edge

points, well you could do a couple of things. One is you could actually try to look along that circle looking for those edge points. Or you could have been more clever to begin with. You could have, when you voted in those buckets, in those bins, you could have kept track of every point that voted for that bucket. And then once you have a winning bucket, you go and you look at, in that bucket and there's the list of points that voted for it. And that way you could find the points in the edge image that actually selected that object.

6 - Pros and Cons

So some pros and cons of the Hough transform. All the points are processed independently, so it's okay if, you know, part of my stuff is occluded because each point just gets to vote. There's a modest amount of robustness with respect to noise so, because the noise points, as we said before, there are, they're unlikely to sort of collude to vote for the wrong thing. And you can use it to find multiple instances of the object within the single image. There are some downsides. The biggest challenge is that the complexity goes up exponentially with the number of model parameters. So you're never going to use this if you've got, like, seven model parameters, or maybe even four, that you're, that you're trying to explore. You'll have to use other methods. And the other thing is if you have sort of non-target shapes, suppose instead of circles, you actually had slightly squashed ellipses, then the voting can break down, and you have to be careful in terms of how you handle that. And as we said before, the quantization is hard to pick of good grid size and does require experimentation.

7 - End

But that ends the lesson on using Hough transforms to find circles. That, along with the lines, is sort of the, sort of very old approach to parameterized method methods of finding parameterized objects. Even though it's old, it's, it's a good way of learning about extracting structure from an image. Remember, we go from pixels to things. And that's actually why it's the first sort of serious problem set. For those of you who are taking the online class, or those of you who just want to try it. And by the way, it's also one of the first problem sets I give my on campus class. Because they are not used to this idea of going from, sort of an image to a, a data structure. Next time, we're going to talk about, something more general uses of Hough transforms. Where your parameter is not based upon, analytical model, like lines and circles. But instead is based upon the shape of the objects that somebody gave you. And we'll talk about how to do that and also why that has developed a little bit of a resurgence in recent years.

2B-L3 Generalized Hough transform

1 - Intro

So today what we're going to do is we're going to finish up Hough with two significant changes. The first one is we're going to use non-analytic models. So it's not going to be lines, not going to be circles, where we had, where our parameters were analytic parameters. Instead, the parameters are going to express, say, the orientation or the scale of some fixed but arbitrary template. The second thing, and this we'll only talk about briefly. Is a visual code word based approach where we used what are called visual code words for the features instead of edges, and this is much more modern computer vision. Edges really came from a, an older view of, of how objects would be described. And now the focus tends to be a lot more on individual features. And, later in the course, we'll talk about interest points, and features about those points because I don't want to teach you just old computer vision. So, that was the, so the, the first one was then, and this is the now

2 - Generalized Hough Transform

For a generalized Hough transform, it was easy when we had a specific shape, because based upon, our equations for a single point in an image or a single feature or single edge line, we would know how to vote, we would essentially solve for the equation that would tell us how to vote in the bins. But if you have an arbitrary shape, how do you know how to vote? Well, the way you do this is, you build what's called a Hough table. It was actually called an R-table in the original article but I think of it as a Hough table. The way it works is as follows, for each boundary point, and here we have our first one P1, what we do is, we compute a displacement or an r vector from that point to, here I've got it as the center, but it's some reference point, this is the point that we're going to use to locate the object, all right. And we measure that r, that's a displacement vector. The next thing we do is, we compute the gradient, and edge orientation. And what, and what we do is, we take that r, and we put it into a table that's indexed by theta, all right. So if we had some other point, that had the same theta, it would also add its r to exactly the same table location. So you would have multiple displacements, and I'll show you an example of that in a minute. But the idea is that you create the r based upon the offset, and then you index it by theta. Here's our second point P2, it has a different theta, and there would be a different r entered into that table. So we store the displacements in a table indexed by theta. At recognition, we essentially go backwards. What we do is, at every boundary point, we compute the theta that's there. By the way, for now we're assuming that the orientation of the object doesn't change. In a minute, I'll show you how to solve it when, well more than a minute, I'll show you how to solve this when the orientation does change. So at every boundary point, here's P1, we figure out the orientation, we go into our table, we find all of the displacement vectors that are associated with that orientation, so there's going to be this one, maybe there were some others as well, but we're definitely going to vote with this one for this point. Then later at P2, we find its orientation, we find all the displacement vectors that are associated with that orientation. That one's going to, definitely has this one, so this votes as well. And the whole idea is that the, the correct reference point gathers a whole bunch of votes. So what you're doing is, you're using this table, this R-table or Hough table, in order to tell each, found each point in the new test image how to vote. This was originally done by Dana Ballard, and I'll point out that this is back in 1980, 30 plus some odd years ago. And that's just a way of showing you that what goes around sometimes comes around, even in computer vision.

3 - Generalized Hough Transform Example

So let me take you through an example now and in this example here you've got a picture. I'm going to assume a few things to make it easy. The first thing I'm going to assume is that we've got these little edge elements, these gradients. And I'm going to assume that we know which way is inside to the edge and which way is outside. It just makes it easier for the image for indexing. So, let's start by looking at all of those bottom horizontal edges that are pointing inwards, all right? So they all have the same theta, all right? Now, as we go across all those different edge elements, they each have different displacements to the center. So, all of those red lines, those are all the displacements associated with being a bottom horizontal edge. So if I found a little bottom bottom horizontal edge, I'd have to vote with all of those displacements. So what that would look like at run time is as follows. So here I have a little edge element. And it's voting for all these different displacements, which all would lie along that line. Where did all those different displacements come from? They're all here. These were the displacements for every possible bottom edge line, so everyone of those vectors have to be voted by each bottom element. So that's just one element but, of course, there's the next element. It has to vote the same way, it comes from the same theta. And another one, and another one, and another one, and eventually this big long line of possible offsets are voted for. Now, I didn't indicate here, the one in the middle is actually voted for more often so maybe you can think of the middle as being thicker than the ends. But it is that this line has the votes for, for where the center might be. We don't know where along the center it is yet. Well, now what we can do is we can take a look at the downward pointing diagonals. And you'll notice that for the downward pointing diagonals, it can be anywhere from being straight above it all the way

over to the right of it, okay? So it can be straight above it or all the way over to the right. So what this means is at run time, each little diagonal element has to vote straight above it, all the way to all the way over to the right. And that's what's represented here for the first part. And here's another diagonal element, and another diagonal element, and another diagonal element. Until finally they've all stacked up. And you'll notice, already, I know where the object is. This is how this center was found. So the idea is that we're using this table of those offsets, indexed by the gradient, in order to vote for the center.

4 - Generalized Hough Transform Algorithm

So the Generalized Hough transform or Generalize Hough transform algorithm is very easy. If the orientation is known, you know the orientation of the object, you do exactly what I showed you. For each edge point, you compute the gradient direction, you retrieve the displacement vectors r for that direction to vote for the reference point, and the peak in this space, in this case, we're only voting in X, Y. That's the reference point, that's where your object is. It's the one with the most, sort of, supporting edges. Now, it's sort of unreasonable that you might know the orientation and the scale, so let's talk about orientation. Okay? Supposed the orientation is not known. Well, all we have to do is try the different possible orientations, and that's written here, again for each edge point. But now, we have a, what we'll call a master orientation. This is the actual orientation of the object. For each orientation, possible master orientation, we compute the gradient direction, but we subtract off the master. So the master here is this theta star, the actual gradient is theta. But the, the gradient we're going to use is this theta prime, which is just the gradient with the overall orientation subtracted off of it. And so now, we retrieve the displacement vectors for theta prime. So we're still voting, and peaks in this Hough space are the reference point, but notice now that we're voting in X and Y and also theta star. So you remember that space complexity, which is k to the n? Well, n just went from being two to three, and the bottom line is any decent sized number squared is much smaller than any decent sized number cubed. So you just got a lot bigger by going to this adding this orientation. What's kind of cool by the way and this was in the original arbitrary shapes paper, you could also do this for say scale. Suppose the scale was unknown, our algorithm looks stunningly similar for each edge point. But now instead of having a master theta, right? We have a master scale. So for each possible master scale, we again compute the gradient direction. We again retrieve the displacement vectors, but now we have to vote with the scaled displacement, the vector scaled by S. And again we vote, whichever gets the most votes is the right point. And the peaks in this space and now the space is X, Y and S where S is the scale. So again, it's cubic. Now you can imagine that if you had scale and orientation, that would be quartic, that's raised to the 4th. Modest numbers raised to the fourth power are brutal, so you have to be careful in terms of how you do this.

5 - Application in Recognition

For the last, part of the lesson, I want to talk about a much more modern approach to using these Hough transform, displacement vectors. In fact it was proposed as recently as 2004, and subsequently, as a way of, locating a particular object. What we're going to do now is, instead of using edges or the idea of high, high value gradients as a, as a feature that you can find. What we're going to use are little feature patches, all right? And feature patches are just little chunks of an image that, for some reason, for, for one reason or another, you've decided, these are useful things to try to go find. So, if you have those, this is the way that would work. So we have a training image here, and I'm going to assume for now that our feature patch is tire, and I'll tell you in a minute about how we go about finding those feature patches, all right? And so here I've located the tire in two different places, and so in the same way as with the gradient, so when I first find it here, I say, okay, there's a displacement vector to the right like that. And I'll put that in a table that's indexed by the feature patch. And then I find that, oop, I found it again, it's over here, and it's index, and it, it has a displacement vector to the left, like that. So I put that in a, in the same table, and in fact it's indexed by the same feature type. So, these features are referred to as visual

codewords, which I'll describe in just a minute, and the idea that I would have a table based upon the codewords. And associated with each codeword is the set of displacement vectors that I have to vote with every time I find that feature anywhere in the image.

6 - Training

So, the first step in training, are developing what's called visual code-words. And, basically the way it works is this, you have some sort of an operator, and we'll talk about interest point operators, that generate what are called interest points. That is, these are points in the image where reasonable amounts of interesting stuff is happening. And we'll talk about Harris corners and other ways of finding that. What you do is, you take your interest point operator, pull out all the interesting points on a bunch of training images. You collect the little image patch right around those points, you may get hundreds of them, or thousands of them, and then you cluster them, and you use some algorithm for doing a clustering. And when you're all done with those clusters, the centers of those clusters are referred to, as visual code words. So here, you can see all of these images that were taken from something like tires, then they were all clustered, and so the code word is this little tire piece. Here's a, here is a centered tire, right? This is the piece of a tire, this is a full tire. And there are other kinds of code-words, so these become the little features that we're going to look for in different images. And this, of course, would assume in this particular case, we're looking for cars, we got a bunch of training images on cars. By the way I should say, all of this is done automatically, okay? So, you're going to see some things that look a little strange in terms of code-words, well. It just happened to fall out of the data. This is not a human doing it. This is telling the system to go ahead and do this. The second thing you do is you take these code words, these are our features. Remember like we had the tire, and we found everywhere that the tire landed in the image. So what we have here is all of these marks, or these little interest points. And what we do is, for every interest point, we find the feature that seems to look best at that point. So that becomes the label of that point. All right, so remember just the way we had a label before that said the gradient was horizontal pointing inward, here we have the label is that it's the bottom right-hand corner of a tire, okay? So this is mapping each of the interest points to some particular patch. Finally, what we do is we take each of these little features and we treat them just like we treated those little gradient images, right? We take the patch, we find the displacement vector to the center, and we write down that displacement vector in a table that's indexed by a patch label. So if I find a tire and it's to the left, which means the displacement vector is to the right. I put down that displacement vector. If I find a tire to the right which means its displacement vector is to the left, I add that same displacement vector to the table with the entry of the tire, and that stores all those displacements.

7 - Application in Recognition

So at run time, it works like following. So here I'm going to show you on two cars using just the tire patch as the example, all right? Suppose we have only one feature and it's the tire. And suppose we try looking for it everywhere, okay? So we look for it everywhere and we find these four. Okay? Well, when we find them, what we have to do is we have to take the displacement vectors associated with that codeword and vote with all those displacement vectors. So let's remember for the tire codeword, there are two displacement vectors. One to the right, one to the left. So we vote for those. And then we simply look for spots, points that have more votes than other places. So in the case, where is it, where are those points? Well, just in the center of the cars. As I said, this was originally proposed by some folks at the ECCV, European Conference Creative Vision 2004 and it subsequently been evolved from that. And the, so the idea is that we're now using this notion of detecting features and something about their configuration in order to try to find these objects

8 - End

So that's the end of this lesson and its the end of our entire discussion about Hough transform. It's an interesting example of a general approach in this case it's voting that was first developed many

years ago like I said back in the 80s. Which resurfaces in sort of more sophisticated forms as things come along. In fact, there's something recently called Hough forests and Hough forests combine the Hough offset ideas that features vote for offsets with an ensemble or a collection of classifiers similar to random forests and that's why these things are called Hough forests. Fortunately for you your only going to be using the basic Hough transform for lines and circles for your problem set. It's more work than it looks, so, you know, make sure you leave plenty of time, and I think after you get that working, you'll have an appreciation for what it means to get these more sophisticated methods to work.

2C-L1 Fourier transform

1 - Intro

All right, welcome back to Computer Vision. So let's see, so we did the images as functions for a little bit, and started talking about edges. And then we took a little bit of a detour where we talked about Hough transform in terms of extracting information using those images as evidence. So now it's time to get back just a little bit to image processing. And this lesson, and one more, or maybe two more beyond this. We're going to be talking about what's called frequency analysis. Now in the old days, that is, and that's actually pretty old already. Now we're talking 40 years ago, 30, 40 years ago. All of you who would be working on images, you would have actually been electrical engineers, and you would have spent a lot of your time studying this stuff that we're going to be talking about extensively. You define your filters and other types of image analysis operation in terms of what's called frequency response. That is, how things change when you have high frequency stuff, and then low frequency stuff. But most of you are much more oriented to computer science, where we tend to think of images as data structures and not so much as a signal. But some of this understanding is still critical. This is especially true for a phenomena referred to as aliasing. I know many of you are familiar with aliasing in a sort of image perspective where you see the little jaggies, because you've got a line that doesn't go perfectly straight, but it's actually a frequency phenomenon, and in fact, the goal of this lesson and the next few, is to get to a point where you can talk about aliasing from a frequency perspective. And, it'll teach you why, if, for example, if you have an image and you want to make it half the size, you shouldn't just throw out every row, every other row, and every other column. That's the wrong thing to do. And, by the time we're done with this, you'll, you'll see sort of why that's the case.

2 - Dali

Here's a picture, a very famous picture, and the question is, what do you see? All right, so most people probably first see a, the kind of stylistic drawing of a naked woman, and then maybe if you look you'll see down here there's a picture of Abraham Lincoln. But if you take a look at the title of the picture, this is a painting by Salvador Dali and its Gala Contemplating the Mediterranean Sea which at 30 meters becomes a portrait of Abraham Lincoln, now you don't actually have to step back 30 meters we can just blur the picture and when, when, and when I blur the picture, it looks like that, and uh-huh, there's Abraham Lincoln. And so the question is, what's going on there? All right. And, loosely the, the answer is that at this image there's all this hig frequency information that's showing you all these squares and you see the, the outline of the woman and you can see the, the, the, what she's looking at something in the distance, and you can see that picture of Abraham Lincoln, but at the low frequency, it actually is a reasonable picture of Abraham Lincoln. And the idea is that images can be decom, actually any signal can be decomposed into a set of things that describe it. And, we're going to get to frequency, but really what we're talking about is what's referred to as a basis set.

3 - Basis Sets

So what a basis set? So here we've lifted this from Wikipedia. Edited a little bit. So a basis set, B, well you have to talk about a vector space. So something is a basis set of a vector space, V, is a linearly independent subset of V that spans V. So let's expand those words out just a little bit. So let's suppose that we have B that's just these vectors v1 through vn, and it's just some finite subset over this field. So field comes from mathematics, but let's just talk about the realer complex number. Okay, then a basis set B if it satisfies the following conditions. So linear independence. Okay, basically means that if I have some different constants, a1 through an, so these are constants. If I take a constant times each of the vectors, I've got a different constant, and if the sum of those is 0. Then it has to be the case that all of those constants were 0. This is no different if you think of this in two dimensions. If I have two vectors that are not in the same direction, then if I combine them, right, I get this answer like that. Well the only way that distance can be 0 is if both of the coefficients are 0. So that's linear independence. And then there's what's referred to as the spanning property. Okay? That, remember it spans this vector field. And what that means is that for any vector in that vector space, I can create it by just some linear sum of the basis set. Then again, you know, basically the idea is, so we normally think of a basis set as being perpendicular, but it doesn't have to be. Right? So if I've got this and I've got that as my two basis elements and I want to create a vector that's like that. Then, basically, what I have to do is I have to take some multiple of the horizontal plus some multiple of the vertical we'll get me to that picture. By the way, most of you are used to basis sets being orthogonal. Okay? So you know, x, y. All we said they had to be independent, okay? Now, it's helpful if basis sets are orthogonal, we'll talk more about it in a minute, because what it means by orthogonal is in order for me to figure out how much of the x there is in a vector. I don't have to worry at all about the y because the y amount and the, it doesn't affect the x amount because they are orthogonal. Another way of saying that is if I take a vector and I take a docked product with the x direction I get one value and a docked product in the y direction I get another value. Well, if I take each of those dot products times their basis vector and sum them up I get back the, the new vector. And that's just because x and y are orthogonal, and that's helpful if your, if your basis sets are orthogonal. But technically they don't have to be, they just have to be independent. That'll let you reach any vector.

4 - Fourier

If we had a basis set for images, and we'll talk a minute about what that means, that could be probably useful for analysis, especially for linear systems because we could consider each basis component independently. Why is that true? because you remember, in linear systems things just sum. So if I could say how some operator applies to each element of the basis set, and then I just have to scale it and sum it, then I'd know how this operator applies to the entire image. And that's part of the reason that we're going to go through this whole basis set decomposition, is because most of our operations are going to be, that we'll be considering, are going to be linear operations. And when I was talking about things from a long time ago and filter theory, et cetera, those are all based on linear, types of operations. To consider basis sets of images, what we're going to do is, we're going to think for now, of an image as just being a single point in a very large space. So if I have a space that's, N by N image, right, so let's suppose it's 100 by 100. So 100 by 100, that's 10,000, okay, so you can think of the entire image as just a 10,000 dimensional vector if you wanted. That would be fine. Okay, so what would be a basis set for a 10,000 dimensional vector space? Well, one particular basis set is just our usual one, right? We do zeros except we have a 1 in one of the elements that's like x and y and z and p and q and r and you run out of letters before you get to 10,000. But, but basically you have a 1 in every place. Now are these independent? Yep. Because if I sum up a whole bunch of them, the only way I can get to a zero vector is if I have a zero multiple of each of them, because they're independent, they don't effect one another. But, it's really not a very helpful basis set because each one of these ba, basis vectors, that one, that's just a single pixel. So telling me what, about, sorry, how it behaves in respect to a single pixel, that's not a particularly helpful basis set. Instead, we're going to look at a different kind of basis set. And it's a

basis set that considers this image, or any image in terms of how quickly it varies in different directions. So here you see an image that is just, you know, no variant stalls. You can think of that as just a constant intensity. And as you move to the right, it starts to wiggle faster and faster and faster. As you move down, it wiggles faster and faster and faster, but this time horizontally, and then if you combine them you would get these different orientations. And by the way the two different pictures of the blue and the green are sort of, you know, you can think of these as like, kind of like sine and cosines, kind of off set. Now this particular basis set has a name, okay, and it's referred to as the Fourier basis, or Fourier decomposition. And it's named after this dude, John Baptiste Joseph Fourier, okay. So he had a crazy idea, a crazy idea in 1807, that any periodic function, that's a function that repeats, could be rewritten as a weighted sum. Try to say that quickly, rewritten as a weighted sum of sines and cosines of different frequencies. In other words you could make any repeated pattern, doesn't matter what it looks like, as long as it eventually repeats, using sines and cosines. So don't believe it? Well you wouldn't be alone. Back at the time there were some other folks you may have heard of like Laplace or Poisson and other mathematician types. They didn't believe it either, lot of controversy. This question about whether Fourier's proof was correct it turns out his result was correct and in fact it was so controversial it wasn't even translated into English until 40 some odd years after his death. But like we said it is true, and the series now, now that he's dead and famous, we call this the Fourier Series, all right. We're going to get from Fourier series to Fourier transform to discrete Fourier transform, just so you know where we're headed.

5 - A Sum of Sines

The idea is that if you have a signal, the signal has as its building blocks these sinusoids. And if you add enough of them, you could get whatever component you want. So here we're adding up different, this is, this was our target, these are the elements that we're adding. We're going to show this more in a minute. And this is the function that we're building up. So before we look at that more carefully, how many degrees of freedom are there in this signal, right here, okay? There are three degrees of freedom. What are they? Well a here is just going to be the amplitude, that's going to be a scale, all right? And then there are these two numbers, omega and phi. And omega is what's referred to as the frequency, and phi is the phase. So which of these is most important? Which of these encodes the, the notion of a coarse signal versus a fine signal? Well it's going to be this one, the frequency. All right? So as you change that frequency, as that number gets bigger, essentially the thing wiggles more quickly.

6 - Time and Frequency

To give you a little bit of a notion of sort of frequency and time, et cetera, pretty soon we're going to throw time away, because we're only going to be interested in space. But, since we're talking about how, how quickly something wiggles, and it originally was derived from one dimension, we'll talk about it this way. So, here I have a signal, okay, g of t, okay, that's made up of two parts. Okay, it's got a sinusoid of some frequency f and then another sinusoid of 3f but it's been scaled to make a third. So that's this signal here and you can write that as just the sum of these two sinusoids. After all, that's what this equation says. And because of linearity the stuff just sums, the combination of two on the right gives you on the left. And if I were to draw out sort of the contributions of the sinusoids to the signal, it would look like this, okay? So this is written, in this particular way. I'm only showing you positive frequency, we'll talk about negative in just a minute. But whatever f is, the idea is that the signal here has a frequency spectrum with a certain amount of power at f, and sort of one-third of that at 3f, okay? And this is one form of drawing out spectra. We'll talk more about spectra in just a minute. You'll notice that this picture on the left is sort of kind of approximating getting towards a square wave, and it turns out that, actually, if you really wanted a square wave, what you do is you keep adding these odd frequencies, all right? And I want to show you this because, sometimes, people will ask, well, how can I get sort of a sharp boundary using just smooth sinusoids? So here we have our original picture. And now I take that original one and I add in something that wiggles at 5 times f, and you see that I'm getting closer yet to a square

wave. And then I get to 7 times f, and what comes after the 9, and what you can see is that we're getting closer and closer to having this as a perfect square wave, okay? And in fact, you can show that a square wave can just be written as this infinite sum, okay, of these frequencies, okay? And it, the amount of power you need, goes down as the frequency, increases, all right? So, what we want to do is we want to get to a point where we can look at a signal and say, process it in a certain way, or, or compute it in a certain way. To say, what are the sinusoidal components that go into making up this image? And in fact, you can show that to do a square wave like this, you need a, an infinite sum of decreasing amount of increasing frequency. And that's what's shown here. Now what I haven't done is I haven't talked to you at all about phase. And in fact, I'm not talking at all about, you know, what the phase, right, I could have this offset, or I could have had different elements offset. And in fact, all I'm showing you here is the amount, or you can think of it as the power, we'll talk more about that in a minute, of the content of each sinusoid. And the reason we're not talking about phase is, our goal is not to reconstruct an image. We just want to, typically what we're going to want to know is, how much power there is in each of the different frequencies. In Computer Vision, we're interested in sort of analyzing what's going on. If we're actually going to reconstruct images, then we would probably have to worry more about the phase to, get you, better reconstruction, but we're not going to do that, today.

7 - Fourier Transform

Given this idea that any signal, certainly any periodic signal, can be made up of a sum of sinusoids, we're going to start to move to, from just this notion of a series, to notion of a continuous signal. And we're going to talk about a, something that will allow us to know how much power of any given frequency there is in an image. That is, we want to transform our image from, just something that's or, or our signal that's something a function of time, or just of space, to know what it is in terms of its frequency. And that transform is called what? It's called the Fourier transform. And now we're going to, so we did Fourier series, then we go to Fourier transform, and then we'll go to discrete Fourier transform, which will get us to that image stuff, alright? So we want to understand the frequency, and frequency is usually written as omega, of our signal. So, we want to parametrize, we want to transform our signal, instead of by x, which is in space, by omega. And we do that by what's known as the Fourier Transform. And the Fourier Transform is going to take some spacial signal out and it's going to give us some properties that encode both the phase and the magnitude for each given omega. Okay, so we need the phase and the magnitude for each given omega. Specifically for every omega from zero to infinity, and actually its from minus infinity to infinity. The idea is that f of omega holds the amplitude and the phase alright of the corresponding sinusoid. So this f has to have the amplitude and phase. How can f single thing, hold both amplitude and phase? Notice I didn't say a single number. Cause remember, f of x is just the value of F, right, so it's a number. F of omega is going to be amplitude and phase, how do we that? Well, f is actually, the capital F, is actually a complex number. Now hopefully, you remember something about your complex numbers. If not, I'm going to [LAUGH] give you a really, really brief review, okay? So basically, F of omega is made up of two parts. A real part and then what's referred to as an imaginary part, okay? So remember, A plus BI. Okay well in this case its the real part of omega plus the imaginary part. And you may remember that the magnitude of a complex number is just the square root of the sum of the two elements. Normally the square root of A squared plus B squared so here its the square root of the R squared plus the I squared. And re R is for real and I is for imaginary and the phase the relationship between these is going to be written this way okay. Its the arch tangent of the imaginary plus the real. And the thing that's missing here is the idea that the real part is going to be due to the even function and the odd part is going to be and the, and the imaginary part is due to the odd. And all that means if I have a function here this is my cosign function, alright. And that's even with respect to the origin that is symmetric about respect to the origin. The sinusoid and I'm going to screw this up right because it has to come here. And then be there, and then, so it has to be zero, there we are, that's my sinusoid. because it, it has to come down to zero when that thing is high, all right. It is odd, that is sine of, of negative x is minus sine of x, whereas cosine of negative x is equal to the cosine of x. So the bottom line is, is that this part

is, the real part is going to be the cosine part. And the odd, and the imaginary part will be the sine part.

8 - Computing Fourier Transform

When we do a Fourier transform, all we're doing is computing a basis set. And I'll show you what I mean by that. See this ugly, ugly integral here? Ok, what I've got is, I've got the sine of one function, and the sine of another. Two different frequencies, a and b. Okay and I'm integrating this all the way from minus infinite to plus infinite. And I claim here that it equals zero if a does not equal b. Now why is that true? Any guesses? Well let's just think about this intuitively all right? If I have some sinusoid. And now I've gotta do one of a different frequency. That's easy. [LAUGH] Boy, that's beautiful, huh? Essentially, at some point when this is positive and this is that same positive value, I'm going to try to find, hopefully there's some positive, I have to make one up. Yeah, let's pretend the red thing went down here like that a little bit, right? That when the, the red is same positive. The yellow was the negative of it. So the product of those two numbers and then some together is going to cancel each other out and that sort of a hand wavy way of saying that this entire integral is going to be zero as long as a does not equal b. But you might ask what happens when a does equal b? What happens when a does equal b? Let's start with assuming they're in the same phase, exactly the same phase. Well, then it's just sine squared. Sine squared is positive. Sum that over infinity, what do you get? You get infinity. So, if you have two sinusoids that are the same frequency, you'll get an infinite value. Unless they're perfectly pi over 2 out of phase, in which case it's going to be 0. In other words, sine times cosine keep going. But the bottom line is the way of thinking about this is that the, if I integrate a function, if it's made up of a sinusoid that's different than omega, when I take the sine of omega and I do that full interval I get nothing. But if it's equal to that sine, I would get infinity. And that's going to be written like this. So let's just do a simple example. Suppose I've got a simple function: f of x as cosine of 2 Pi Omegax. All right? Well then, let's pick some frequency and we'll call it u here just keep things simple, right? If I take this integral, okay, that's going to be infinite if u is equal to that same omega, all right, and it's going to be zero otherwise. So that looks like this, okay? So we just have these two member impulses. We just have these two infinite spikes here. And that's referred to as an impulse that corresponds to the cosine. You see that they're positive this way, and that's because it's cosine. If we have a sine, because sine of negative x is negative sine of x, that would look like it would be up here and that would be down there. That's what the sine is, okay? That's, and that's kept us the imaginary part. So we're just computing a basis set of saying how much of this, sinusoid is in there, all right? And we can do that for all frequencies. But then you might ask, well don't we also have to do it for all phases? Don't we also have to do it for all phases? She studied that one. Well, the short answer is no. One of the cool things is if I have cosine of omega x and sine of omega x, right? So one is 90 degrees or power of two phase out of phase. I can make any phase by a linear combination of those. You can show that, it's just true, we can do a little demo. But basically, if I have a sinusoid of any phase, if you just tell me sort of what its integral is with respect to cosine and what its integral is with respect to sine, I can tell you what the phase of that sinusoid is.

9 - Fourier Transform More Formally

Doing this a little bit more formally, so we're going to represent our signal as an infinite weighted sum of an infinite number of sinusoids. So the Fourier Transform, here it is, in it's beautiful glory, in one dimension, okay. F of u is a Fourier transform. We just take the infinite integral of the original function times e to the minus i 2 pi, u x. You might say, what? Where did that e to the minus i, [SOUND] come from? Where did that come from? Oh, see, she didn't fall into that trap. Well again, remember from your probably calculus class, somewhere e to the i k is cosine k plus i sine k. That's why this is the even part and that's the odd part. This is where i is the square root of minus 1. By the way, if there are any electrical engineers out there, they have j as being the square root of minus 1. I don't know why, because mathematicians use i, I guess electrical engineers didn't like the mathematicians. We're going to use i. So what the Fourier transform does, is it converts from the

spatial domain into the frequency domain, and the frequency domain is either written as omega, sometimes as u, as we're doing here, sometimes even s when you're talking about loss transforms and stuff like that. But the idea is that this cap f is the frequency spectrum, or I should say it's, it's frequency of f of omega, but we're going to talk about its magnitude in a minute, which is going to be sort of what we, we refer to as this spectrum. Just for completeness, by the way, so here we had the Fourier transform. This is the inverse Fourier transform. Unsurprisingly, if you tell me what all of the sinusoids are, if I sum them all up, okay, at some location x, and I sum up over all the sinusoids, I can recover the original signal. And that's called the inverse Fourier Transform.

10 - Frequency Spectra

So putting those equations we just saw sort of into some simple pictures. Our frequency in general can be thought of going from minus infinity to infinity. And the real part is the cosign part and its impulses that look like this the imaginary part is the odd part that's the sin looks like that. And as we said before typically we're only going to worry about the magnitude of the power. And so you just take the sum of the squares screw it up and this would be the power. When I combined the real part and the imaginary part to make power, I can take a look at the power spectrum from different kinds of signals. So here's some and I apologize for sort of the terrible picture. So here we have just a sinosoid and our sinosoid just has power there's two peaks right the regardless of the phase the sum of the squares square root of is going to give us those two peaks. And also here we have that square wave and you can see that the power falls off as frequency goes up. And that's what we we showed before and one of the interesting things also is if you take a signal out of some sort of typical natural image, or typical natural signal, you take a whole bunch of them. You also tend to find these spectra falling off, and that's just is a property of what, of how images are formed by looking at things, in terms of, natural images.

11 - Limitations

Let's just talk about a couple limit limitations on this, which are going to get us to the Fourier series in just a minute, okay? For this integral to exist, this Fourier transform, we don't actually want infinites, right? Because that's not so good when, when are, when we, when we try to like, write down infinity, it just goes a really long time, all right. One thing we'd say is that this whole thing is integrable, okay, if the original function itself is integrable. That is if I take the, the function, I actually have to take its absolute value. If I sum it up from minus infinity to infinity, it, it can't explode. It has to be limited to a value. So you might ask, but wait a minute, I thought we had infinite repeating series. I thought we had infinite repeating series. We did when we were talking about Fourier series. We're now talking about the Fourier transform, where we're trying to keep things bounded. So the idea is that these integrals have to exist. It's base, it's based on the same principle, but it's, it's this idea. So in Fourier series, we talked about repeating functions. In Fourier transform we're talking about integral, integrable functions, all right? In a minute when we have a fixed length function, it'll be clear. In fact, let's get to that right now. You see it says, so obviously if there was a bound of width T. Right, so if, if all my entire function fell within T, then obviously I could just integrate from sort of, T, negative T over 2, to plus T over 2, right? That would be the entire integral. And that's going to lead us, at least in my twisted mind, to this notion of the discrete Fourier transform.

12 - Fourier Transform to Fourier Series

The discrete Fourier transform is what we have to do when we want to start doing things in computers because after all we don't have continuous signals, we only have discrete values. So the discrete Fourier transform is written as this. Now we say F of k, where k in this case would be the discrete frequency, all right? And, what we're going to do is we're going to sum over all the pixels, so we're having x going from, or signal N-1d in this case, from 0 through N minus 1 if I have a total of N of them. So there's this 1 over N out here. And now we have this e to the minus i. And

you see how it's k over N? Because basically, instead of now thinking of frequency as how quickly it wiggles in terms of time, it's sort of, how quickly you wiggle over the entire length of the image. So, k is written as, it's the number of cycles per period of the signal, or per period of the image, all right? Cycles per image. This only makes sense essentially from 0, up to N over 2. Why only N over 2? Well, think about how quickly a sinusoid could possibly wiggle in a image that has N pixels, right? Well, it would go white, all the way to black, all the way to white, all the way to black, all the way. So it, that would be a period of ha, of, of 2. So the frequency, the number of cycles, would be N over 2. So basically, k can only go from minus N over 2 to plus N over 2, because that's the highest frequency you can have in a discrete image.

13 - 2D

I've been showing you this in one dimension. It extends pretty simply in two dimensions and here are the two dimensional forms, okay? So this f of u, this is just when I've got a, a continuous two dimensional function. Here we have f of x, y. Well, when we do it in discreet land, f of x, y is now going to be at discreet points and we have f of kx, ky, those are the two different frequencies. The fr, the discrete frequency in x, the discrete frequency in y. And by the way, this works middle if you put the origin of k in the middle, you want to put the zero in the middle. So for minus k k n over 2 to plus n over 2. Let's take a look at applying this two dimensional discrete Fourier transform to images. So remember, we showed you, we talked about that we are going to have these bases that have horizontal vertical components that can be combined. What does that look like when we apply these things to images? So let's take a look at some very simple examples. All right. So here I have a sinusoid and it's at some particular frequency and it's only made up of vertical stripes. What is the spectrum, the power spectrum of its Fourier transform look like? Well, it looks just like this and its a little hard for you to see, but there are two bright dots there. The origin is here in the middle and there is a dot to the left and a dot to the right. Okay. Those are my spikes for the, the frequency of this particular sinusoid. So this picture, you see has only horizontal stripes and it varies more quickly. So these dots are now spaced vertically for sinusoid that goes in this direction and they're further out, because the frequency component is higher. If I have a sinusoid in an oriented direction, same deal again. It's a little hard to see. We've got these dots that are spread in that direction. Let's go back to our linearity. So let's suppose I take the picture on the left and the picture in the middle and I sum them together. And we're actually going to do a sum, so it's linear. Right? So this picture's going to look like that, so, you know, it's some combination of sinusoids. But remember, a Fourier transfer, Fourier series is just made up of sums and multiplies, it's a linear operation. Okay? So, the Fourier transform of the sum is just the sum of the Fourier transforms. Okay? So that's why this picture is just these combined.

14 - Examples

So let's take a look at some spectra of some real images and some examples, okay? So here's one. Now, if you remember, this is Lena from way back in 1972, and this is no relative of Lena whatsoever, okay? You'll notice that, just looking at the image, you know, there's a lot of power in the middle and then stuff falls out towards the edges. And that's to like I'd said, it's often the case, that natural images have similar power spectra. And what really matters for reconstructing the image would be the phase. We're not doing reconstruction here, we're only looking at the magnitude. So suppose I wanted to get rid of the high-frequency content. Well here's what you could do, and we'll talk about using the inverse Fourier transform next time. But I could take Lena, I could take her power spectrum. I could remove all the stuff out here, and then I could reconstruct it. You can see that there's all these ugly lines in here, okay? That's what's referred to as ringing. This ringing happens because this actually doesn't have any high frequency at all, but you actually needed the high frequency to smooth out those little edges. Okay, and so when I remove them, I end up with having just these little ripples, and that's referred to as ringing. And here's an example of doing the other thing, we take Lena, okay? We remove the stuff out of the middle so you can see this thing zeros there, and you'll notice that what's left here is essentially an edge image. And if you

think about that, the high frequency components are giving you where the edges are, 'kay? And we'll, we'll also look a little bit more about that when we talk about aliasing. Couple more things real quick here. Remember our sharpening filter? So our sharpening filter sharpened the image by subtracting, by giving you twice the original image, minus a little bit of blurring. So what's happened here is this is our Lena again. And if you compare this one to this one, you'll notice that there's a little bit more bright stuff out here than there was there. And you'll notice that this picture is, in fact, sharper than that picture, and that's because we've accentuated the high frequencies. And one more here. So here's a cool texture. Here's our brick texture, right, that has mostly these lines here but a few of these there. And what you can see is you basically have just the verticals here, horizontals there. Okay. And then in this picture, okay, you've got all these angled lines and that's what's showing up as all these angled sinusoids in here.

15 - Man Made Scene

Let me show you just more example just to highly a particular little problem. So here we have a nice man-made scene of a some beautiful bridge somewhere in Europe. And you'll notice that there is this strong vertical here. And a vertical if you think about coming down this way and coming that, is a step edge horizontally. Okay. You remember, dots like this are sinusoids that, that vary as you go up and down. And you remember that if you have a step edge, remember we had a square wave, okay? It had all the frequencies, just in decreasing power. And what you're seeing is a whole bunch of frequencies in decreasing power coming down. So the question is, where is the strong horizontal edge that's suggested by that vertical line? Remember a vertical line is all of these frequencies, which means that it's got a horizontal sharp edge, and the answer is a little tricky. Basically, if you think about how the sinusoid goes in the image, it's essentially as if it assumes the image just kept wrapping around, okay? Remember we talked about the Fourier series, assuming that everything just kind of repeats? So here what we're essentially doing when we do this sum from minus n over 2 to n over 2, is we're assuming that the thing continues, that it's periodic. Which would mean that the thing over here should be the same as the thing over there. And if it's not, that would represent a significant change. And, this edge down here, is really different than that edge down there. So when you wrap it around, you get a big step. And that's what cor, causing that sort of weirdness in the Fourier transform. Do you remember when we talked about doing filtering in MATLAB? There was, there was a wrap option. To head out off the edges instead of replicate or reflect. You could take the stuff from the top and put it back down on the bottom. And I said that won't make any sense to you at all until we get to Fourier analysis. Well, that's as if you had a periodic signal. Because if you don't have that, you get this funny sort of sharp edge. One way to eliminate that when you're doing Fourier analysis is you'll take you're image, and you'll multiply the whole image by a Gaussian that tapers off towards zero by the edges, and then everything becomes nice and smooth. When in doubt, fix your life with a Gaussian.

16 - End

That ends the first lesson on frequency analysis, it gives you some relation between sort of what's in the image, and the frequency components in it. Next time, what we're going to do is we're going to talk about how filtering and convolution are expressed in the frequency domain and that's how it allows us to get to aliasing. I'll also say if you didn't follow everything that just went through, that's okay, I really just want you to sort of stay focused on this idea of frequency, and the con, and the Fourier transform. And then, as a way of, of getting the different elements. And then I'm going to show you how the frequency transform relates to sampling, which, or I should say, frequency domain relates to sampling, which will get us to aliasing.

2C-L2 Convolution in frequency domain

1 - Intro

All right, welcome back to computer vision. We're going to to continue our discussion about frequency analysis and how to study images with notion of frequency components. If you remember last time we talked about the idea of decomposing images based upon frequency. And we motivated it by showing you this picture. This is that famous Dali picture which when you allow the high frequency imagery in there, you see a picture of a woman admiring the, the Mediterranean or something like that. But when you blur it, that is when you remove the high frequency you just have the low frequency, it becomes a picture of Lincoln. And this idea of how do we decompose images into thinking about what was going to be a basis set of that involved frequency and the idea that a basis set was going to be some set of functions that we're going to linearly span our possible signal. So the idea is that we should build and make any possible signal out of this set, and that they are linearly independent so I can ebb which basically allows me to do this summation. We're also going to end up within our Foginal basis set so that when you change the amount of one basis element you don't get any change in the others. And the basis set that we were talking about was this Fourier set, right? And the Fourier set basically decomposed things into sinusoids and they were sinusoids here of increasing frequency that were sort of moving vertically and horizontally, and the idea is that we could construct images doing this. So related to that idea was this notion of what we call the Fourier Transform, and the Fourier Transform was that you would put in a signal, showing it here in 1D, but also talk about it in 2D, and what you would get out was instead of f as a function of x, that is the intensities of function of location, you got out a F, cap F, as a function of I L Mayer, omega was frequency. And the idea is that cap F would decompose into an even component and an odd component being represented by complex numbers, and so this would allow us to represent both the magnitude and the phase for a given frequency. And if you wanted to, you could recover that magnitude and phase. The magnitude is just the square root of the sum of the squares of each of the real and imaginary component, and the phase was just this arc tan of this relation. To do the Fourier Transform is represented by this sort of ugly looking integral. Here I'm showing it to you in 1D, we'll do it again in a minute in 2D. But the idea was you took your function and you integrate from minus infinity to infinity. And you multiplied it by sinusoids of whatever frequency and sometimes frequencies omega. When we're going to do it for two dimensions u and v, so here's it's written as u. And even though this does not look like a sinusoid, you had to remember that e to the i k was cosine k plus i sine k. So this is what allowed us to go from the spatial domain to the frequency domain. We also mentioned that you could inverse this, if you wanted to, right? So if you give me the Fourier transform f of u of a signal thereby just changing whether that's plus or minus, I can do the integral and get back the original signal. But now we have both u and v. We also talked about the discrete version. And we showed some examples here where you're taking the sums instead of the integrals. And most of the time, we're mostly interested in the power. Remember, the square root of the sum of the squares? And that's represented here. So I might have a, a signal that has a certain amount of even part and an odd part, or the real, imaginary, you take the square root of the sum of the squares and you got the magnitude, and the magnitude of the power is what we'll typically be talking about when we talk about the presence of frequency with any given image. We want to know the power of it. The phase is important for doing reconstruction but in general, we're not going to be doing a reconstruction. So today, what we're going to do is in this lesson, is we're just going to relate the Fourier transform to convolution. And in particular, how it's, how convolution in space is multiplication in frequency. And let me show you how that works.

2 - Fourier Transform and Convolution

Let's suppose we have a function, g, which is just f convolved with h. Maybe h is a filter, f was the original image, doesn't matter, it's f convolved with h, and suppose I want the Fourier transform of

g. All right, well, that's going to be written as follows, 'kay? Our definition right, G of u, is just the Fourier Transform of g of x. But g of x is just the convolution, so that's sort of in here, this part here, of f with h. Remember that was my definition of G, the convolution of f with h. What I can do is I can rearrange this, and you'll see what I did here was, this was u x, so here we have u tau, and here we have x minus tau. All right, and so if you multiply those together, you would just get u x, so it's the same thing. But I've rearranged it, so all the tau stuff is here. And all the x stuff is there, with this tau in there. And so, we're going to make a quick change of variables, right? I'm going to say, let's define a new variable of x minus tau is x prime. So, now I've just, just converted that to x prime. So, now I've got one integral here, and one integral there. And if you stare at those, and you ask yourself, get ready. What are those integrals? What are those integrals? Well, what are those integrals? Well, let's take a look at just this integral. That's the fourier transform of f, right? You happen to use tau instead of x, but it's the same thing. This is the Fourier transform of H, so that can be written as just F u times H u. Megan laughed when I said F u, by the way. So G of u is just F u times H of u. In other words, remember G was the convolution of F with H. G of u, the frequency domain is just the product of the Fourier transforms of F and H, okay? So the idea here is that when you do convolution in the spatial domain, you're essentially doing multiplication in the frequency domain. And this is going to be really important, for the next lesson when we talk about aliasing, and also for what we're going to talk about here in terms of how convolution does frequency manipulation. So this is just written here, is that if in if the spatial domain I do convolution, in the frequency domain, I'm doing multiplication. Also by the way, because of the symmetry of how the transforms work. If I were to multiply in the spatial domain, that is, if I took two functions and I multiply them together, I don't convolve it, I just multiply them. The Fourier transform of that new function is going to be the convolution of the Fourier transforms of each of the individuals, okay. We're going to make use of that, both here, and a little bit when we talk about aliasing. So, in the spatial domain you do convolution in the, in the frequency domain it's multiplication or vice-a-versa.

3 - FFT

That actually means that, and this was more important a little while ago, but people still use it today. If I wanted to let's say, do a convolution of a function with some really big mask, which is generally expensive operation, I actually could do it a slightly different way, all right, and that's shown here, right. Suppose I want, g to be f convolved with h, let's suppose f and h are both very large things, so you remember convolution is ugly cause it's N squared M squared where N is the size of the image N by N and M by M, well so what we can do is, we could actually take the Fourier transform of F, and a Fourier transform of H. And, there are fast ways of doing Fourier transform called the FFT for fast Fourier transform which we won't talk about, but I can do those Fourier transforms quickly then I can multiply, those two Fourier transforms together to get the new Fourier transform, and then I can do the inverse Fourier transform to get back the answer. Like I said, this was really important when machines were slower, these days people tend to work in the spacial domain, but even today, if you have a large mask or some large function you're trying to convolve against a large image, it's not uncommon to use the Fourier trick to do that.

4 - Smoothing and Blurring

But we're not really going to talk much about doing this inverse part. Instead we're going to talk about how the Fourier transform shows you what's going on in frequency. So, suppose I have a function, and here's a function. And suppose this is this noisy signal. I want a nice smooth version of it. Right. So how might I do that? Well, we learned before that you might blur it with, let's say, a Gaussian kernel. Okay, and what does it mean to blur it with a Gaussian kernel? We're going to convolve the underlying image with that Gaussian. But it turns out the Fourier transform of a Gaussian is a Gaussian. All right. And that's what this shows here. So the Fourier transform, if this is h of x, this is h of u, it's another Gaussian. But you'll notice that here it's over sigma squared and here's it's times sigma squared. Okay. And that's what this is trying to show you here is that if I have

this Gaussian kernel with a variance sigma squared then I get a new Gaussian, and I apologize, this is an awfully drawn Gaussian, I get a Gaussian whose variance is 1 over sigma squared. Now, if you think about that, that make a little bit of sense, right? First of all, let's think of a really, really skinny Gaussian. All right, like, almost an impulse. Remember, when you convolve a function with impulse, you get back the original function? So that means you basically keep all the frequencies. So if my kernel was a very tight little Gaussian, I would expect to be able to keep all the frequencies I've, oops, don't, there we go, I expect to be able to get almost all of the frequencies back. Likewise, let's suppose my kernel was really, really fat. Blurs the heck out of everything. Well then, I would expect to get hardly any high frequency and only to get a very, only the, the low frequencies. And that's why the Fourier transform, that's one way of thinking about it, that the Fourier transform of a fat Gaussian is a skinny one, and the Fourier transform of a skinny Gaussian is a fat one. Remember, they, they go back and forth. All right. So, what's going to happen when we blur this function with the Gaussian is we're going to multiply its Fourier transform by the Fourier transform of that Gaussian. And what it's going to do is, it's going to keep the low values and it's going to lower down the higher frequencies. It's going to reduce the higher frequencies by doing that multiplication. And we say that that means that it attenuates, that's the word we use, it attenuates the high frequencies.

5 - 2D Example

So I can show you an example of this being done in 2D, that is, with an image, here. Okay? Here we have a, an original image, a beautiful little still life of some fruit in black and white. And here is its Fourier transform power spectrum, just the magnitude, so I'm not showing you the phases. The middle, remember, we took, we put the origin in the middle, is the low frequency. And you can see that it falls off as you get out to the outside. Now, we're going to blur this thing with a Gaussian. And I'm showing you this Gaussian here at the same scale as the whole image. So this Gaussian might be 15 by 15 pixels, but it looks really small because we have a good sized image. But it's get, that would blur the image a little bit. And when I convolve the Gaussian with the original image, I get exactly what we'd expect, a slightly blurry version of the image. But look what happens in Fourier space. The Fourier transform of this tight little Gaussian is this big fat Gaussian. Remember, we said, small Gaussian in space has a big Gaussian in frequency. And then we multiply this times this. And what that's going to do is, it's going to keep the areas in the middle, which you see here. But it's going to attenuate the areas on the outside, so it reduces those low frequencies. And in fact, if I had wanted to, in order to recover this image, what I could have done was taken the Fourier transform here, multiplied it by this one, gotten this, and recovered back. Except that to actually do the recovery, it wouldn't be enough to just look at the magnitude. I would actually have to keep the real and imaginary parts. I'd have to keep the phase of the transform. But here, I just wanted to show you what happens to the different frequency power.

6 - Low and High Pass Filtering

We've actually looked at this just a little bit before when we introduced the notion of frequency. So here we have a picture of this, this yeah, we did have this picture, this man made scene, all right, and here what I did was, I took the, inside part, and I set this all equal to 0. So instead of tapering it off like a nice Gaussian, I was ruthless and I set all of the, the frequencies outside a certain diameter to 0, so there's no high frequency component at all. And what you get, is this ugly looking thing called ringing, al lright, so this actually has low frequency, but because of some slight variations in here, it can't cancel them out with higher frequencies that it needs, okay so what gets left over is what's called ringing, and that's what happens when you clip the frequency precisely and I'm going to make that a mathematically and better defined in just a minute. The flip side of that, is if I remove, just those center frequencies, and I think in the image processing world, this is what we call coring, where you pull out that center, all that you get left with is this high frequency bit, which is a lot like, what, the edges, okay, we know that there's this relationship between frequency and edges at the high frequencies is where you get these steep derivatives. All right.

7 - Properties of Fourier Transform

Let's continue looking at this relationship between the spatial domain and the frequency domain. Here's some simple properties of the Fourier transform. We started with one that we already talked about. Everything is linear because we're just doing sums and multiplies, all right? The one we just described was this convolution one. That convolution in the spacial domain is multiplication in the frequency domain and vice versa. One that's kind of interesting is scaling. So, if I scale a function by a constant a. So let's think about it this way. Suppose a was greater than 1. Let's suppose a was 2, all right? Well, that would mean that when x had been 1, now x is 2, so it, it would sh, shrink, right? When you multiply by a number bigger than 1, right? So if I had some function, right? Now what happens is this value gets moved over there, so I essentially shrink that function, okay? So when I shrink a function things are going to now wiggle faster, right? because they were wiggling slower before. If I pull it in it now wiggles faster. And so what happens is I end up stretching the Fourier transform. And this is the formula for it, 1 over a times F of u over a. So it's got that inverse property. Which is just like that Gaussian idea also, right? If I make it fatter, my Fourier transform gets skinnier. One last one to look at, and we're not going to use this much, but I want you to just see this is. When you take the derivative of a function, so this is some nth derivative. You multiply its Fourier transform by a function that's proportional to the frequency. So suppose n was just 1 and you're taking the first derivatives, right? What that says is that you would be multiplying F of u, times u. Which means the higher frequencies get accentuated. Which was exactly what we saw when we took derivatives. Remember when we added noise, high frequency noise, and then we took the derivative, the whole thing blew up? That's because you're multiplying it by, when you take this derivative, you multiply the high frequency components.

8 - Fourier Pairs

If you take a look at Rick Szeliski's book which we mentioned earlier, you can see a whole bunch of Fourier Transform pairs. For example the impulse, which is drawn like that has a Fourier Transform of 1. As we said that makes sense, because the impulse when you convolve it, just gives you back the original function, so you'd better keep all of the frequencies. Likewise, we said here that the Fourier Transform of a Gaussian is another Gaussian. But again, G of sigma goes to 1 over sigma. So fat Gaussian in space, skinny Gaussian in frequency. But look at this one. If I have a box filter, which is just a filter that's shaped like a box, it produces what's called the sinc function. Okay. And that's drawn here. And let's look at that a little more closely, okay? So here in this spatial domain, we have our box filter and this is our sinc function. All right? Now sinc of x is sine x over x, all right? And we can think about that, if you take the limit as x goes to zero of the, you remember L'Hopital's rule take the derivative of the top that's the cosine. Derivative of the bottom is just one plug in a zero it's one. So the, the limit as x goes to zero is in fact, one right there. And then as x gets bigger, the sine gets attenuated and that's what you're seeing here. So in other words, if I were to convolve something with a box filter. My frequencies are going to behave kind of in a weird way. You know, I have this nice kind of low pass stuff, but then I'm going to have these various other frequencies that are going to get accentuated. And if you remember, we did that very early on when we introduced filtering. Remember, we convolved something with a box, with a square? And we said, man is that ugly? This is why it's ugly. So you see this picture here? You see how it's got all these little striations. And I said, when we first did this, that that's because boxes aren't smooth and that we would make this more formal when we did Fourier analysis. Well, I made good in my promise. Okay? The reason that when you convolve an image with that little square box that you see all this junk is because of this ringing here in the frequency domain. That's why we said, what you should do is do what? Is you should apply a Gaussian not a box, it makes a nice, pretty picture, you know, a nice, smooth picture. Because in the frequency domain, you get this nice, smooth Gaussian. One more thing about these sinc functions, okay? So we showed that the Fourier Transform of a box is the sinc. Well, because of the symmetry, if I had a box in frequency space, that would be a sinc in the spatial domain. So what if I were to take some function, take its Fourier spectrum and multiply it by that box, what would that look like? Well, I

already just showed you that. Right? This is essentially, it's sometimes called a pillbox, because it's a box. But in 2D, that is instead of being just a particular width, it's a radius. Right? So everything outside that diameter is set to zero and everything inside is kept exactly. If you remember the picture we got was this ugly picture with all this ringing in it. Well, now we know why. When you multiple the frequency by a box, it's as if you convolve the image with its Fourier pair over here with the sinc function. So if you take this pretty picture and you convolve it with this ugly beast, you get something that looks like that. And that's where the ringing comes from. That's why you never want to just cut out the high frequencies, because what you're essentially doing is convolving your image with a sinc function.

9 - End

That ends this lesson on the relationship between convolution in the spacial domain and the frequency domain. In particular the convolution in the spacial domain is multiplication in the frequency domain. We've also made good on the promise of explaining some of some of these ugly artifacts that we saw particular ringing is sort of a Fourier transform of a box that is if you cut off the frequencies hard. You get this ring or if you have a cutoff in space that's hard, you get this ringing in frequency. What we now have are the tools that will allow us to tackle aliasing. That is when you do something to an image that involves sampling that causes it to look really ugly. And the high frequency stuff. It's a little bit of of a bumpy ride, so strap yourself in and get ready for the next lesson.

2C-L3 Aliasing

1 - Intro

All right, welcome back to Computer Vision. This is going to be our last lecture or lesson or whatever this is on frequency analysis. And I bet you're happy about that because it's kind of off in the ozone of mathematics and it's going to, hard to pull it back to the work that we'll be doing in terms of manipulating the images. But it's important and in fact today we'll finally get to the, what is coup de grace, piece de resistance my, my french is terrible. The fine, what I really wanted to get to is the notion of how frequency explains the notion of aliasing. So if you remember we've been talking about Fourier basis sets where we decompose things in terms of sinusoidal basis. And this is a picture that we've been using to represent that. And then we talked about the relationship between convolution in the spacial domain giving you multiplication in the frequency domain. And we showed it with the math here. We said that if g was the convolution of some functions f and h, that we could derive that convolution in the spacial domain was multiplication in the frequency domain. And that was shown like this. And, by the way, then this is really important for today that multiplication in the spacial domain is convolution in the frequency domain. And you might ask, when do we ever multiply functions together. When do ever multiply functions together? Today. It'll be some very special functions, but we'll do that today. We also showed some Fourier pairs. We talked about how the Fourier of a Gaussian is a another Gaussian. We talked about an impulse is straight. We spoke about the box filter giving you this sink thing that explained ringing in that like.

2 - Fourier Transform Sampling Pairs

But there's one other pair that I, I need you to know about. And again we're not going to derive this but it's important. And that is the notion of a pulse train or a set of impulses, okay. And so this is refered to as a pulse train, you can see that here. And the Fourier transform of pulse train is another impulse train. And the thing that you need to realize is that, and remember the scaling theorem give us this and a bunch of things give us this, right? As the pulses in space get further apart, the pulses in frequency get closer together. And this is a hand-wavy way of thinking about this. Imagine that our pulses got very, very far apart, okay? So far apart that I have an impulse in the middle and the

next one is infinitely far away. Well, that would be just like having sort of a single impulse. That's the hand-wavy part. Well, what is the Fourier transform of a single impulse? Well, we said it was a 1 because an impulse gives you back the, when you can involve it, it gives you back the entire image. So therefore the frequency has to keep all the frequencies. So the hand wavy way of thinking about this is, as I stretch out my impulses in space, I have to shove all these impulses and frequency in time infinitely close together to make a solid bar. Okay, and that's one way of thinking about how the Fourier transform of an impulse train is another impulse train, and as they get bigger in space they get closer in frequency.

3 - Sampling and Reconstruction

So now we can start to talk about sampling and aliasing. So before we do that, we have to talk about the notion of sampling at all and the notion, and, and the idea of reconstruction. So here we have a the idea of some nice smooth scene illuminated by a light source and then we have an imaging system that captures the intensity at a bunch of discrete places. And of course, what we'd like to do, is we'd like to take the captures from these discrete places, and be able to reconstruct this nice, smooth signal. That is to know what that signal is. Let's talk a little bit about sampling. Sampling arises when we come up with this notion of, how do you store a continuous signal in a computer, right? So here we have a continuous signal, and this is in 1D, and we're going to do a lot of our stuff in 1D today because it's easier to draw. And so the obvious way of representing that, it's actually not the only way because you can imagine different clever basis kinds of things, but sort of a, a normal way of thinking about that is by samples. That is, you just take the samples at each different location and you write down what the value is. And once you have that set of samples later what you want to do is reconstruct what the original signal was, right? And why do you want to do this? Well, maybe you actually want to use the signal, you want to play it. If this is audio, you want to play it through a speaker. And I don't need samples, I need a continuous voltage. Or maybe I want to do some sort of analysis or a mathematical processing in terms of thinking about what this signal is. So essentially, what you have to do is you have to guess what's going on between each of the samples. That's all you can do. You have the samples at discrete places but you need to connect the dots if you will. You need to fill in what's going on. And a lot of what we're going to be talking about today has to do with what's actually going on, when you think about taking a set of samples and then trying to guess what's in between. So to illustrate most of these points we're going to use a 1B signal, so you think of this like audio, all right? So here's an interesting audio signal. You see this signal? This, so this is time, okay, it goes this way. And it starts off with a low frequency and it gets higher and higher and higher and higher, higher, higher and stops. All right, if you were to play that sound, you would hear what's called a chirp. Okay, it's actually what it's called, it's called a chirp, it comes from radar, et cetera, whatever. It's something that has power in lots of different frequencies so when you send a chirp out, no matter what the frequency response of the thing you might be hitting is, you'll get something back. It also, it, it's also a nice way being able to localize it. There are lots of good reasons to use what's called a chirp. But here's a chirp that has higher and higher frequencies. But it's a good signal to think about looking at aliasing.

4 - Sampling in Digital Audio

Continuing with thinking about the digital audio, we can think about sort of music or, or any other way of dealing with audio right? When we record a sound, we have a microphone and a microphone generates out a continuous voltage. By the way, you guys know how a microphone works? More or less a microphone works the following way. We have a permanent magnet and we've got a electric coil alright with some wires in it, and out, out of that coil comes another wire. And usually, the the coil is on a little springy mechanism. The permanent magnets are fixed. You speak and that causes a pressure wave, a press, a pressure wave causes the coil to move back and forth. When you move a coil of wire through a magnetic field, that induces a voltage. So that voltage comes out on that wire, and it's a continuous signal. But, in order to encode this into a computer, we put it through what's known as an A to D converter, analog to digital converter, and

we get out a bunch of samples. And then, by the way in the old days we used to put those samples on something called a CD. Compact disk. You guys probably don't even buy those anymore. If, if you do, you only buy them to rip them. But, basically, it's a set of samples, it's in a file, whatever it is, okay? When you go to play this thing, okay, what you do is you put your CD into your ancient CD player. It pulls out the samples, and then it goes through what's called a D to A converter, digital to analog converter. And then we take that continuous voltage and we put that out through a speaker. Which by the way, is almost exactly the same as a microphone. Right. It's a coil of wire, you pump electricity through it. It generates a magnetic, varying magnetic field. There's a set of permanent magnets that causes the speaker to move, with respect to those magnets. And voila, out comes a continuous pressure wave. So, let's talk about this sampling operation and then the reconstruction. So, here we'll take as an example just a sign wave. Okay. So, I collect a bunch of samples. If I collect a bunch of samples, then what am I going to do to reconstruct it? Well, it might be enough to basically just connect the dots and get back the next sine wave. Alright, and so, as long as I had enough dots, that's easy enough to do. So I have enough dots, I connect the dots.

5 - Undersampling

But what if we missed things going on between the samples, what if we don't have enough dots? All right. Well, not surprisingly something's getting lost, right, some information is getting lost, right, so here we're showing you right so I've got a dot here, I've got a dot here and there's all this change going on in between, that is not realized or I should say not captured. Perhaps, the surprising result though, is that actually this is indistinguishable from a lower frequency, okay, so here what's drawn in is, this the same set of dots, that we had before, but there's a low frequency sinusoid that fits through there, 'kay. Continuing with that, with sort of a surprising result is that, the low frequency that was always indistinguishable from those high frequencies, if you only gave me a small number of dots. Okay, that is, I can't tell, whether or not it's a high frequency or a low frequency, and that's the definition of aliasing, or I should say that's an example of aliasing, the definition is it's the signal that travels sort of in disguise of another signal or of other frequencies. All right. Now we're going to make this formal in just a minute, but that's what's going on in aliasing is that I've got a signal and I can't distinguish between a low frequency that was in there and actually a high frequency

6 - Aliasing

By the way, this also happens in time. Have you ever watched a video where an airplane propeller is starting to go? And then all of a sudden st, you start to kind of see it's spinning backwards, and then you see it spinning forwards. Okay, airplane propellers don't spin backwards. They only go forwards. So what's going on there? Oh, and by the way, I hope you know that if you were actually standing out on the tarmac, watching the airplane, you wouldn't see this phenomenon. The reason you're seeing this phenomena is that this was recorded on either video or film that was taking a picture every so often, and that's what's displayed here. Imagine we have this wheel that's turning this amount each time. So if you track the dot, you can see that the thing is turning almost 90 degrees at every rotation. But, if that dot wasn't there, you would actually see this thing going backwards just a little bit. Because of the fact that you couldn't tell which one of those cross patterns was which, and so you would see it rotating backwards, okay? And that's aliasing in the temporal domain. We'll, we'll talk about that in a minute in the spatial domain. And essentially, the thing is moving too fast for how often you're sampling a time for you to actually be able to tell what's going on. The high frequency and the low frequency can't be distinguished. To show you a simple example in an artificial image, we'll see some natural ones later. Here we have a rendering, and as this checkerboard gets further away from you in the distance, right? It's supposed to start getting thinner and thinner and closer and closer. And you'll notice, that somewhere right around here, it starts to break up. It's not looking like a checkerboard look. In fact, look at this nonsense, right? It's like low frequency all over again. What's going on? Well, we can look at that in MATLAB, all right? Suppose we have this input signal. Okay, this is like that chirp that I was

showing you before. Well, we can plot this as an image in MATLAB, and that would look like this, okay? So here I'm just doing the x from zero to five by 0.05, and I'm, I did this image of sine of 2 to the x times x, all right? And, what you're seeing is, so it starts off a slow frequency, and then it's getting higher and higher and, eventually, you start to see stuff that looks like that. So what's going on there? Well, that's aliasing. There are not enough pixels in the, in the plot for you to be able to see what's going on. And that's illustrated down here, where we have a small number of samples, a not dense enough sampling, in order to recover those frequencies.

7 - Antialiasing

So the question is, how can we prevent aliasing from happening? One is we can get more samples, okay? We can join the megapixel craze in the current video technology. My birthday's coming up this month and I treated myself to a new camera, and the camera has 36 megapixels in an image, all right, so that I can pull out really fine detail in a, in a, in a portrait. But, in general, this sort of the megapixel thing doesn't go on forever. You know, stuff always can or will go faster and faster as I think, as things get further away, for example. So what do you want to do, well, what you want to do is, you want to make sure your signals are less wiggly. That is, they don't have all that high frequency component. You want to get rid of some of that information. So the idea is, we're going to get rid of some of the high frequency information, but that's going to be better than aliasing, all right? The idea is we don't want to see that weird checkerboard effect, right. So we're going to remove some of the high frequency, but the idea is that the thing will be well behaved. So let's talk about that real quickly in our audio example. What we're going to do is, we're going to introduce lowpass filters. And what the lowpass filter is going to do is, we're going to put that right here. So where the analog voltage is coming out of the microphone, so the signal that goes into the A to D converter, doesn't have frequencies higher than a certain amount. We'll say, okay, great. So that says that we can reduce the number of samples we need to take, or limit the number of samples we need to take, and then when we do the reconstruction, we'll know that anything that was reconstructed that was of a higher frequency than we let in should be thrown away. So we use the lowpass filter again for the output to the speaker.

8 - Impulse Train and Bed of Nails

So that's the hand wavy version about aliasing. Let's be big boys and do, and girls. Let's do this in the frequency domain. A little bit of math. The first thing we need to do is define a comb function, which is the same thing as an impulse train. So a comb function is written this. I'm not sure we've defined what this chronic or delta function is. Basically it's a function that's a pulse of one when its argument is zero and it's zero otherwise. And what this would mean here is as k goes from minus infinity to infinity this thing would count up by M's so x every M would be a one. So if this is two, then this would be a separation by twos of, of this impulse train. Okay. So if M gets bigger, my pulses get further and further apart. 'Kay, that's an impulse train. As I said before, the Fourier transform of an impulse train is another impulse train. But also as I said before, as the spacing gets larger in space it gets narrower in frequency. Remember that was the scaling property of the Fourier transform. So the less often we sample in space, the higher the samples in frequency. Okay? So that's in 1D. We can also do impulses in 2D. And that's called a bed of nails because instead of just having a train of pulses that land on the integers or, or some multiple of integers in 1D, it's actually lying out on all the discrete coordinates in 2D which if you're in a macabre sort of mood you can think of as a bed of nails. So that's written here. Comb of M, N because we have a separation both in the x direction and in the y direction. And also in the same way that a, the Fourier transform of an impulse train is an impulse train. The Fourier transform of a bed of nails, is another bed of nails. And again, as the nails get further apart, the Fourier ones get closer together. And you can think of it from the same reason, if I spread the nails out infinitely far away, and I get just a single pulse in the middle of the bed, I would not recommend lying down on that single nail. That would hurt but, if you think of it as an image, it's a single impulse. It would recur, it would, if you convolve that with your image, you'll get back the entire image. So you have to cover all the full plane in the

Fourier space, okay? And so you have to shove all those Fourier nails back together to make a, a flat board that you could lie on.

9 - Sampling Low Frequency Signal

For the discussion about aliasing, though, I'm just going to talk about this in 1D, because it's a lot easier to draw the pictures and to make it clear. So, hang on, what we're going to do is, we're going to now talk about aliasing in the Fourier space, and we're going to talk about sampling a signal. And we'll first talk about sampling a low frequency signal, and then a high frequency signal. Here we have a function, so here we have some nice function f. And it's a nice smooth function, let's say, all right. And it has some Fourier spectrum like this, F of u. And you'll notice that it's kind of limited, right, it doesn't go, it doesn't have very high frequency components, like, no high frequency components, very, very low. And here's our comb function pulse train, okay. And, as we said before, the Fourier transform of that pulse train is another pulse train. And if the separation of that pulse train in, in space is M, then in frequency it's 1 over M. Now comes the point that's not so obvious. Suppose I want to take samples of a continuous function, right, I want to take a sample here, and a sample here, and a sample here, and a sample here, and a sample here, et cetera, and I wanted to do it with a spacing of M. How would I do that? I actually have what I need here, I just multiply this times this, and I get this. All right? That is, here is just the set of samples, multiplied by f. Okay? So sampling is just multiplying your continuous signal by this discrete comb. This is is why we had to introduce the notion of multiplying functions, and the notion of a comb. Sampling is the multiplication of your signal by a comb. Now, when I multiply in space, what do I do in frequency? I convolve. Remember, convolution in space is multiply in frequency, and multiply in space is convolution in frequency. So what I'm going to do is, I'm going to convolve the Fourier transform of f with the Fourier transform of the comb, I'm just showing the power here, and I get that. Okay, so this is the convolution, and you'll see, if I just take this function and I slide it along, I can re, I'll reconstruct it here, reconstruct it here, reconstruct it there, and that's what this is showing you, okay. So that's the Fourier transform of the multiplication of the comb times the function. Just copying that bottom row. If this function is limited, in fact, I can show it down here like this, all right, then if I could pull out that part of the signal, the part of the signal that's frequency is just within some range. Then I would be able to get back what I wanted, right. I would be able to get back the original spectrum, which would give me back the original function. So this is no problem if this outer edge is small enough, that is, that the, the maximum frequency of my signal is low enough for the comb filter that I used. Exactly what is that small enough? Well, here it's written like this, okay? If my comb filter is spacing M, so then the comb of my Fourier of that, of the Fourier of the comb is 1 over M, half this distance here is 1 over 2M, okay. If the maximum frequency of that function, let's say is W, for bandwidth, is W, if that's less than 1 over 2M, then I can reconstruct that original signal by looking at just this part, that is, nothing has contaminated that inner part. So if there's no overlap, if W is less than 1 over 2M, the original signal can be recovered from its samples by low-pass filtering, which we'll talk about in a minute. So those of you that know something about Nyquist sampling, so you know that if I want to recover, say, something up to 20 kilohertz, right, 20 kilohertz, about the, the extent of human hearing, I would need to sample at least 40 kilohertz. By the way, CDs, like we used to, were talking about, sample at 44 kilohertz, why? So that we can recover everything up to 22 kilohertz, which is the extent of human hearing.

10 - Sampling High Frequency Signal

So, as long as my frequencies are low enough, I'm okay. But what if they're not low enough, what happens? Well now, let's suppose I have higher frequency thing, so this is some random function, and it has some high frequency. So, here is its f of u, and you see its w goes out a little bit further. If now, I multiply that by the same comb of M, I'm going to convolve this thing with that 1 over M comb that I had before, and I'm going to get this part here. Now remember, these things actually sum. I'm drawing each individual component, but in reality, this thing would sum and come back up here, all right? So, these things would sum and come back up there. And this area right here is

where this is, essentially, this energy has folded back on itself, it's overlapped and then they're going to add, okay? That high frequency that was really this little edge thing here, okay? Or, it's the thing that's there, is pretending it's low frequency. That's the aliasing. And once you've aliased the signal, once you've done this operation, you cannot undo it. The high frequencies have pretended that they're low frequencies, and you can't separate them out from real low frequencies. You have to remove the high frequencies before you do the sampling, in order to remove the aliasing. So, how might you do that? Okay. Well, how do you remove high frequencies? Well, we know how to do that. We convolve with h. Maybe its another Gaussian. What does that do? That reduces the high frequencies. So the, the frequency spectrum of the convolution of f and h. It has much less high frequency. So, that h is what's referred to as an anti-aliasing filter, and it has the net effect of trimming the high frequencies of the original signal. So that then, when I do the multiplication by the comb, which is shown here, right, I'm now convolving that with a comb filter in the frequency domain. But, I don't get that overlap, right? So no aliasing goes on, all right? And then to recover the signal, whatever I get after I reconstruct from that using the new sampled, convolved function, I can throw away any high frequencies because those are not real. As we, so just the way we did with the, the CD right? We, we low pass filtered the microphone. Did a A to D conversion to get some digital samples. We did a D to A conversion to get an analog. And we knew that any high frequency that showed up, that's not real. Okay, we could low pass that away and recover the original signal. So, that's summarized here in terms of aliasing. When I don't have an anti-aliasing filter, I get these overlap regions that cause me to get aliasings that I cannot remove once I've done it. But, with an anti-aliasing filter, I don't get that overlap. So, that's the mathematics behind aliasing.

11 - Aliasing in Images

We can see this going on in images. So here's a nicer picture showing you aliasing in images. So you can see what's happening is that this is a radial sine wave, and you can see as you get closer and closer, the pixels are supposed to be wiggling faster and faster. And eventually I run out of pixels. Okay, I don't have enough samples to get how quickly the thing varies. And so that is an example of aliasing here. So, what is its impact on you? Well, its impact on you is you have to be a little careful how you treat your images, for example, and this comes from Steve, all right? Suppose I've got this image, a beautiful Van Gogh picture. Suppose it's too big to fit on a screen. I want a smaller version of it. How can we reduce it? I don't know. Make it half the size or quarter of its size to do that. Well, one obvious way would be to throw away every other column and every other row, all right? Just throw those away. And that's called image sub-sampling. Now by the way here it says by half, what it means is I throw away half the columns and half the rows, which of course is actually a quarter size smaller, right. But half meaning, half the rows, half the columns. If I did that again, I would now have one quarter of the total rows, one quarter of the total columns, one-sixteenth the size. I could do it again, one-eighth. In fact, you might start to notice this is already looking a little ugly. And in fact, to make it clear exactly how ugly it is, let me blow up the half, the quarter and the eighth by zooming in on both of them just blowing up the pixels. And you'll notice, this is atrocious, okay. The, the one-half one doesn't look too bad, but remember here we've only taken one out of every eight columns and one out of every eight rows. And then we blow, blew the thing back up. And you can see that this does not look like a blurry version of the original, okay. And the reason is we have aliasing. We have sampled too infrequently for the amount of variation that there is. Essentially, things wiggling here more often than one out of every eight. And so if I just take one eighth, right, I'm not capturing all the wiggles. So what's the right thing to do? Well, we need to do that anti-aliasing, we just learned that. So, a simple one is to do Gaussian, okay? So, what we'll do is we'll filter, right, so we'll take the Gaussian and then take every other row, all right? And then if you want to be not so clever about this, you could do a Gaussian of that, and take every other row of that, to get the one quarter. And you could do the Gaussian again, and do that. Or, what you could do is you do a much bigger Gaussian to begin with, and take one out of every eight. You can convince yourself that mathematically these are going to be equivalent. When I do that, and I blow up the Gaussian one-eighth, you'll see that this actually looks pretty much like a

blurry version of that, right? Compare this to what we did with the subsampling, right? So here's sort of our original. Here is the Gaussian eighth 4x zoom. And here is the subsample 4x zoom. And you can see that the Gaussian does a much better job doing the reduction and that's because it's been anti-aliased. So if you're doing Computer Vision and you decide, you know, what I've got this huge 1920 by 1080 image coming in, but I'm just trying to recognize faces, all right, and my faces are going to be, you know, 100 pixels big let's say. And you say, you know what? What I can do is, it's going to be expensive to do that face recognition. We're going to talk about face recognition towards the end of the class. It's expensive to do that over the entire image, when it's so big. Let me make the image, you know, one-eighth its size this way, one-eighth its size that way. And, I'll make my faces smaller, and look for them that way. You know what? That will work pretty well, if you do this, if you do anti-aliasing. If you just pull out the arbitrary pixels right, you're going to make a mess of the whole thing. You'll get aliasing, and then the faces won't look right. So that's why doing this kind of stuff matters even in computer vision.

12 - Campbell-Robson Contrast Sensitivity

That almost ends our lesson on frequency. There's just one more thing that I wanted to talk to you about today, because it's so prevalent in the processing that you're going to be doing. Some psycho physicists, Campbell and Robson studied a long time ago the contrast sensitivity of the human eye, human vision system not just the eye. Now what you're looking at here is a curve where frequency goes up to the right and contrast goes up down. And probably, if you're like me, hopefully you're not like me, one of me is way more than enough, as my children will tell you. But anyway, if you're like most humans, you probably can see this stuff really well, but not so much up there. And somewhere around there, that curve is where the fall off is. That essentially, you know, lower contrast up here, you can't really see the variations. Same thing with this high contrast, okay? Your eye is sensitive in different ways. So what this means is that some frequencies matter more in terms of the importance to you in an image, okay. In particular, if I were to leave out a bunch of that freq, you know, frequencies. So that is, if there was low contrast at the high frequency, you wouldn't even know, so you can use that to do image compression.

13 - Image Compression

You probably have heard of JPEG, probably have JPEG images all around you, okay. JPEG uses something called the Discrete Cosine Transform, or variations there of. Basically a way of thinking about that is they take a little 8x8 region of the image. So you carve the whole region up in little 8x8 segments. And then it takes, as a basis set, these sinusoids and co-sinusoids, right. So here we have the ones that are vertical, here we have ones that are coming down and here we have the product of them. And you can see the lower frequencies are in the top left hand corner and the higher frequencies are in the bottom right hand corner. So what you can do is you can say, how much of each of these do I need to make the picture? And you can actually order them, right. You can, you can sort of go through them and order them this way. With the idea being that the top left hand corner might be, in fact go back here. You see how this is a constant? Okay, so that's just, if you, that's just the average of the picture. So the top left hand corner, what sometimes called B(0,0) that's just the DC component, the average and then as you get further and further your getting the higher and higher frequencies. So the top left hand corner represent the lower frequencies. The bottom right hand corner represents the higher frequencies. Now, what we said before is, we don't have to represent the higher frequencies that well, right? Just sort of the high contrasty components of it. So one way of doing that is saying, what if we encode these coefficients better, more bits, than these coefficients? And that gives you what's called a quantization table. And DCT, discrete cosine transform, does the compression by, so what this 3 means is we keep the coefficient say to the nearest three values, so if you're thinking of this in bits you, you can shift it over by two. The idea is that you round this to the nearest 3. Whereas this is rounded to the nearest 31, okay. So in other words, you're doing more representation of the top left hand corner than you are of the bottom right hand corner, okay. And so what that means is you lose information but you're mostly losing the

information down here, okay. And the same way, by the way, that you do the inverse Fourier transform given the spectrum you can reconstruct the signal. The same is true of the DCT. IDCT, inverse DCT. Given those coefficients, I can reconstruct the image. But now, instead of maintaining all those coefficients exactly, I'm only going to keep more information about the low frequency coefficients than I am about the high frequency coefficients. And that works really well for the human vision system. And so if you take a look at your average JPEG image and here's an example I've stolen from the web somewhere, okay. So they used 89,000 bytes of information to represent this picture. Using just the, the raw intensities and 12,000 bytes here so this is a ratio of what, seven to one. Using DCT coefficients, the JPEG standard, all right. The reason it works, to sum this all up, is that images vary over frequency, that's our base's set. The human vision system is sensitive to different frequencies at different amounts. So by doing similar to a Fourier transform where you take that integral or that sum, that dot product with the wiggly sinusoids, DCT is a variant of that. We can figure out the coefficients for each of the frequencies. Then we can threshold essentially or reduce the level of representation of the high frequency in order to save a bunch of bits in describing the image.

14 - End

All right, so that ends totally our lessons on frequency. I'll bet you're glad that's over, I know I am. As I said when we started this, if you're doing image processing as an electrical engineer, you would be doing a ton of work analyzing images in terms of their frequency content. And you'd be thinking about image processing operations in terms of their frequency behavior. We, however, will only do this a little bit because as computer scientists we tend of think of images more as data structures than as signals. But we're still going to have to worry about things like frequency content when we try to do things like optical flow where it's necessary that the image not vary very much over some distance of approximation that I make. So you'll actually have to do a filtering that sort of removes that high frequency. So, hopefully you now have enough understanding of Fourier analysis and frequencies to be dangerous. And we'll go ahead and practice being dangerous in the lectures to come.

3A-L1 Cameras and images

1 - Intro

So, three is a new unit for us, and we're going to start talking about camera models, and it's going to start off gentle, and then get a little a little rougher later where we start to get into things like perspective projection, so just bare with us. In the middle we'll do something cool in stereo, which you're actually going to to do for a a problem set, and it's pretty straightforward. But, that's what that class will, will keep trying to do, we'll, we'll dive down into the math a little bit, come back up, try to make sure you always have the math you need for doing the problem sets. Some of the stuff we'll give you you won't use directly, on the problem sets for the assignments, or even the exam but it's important that you, that you, that you know about it. So, what is an image? Well, up until today, we've been talking about images as functions, remember, they were a, two dimensional function or a function of intensity, of two dimensions, x, y, u, v, i, j. Today, we're going to start saying wait a minute, it's not just an arbitrary function, these images are actually projections of three dimensional scenes, of three dimensional points.

2 - Heliograph

Here is an example of the first known photograph. It's one of the first known recorded projections done in 1826. You can see the device here on the left, and on the right was a reproduction that was made in the mid 50s of what the image looked like. Now, what's remarkable is this image is called a Heliograph. Helio sun because it took a lot of sunlight in order to make that picture. A pewter plate,

which is metal, was coated with something called Bitumen of Judea, which is some sort of derivative of a petroleum chemical. After a day long exposure, so like, eight hours, the plate's removed and the image is found by, you have to wash it in a solution, a mixture of oil of lavender and white petroleum. Which dissolved away the parts that had not been hardened by the light. So the idea was you have a chemical, it was a liquid that had underwent a chemical reaction from, from the light that would change it. So then you could then remove some of that and that's how you got an image. Those of you who've been fortunate enough ever to work in a dark room, it still has that principle of chemicals manipulating silver halide. Or, or, or some silver elements that are changed by light. Of course, most of us don't do that any more. We we use these fancy devices which have no soul, but which capture pixels better.

3 - Imaging System

What is a camera or an imaging system? So simple definition is, it's some device that allows the projection of light from three dimensions, three dimensional points, to some medium that will record the light pattern. That medium can be film or it can be a sensor, etc. But actually the key word to all of this is projection. So here's a projection. Some people walking on a sidewalk. And there's this globe like thing in the middle. And something that might convince you that something is a little strange might be there's a tiny little guy standing on top of the world this was a pre-clip for the movie Titanic. That was a joke. Tiny little guy. Leonardo DiCaprio's not so big. Okay anyway. What's really in that picture is that. Okay? And the reality is this picture wasn't actually the projection that you thought it was. At least some of you. So you thought maybe it was like this ball thing that you were looking at in an image. And no, actually, it was a flat thing that happened to be projecting into the camera. And that gives you an idea that when you do projection somehow you're losing information, you're losing three dimensions, a third dimension you go from three to two. And we have to figure out what went on in that third dimension

4 - Image Formation

Let's talk about how you might make an image. So, here's an idea, let's just take a piece of film, or a sensor, and, let's just expose it to the light. Do we get a reasonable picture, well, and the fact that the title says image formation, bad method should give it away. Well, this point on top of this cartoon pine tree, well it projects, a ray of light to some point on the film. But of course [COUGH] it also projects a ray of light all over the film. Likewise, another point projects all over the film. So, one of the problems is that every location on the film is seeing light from all sorts of different places, there's no single, value of light that would, that comes from a single place in the world. So how might we fix that? Well, here's an easy way. Let's put in a barrier, a barrier with a very small hole in it. So here we have this little, a hole, and it's called an aperture. Okay. And we're going to use the word aperture to mean a couple of different things, but generally aperture is a small hole, that let's light in it, you might change the size. But here we have an aperture, and what that aperture makes is that each area on the film plane or on your sensor plane is seeing light from only one, point on the object, so it is showing you the light that just came from one place in the world, which is what we're after. By the way, something to note here, and we're going to see this a lot more, is that the image over here on the film is inverted from what's on the object. So the pine tree would be upside down, if you will.

5 - Aperture

The very first known camera, that is not something film-recorded, but camera is actually, go, dates back to Aristotle. And here's an image. And it's called camera obscura which is from Latin, means darkened room. So those of you who speak any Romance language comma bed camera from, from room. Obscura, darkened room. By the way there's a cool museum in Edinburgh called Camera Obscura which is all about the history of, of cameras and photography. I highly recommend it although I didn't go there because I was in the Scotch museum. And here you see the idea of a

room with also a small hole of it, okay, that's the aperture, projecting on the white wall in the back. In fact there's a nice quote from DaVinci. I don't know how you get quotes from DaVinci, but I guess he wrote them down. Or maybe it was on Memorex tap. I don't know. It says when images of illuminated objects penetrate through a small hole in a very dark room, you see on the opposite wall, these objects in their proper form and color, reduced in size in a reversed position owing to the intersection of the rays. Okay? And that's what we were showing before, and by the way, those of you thinking about lenses and, and focal length. The focal length of that system was the depth of the room, all right? So, how does that aperture affect the image that we see? How does the size affect the image? Here's a demonstration taken from Paul Debevec's website of making pinhole cameras. And you can see on the left that he actually took a piece of foil, and he's mounted it directly on a, an electronic monitoring system. On the right is the picture that was produced using this camera, and presumably he had to worry about leaving the shutter open long enough to get the proper kind of exposure depending upon the sensitivity of the sensor. And we're not going to talk about imaging and sensitivity in this course. The first thing you might notice is boy, that's an awfully blurry picture. And the question is why so blurry? We just said that when we put a really small hole, it only lets a single ray of light in it and so it should be nice and crisp. Well the problem, of course, is that there's small and then there's small. And so the question is how big should the aperture be? So here is an actual rendering using a pinhole type camera, where they're showing you the different size of an aperture in terms of millimeters. And the ability to see a crisp image on the other side. And you'll notice when it starts out at two millimeters you get a pretty blurry picture. And another way of saying that is two millimeters is large compared to the rays that you want to try to capture. And then it goes down to one and 0.6 and finally 0.35 millimters. Gives us a pretty nice rendition. So question you might ask is well, why not make the aperture as small as possible? I mean the smaller I make it, the smaller number of rays that get through, and the crisper it will be. And the answer lies in, let's see. Somewhere in physics maybe you played with water tanks. Where you cut little slits and you pushed waves through them and you caused ripples to change. And then you learned about something called diffraction. Okay? And how waves interfere. And with light what happens is if you make those holes too small you start to get diffraction effects. So at 0.35 millimeters we've got a nice, crisp picture, but then things start to go bad. And then when it gets even smaller, not only are we getting less light in, but we're starting to get things blurred out again. And that's because of diffraction effects. So you have to worry about those kind of things if you were making pinhole cameras.

6 - Lenses

The rest of what we're about to do for this lesson is just talk a little bit more about some of the properties of, of image formation and lenses and cameras. Actually belongs more in a course called Computational Photography. And my colleague Irfan Essa at Georgia Tech teaches one. And there are others online. And it teaches you sort of the math of the, of, of photography. I'm only going to talk a little bit about it here. Because next time we'll sort of run back and revert to a very simplified model of imaging. And do a much more complicated notion on recovery of computer vision. So, the first thing about computation photography is well, we don't actually use pinhole cameras. Right? We use lenses. And lenses have this notion of focus. And in general, lenses are designed so that all the points at a particular distance away will all be focused to the same point on your film or on your recording medium. One of the challenges there is that another point that's at a different distance away, so here the blue point on the tree, they're not all going to be focused at that point. They're going to be sort of spread out a little bit. And in fact, that spread is referred to sometimes as the circle of confusion. It's a, if it's a single point there's a circle that it cuts out. So, the way you change where things focus and how much is in focus have to do with changing the shape of the lens. So, talk just a little bit about lenses. Sort of our slightly simplified view of lenses as follows. One is that lenses have an optical axis, along which light passes straight through unbent. The other is, is that light that comes in parallel are all bent to the same point. And that's referred to as the focal point. The other thing is in order to reduce the amount of spread of things, lenses also have an aperture. But now the aperture, instead of being just a single point, is restricting the set of rays that

are coming in. It used to be that almost all lenses were cut out of parts of spheres, because if we understood the mechanics of that and were easy to make, and you could do the math. But these days there are many of what are called aspherical elements because it's all computer modeling. We'll talk about that just a little bit at the very end.

7 - Thin Lens

In computational photography, it's common to make what's known as a thin lens assumption. So a thin lens assumption is basically the drawing that I just showed you before. Where we say okay, for given lens with a given focal point, it's going to image some object at let's say distance, d0, at some distance di, or, or for d for image, back. And the question is can we predict, from the focal point what the relationship is between d0 and di? Well obviously the answer is yes, or I wouldn't have been wasting all this film. Or, I waste your time. So here's a thin film lens diagram taken from the slides by a Forsyth for their textbook, and we're going to use that to derive the relationship between distance in the world. And which we're going to call here z, away from the, the center of the lens, to the distance z prime, which is where the image will be focused, and the focal length, f. First, just to be clear, here is some point p that's at some height y, since we're just going to look at that side. We're going to assume that it's projected at some point p prime, that is, y prime. Off of here. And here is our focal length, f. All right? The idea is that all parallel rays are going to intersect f. Okay? Which is why this one goes back here this way as well. All right? So when we have all sorts of parallel lines and that are going through the same point, it's easy to reason about what's going on here using similar triangles. So here we have our similar triangle. And it basically says that y is to z as y prime is to z prime. Two things to note here. First of all, I only use the magnitude of z and z prime, because we're only going to worry about the length. And I probably could have used the magnitude of y prime, but just to make it clear that the y value is inverted, I set negative y prime as being a positive value. So if y is positive, then y prime would be negative. If y is negative, then y prime would be positive and thus the negative sign. All right, so that's our first similar triangle. Our next similar triangle is here. And that says, and this one's a little bit trickier. Again, the ratio between y prime and y is the same as the ratio between z minus f, so that's this amount, right? All of z minus f, is to f. So y is to f. All right. So just look that a little bit. So y prime, size of y prime is to y as z prime minus f is to f. So that gives us a second set of similar triangles. Combining these two formulas gives us this formula. Not rocket science here, we're just setting the two left hands sides are the same, so we set the two right hand sides equal to the same. We continue, we play around with the math a little bit. And we get this very simple formula. That 1 over the magnitude of z prime plus 1 over the magnitude of z equals 1 over f. That's the thin lens equation, and any points that satisfy that equation are in focus, okay. So if my film plane, on a thin lens assumption or a thin lens model. If my film plane is z prime away from the lens for a focal length f, then everything that is z away will be in focus. So by moving the lens in and out a little bit from my film plane, I change where in the world things are in focus. So I can change the z prime easily. I can move the, the lens in and out from where my film plate is and that changes the points out in the world that are in focus. There's a cool little thin lens app. It took me four times to say that before I got it right. It's posted here on the Web and it just lets you sort of play around with how you change the geometry of what's going to happen.

8 - Focus on an Object Quiz

Let's try and apply this equation to bring an object into focus. Here's our lens, and image plane. Now say we have an object point p, that gets projected onto the image plane at point p prime. Now let's complete the pin lens diagram. We know that, rays of light from p that are parallel to the principle axis after refraction, cross the principle axis at a point that is focal length away. So that distance is f. Similarly on the other side, that is also f. The distance of the object from the lens is z. And the distance between the image plane and the lens is z prime. Let's say the focal length f is 50 millimeters. Now consider two scenarios, one where the object is one meter away, and second where the object is two meters away. At what corresponding values of z prime would the object be

in focus? Enter your answers in the boxes provided in millimeters with up to two decimal places of precision.

9 - Focus on an Object Solution

As you might have guessed, we want to apply the thin lens equation. Let's make our units consistent by converting z1 to millimeters. Now applying the formula, we know that 1 by z1 prime plus 1 by z1 is equal to 1 by focal length. Solving this for z1 prime gives us 52.63 millimeters. Similarly for z2 prime, we have 51.28.

10 - Varying Focus

One of the cool things about real lenses is as I'm looking at a scene if I want too, I can just change, a little bit the focus in and out. Now for those of us who use cameras that just take pictures, our goal is to put something in focus or out of focus et cetera, whatever. If you're like Meghan, who thinks static pictures are just like way old school and the only thing that matters is video, then the idea of changing focus during video is interesting. Generates something that's called pulling focus or a focus pull, and you get this really cool effect. And here you can see this picture of this, pretty sure that's the Water Temple in San Francisco, somebody else might remember. And there's the back end, that's of a duck, but you can kind of see that it's shot through a fence. And as I run the video, you can see how changing focus changes where, how far in the scene, how far z has to be in order for it be in focus on the image plane. And that's pretty cool.

11 - Depth of Field

One of the important things when you talk about real lenses is for the photographers among you, you know about this, for those of you who are new to it, it's cool, it's called depth of field. And depth of field is, and, looks at the question of, okay I've got some depth in space that's exactly in focus, how much does focus change as we get. A little bit off of that, of that distance, right, so if I move a little bit of the point that's precisely in focus, how much does the focus change, and what that's controlled by actually is the aperture. And the reason is, you can see here, so here we have a wide aperture, and with a wide aperture, these, these rays diverge quite a bit, if my film plane instead of being here, okay, was actually here, okay, the rays from this point would have been spread out a lot, so it would have been out of focus a lot. But on this smaller aperture, okay, where things were supposed to be in focus here, if I actually, am off by a little bit there that is that my film plane has moved a little bit, you see that the spread is much less, and that's why the depth of field is much better. That's illustrated here, these are some pictures from Wikipedia, so here's a picture of a flower. You'll notice the flower itself is beautiful and sharp, and the background is totally out of focus, now, it says f/5.6. Now. This is f for aperture, not f for focal length. I apologize, those of you who buy lenses, you'll buy a 50 millimeter lens, f/1.4, the f is about the aperture, it means the, the smallest aperture, the, the largest aperture it can get. Also, fs are inverted, the bigger the number, the tighter the value, so 5.6 is a relatively larger aperture, compared to say f/32. So here, same flower taken from the same position, but you'll notice now that the background is in sharp focus. So if you're a photographer and you, you want to separate out your foreground from your background, you want to have a wide-open aperture. If you're trying to look at a landscape and you want to see everything in focus, you want a very tight aperture. Here is a very nice example of depth of field, the focal plane obviously is somewhere in the middle of the phone, so the stuff that's in front and the stuff that's in back is out of focus.

12 - Field of View

Another property of our imaging systems is field of view, sometimes called zoom, which is probably not quite right. But it's this idea of how wide of a view do we have. And I'm going to

show you this in a lens that can change its focal length to show you that the field of view is a function of the focal length. There are lenses that can change their field of view while staying in constant focus are called zoom lenses. There are some lenses that you can change their focal length but you have to refocus. Those are actually called variable focal length lenses. That's the, just the nomenclature. Here we start off with a very wide angle shot, it says 17 millimeters. And you see this little circle up there that's, that's on that boat? As we step through and we increase from 17 millimeter to 28, to 50, to 85, what we're doing is we're increasing the focal length of the lens. Here we are at 135, 300, 500, and that's 1,000 millimeters. So we've gone from a focal length of 17 millimeters to 1,000 millimeters, it's a ratio of greater than 50 to 1, and there's that guy on the boat, right. Same boat back here, all right. There he is. Okay, so that's incredible zooming. So you think they used a tripod? Well of course they did, right? In fact, this is something you might think about if you haven't thought, right? Let's assume for a moment that your hands shake just a little and they cause a small amount of angular deflection. Well if my image is looking at something this wide, then a small amount of angular deflection doesn't move my pixels very much. But suppose my angle of view is, you know, a degree or just a couple degrees. Well, then a small amount of of angular shift of my hands is going to move the pixels a lot. And that's why the longer the focal length, the more you have to stabilize your image. And that's why for really nice zoomed out lenses you have to have a tripod. It's not just because they're heavy. It's because you need to have things be perfectly still. Now we can compute the field of view. And the field of view actually depends upon two things. One we already know, right? We know that it depends upon the focal length. But it's also going to be a function of, sort of, how big is the light sensitive element? So here calling d the size of the retina, okay? So the retina is the thing in the back of your eye that senses light. Just think of it as the light sensitive medium. So, so d is this whole size, okay. Here's f r focal length, and clearly sort of half of d over f, well, the angle who's the arctan of that, the angle that that's the tangent, that's going to be this angle phi here, which is half the field of view. So basically, the bottom line is, the longer the focal length, the smaller the field of view. The bigger the, the imaging surface, the bigger the, the field of view. In fact what's sometimes called medium format photography, instead of using tiny little sensors, they use much bigger sensors so they can put, still a lot of pixels, but over a much bigger area, that gives it much wider fields of view.

13 - Framed Quiz

Let's say you are an art gallery. You've been staring at a painting that you really like, and now you want to take a picture of it. Your camera has a lens, and the sensor inside. Now when taking the picture, you obviously want the painting to be in focus. But say you also wanted frames such that, the picture takes a exactly the image area, that is, you want to get the image, when projected unto the sensor to match the sensor width. Let's assume that your camera lens has a fixed focal length of 15 millimeters, your sensor width, denoted by d is 35 millimeters, and the width of the painting, w is 0.7 meters. Now with your knowledge about field of view, can you figure out how far the lens needs to be from the painting, and how far the image plane needs to be from the lens to capture the desired image in focus. Be mindful of units when calculating your answers.

14 - Framed Solution

In order to solve this problem, first notice that we have two unknowns, z and z prime. What does that indicate? Well, we probably need two equations to solve. One is our by now familiar thin lens equation. That is 1 over z plus 1 over z prime equals 1 over f. Let us simplify this so we can express one variable in terms of the other. Adding the two terms on the left hand side, and then cross multiplying. Now say we want all the zs on one side, take z common, and finally we can write zs, z prime f over z prime minus f. Remember this last result. Now, we need to think about a second equation. Look at the diagram again. See anything useful? How about these two similar triangles. They tell us that half of d over z prime is equal to half of w over z. Simplifying this, we can write that is z equals w over d times z prime. This is our second result, remember we found earlier that z equals z prime f over z prime minus f, and now we saw that z equals, w over d times z

prime. Equating these two and solving further, we have z prime equals d f by w plus f. Now putting the value of d, f, and w here, you should get z prime equals 52.5 millimeters, putting that here, should give you z equals one point o five1.05 meters

15 - Zooming and Moving are not the Same

So, here's something cool that's related to the mathematics of photography which we'll do more of. And it's that zooming and moving are not the same thing, right. So some people think oh if I zoom in, it's just like getting closer to something. Well, it's not true. Here are two pictures of the same car. Okay? And you can tell already that they look somehow different even though it's the same car, and the car is in the same place. And the reason that they're different, is that on the left, the car was taken with a very large field of view. So that's a very small focal length. So that means the photographer had to get close to the car. Okay. The image on the right was taken with a small field of view, a larger focal length, so the photographer had to back away from the car. So even though that the car is the same size in the image, you'll notice right away, that the shape seems quite different, all right? The one on the left seems to have what, a lot of what's called perspective distortion, and it's not really distortion at all. It's just the effect of perspective. And that's why I said, we're going to do the mathematics of that, really not today. We'll, we're going to do that in the next one. This has dramatic effect on taking pictures of people as well. So here is a example again taken from the, the web, the website in there says StevenEastwood.com where it has tutorials on lens perspective. What we have here is pictures of the same woman. But over here, it's a 350 millimeter lens. Standing very far away zoomed in, here is a 19 millimeter lens standing much closer in order to be able to get the same size face. And you'll notice that you get this incredible apparent change of the shape of her face. And over here that's referred to, again, as perspective distortion. But that's not really a correct word. It's just the effect of perspective. This is why photographers use particular lenses for portrait work. And they tend to pick a lens that lets them stand, what they call a normal distance away. It turns out when you and I think about what a picture of somebody should look like, we tend to think of what it looks like standing somewhere around six, seven, eight feet away. Or maybe a little bit further, but certainly not closer. And that's why they pick the particular lenses that they do. Here's another example of what's called the effect of focal length on perspective. And here it's being called correct perspective effect. And this is a image out of the Zisserman and Hartley book. They're actually talking about the effect of perspective and weak perspective. We'll talk about that actually next lecture. But here two pictures of the same building. Again one taken with a much wider field of view lens. So the one on the left, so you see this effect as the object gets smaller away. And with a telephoto lens, where the image on the right, you don't see any converging of the lines. And we'll talk about the mathematics of that next time.

16 - Dolly Zoom

There's a really cool thing you can do about this, and it's something called a dolly zoom. And this was an effect that I think, at least I've been told, was invented by Alfred Hitchcock. Basically, if you move closer and closer to something, but you widen out the lens at the same time, you'll cause the object and subject in the middle to stay about the same size and to be stationary. But it's going to be very weird, because what's going to happen is that all the stuff from the outside is going to look like it grows. [MUSIC] And in the slides and study guide on YouTube is a link to the original. And there's also another link put out by somebody who wanted to just demonstrate this effect. And they did it for even a longer time, and you can get to see that. [SOUND]

17 - Lenses Are Not Perfect

All right. Our last bit is, is, is, I have, I have to break it to you that reality can be a problem. That's a nicer way of saying life sucks. In reality lenses aren't really thin, lenses aren't perfect, photographers aren't perfect, except some of us. Sensing arrays aren't perfect so sometimes we have to do stuff to try to fix these things. So, here's an example of a problem that's fixable, easily, in mathematics and

geometry. And this is referred to as geometric distortion. So on the left here, we have a nice, regular grid. But sometimes, our lenses are not perfect, and they'll bow things out like this, and this is called pin cushion distortion. And sometimes they'll bow things out like that, and that's referred to as barrel distortion. By the way, for those of you that use Photoshop or other more sophisticated image manipulation tools. You can correct for this in the image, and in fact, in Photoshop and, and in Camera Raw, if it knows the camera and lens that you're using, you tell it, correct for the geometric aberrations and it will go ahead and just fix that for you. because it knows about the lens and it knows about the camera. By the way, I used the word, aberration. That's what we tend to use for any sorts of problems that are, are in how the lenses, how the images are created. Here's an example of the correcting of that radial distortions. Radial coming out from the middle. Picture on the left taken with a lens that's actually designed to try to capture as much of the scene as possible, but it's very distorted. On the right, you can see the recovery making it as if everything were perfectly flat. Another kind of aberration or problem in computational photography is what's called chromatic aberration. Oh sure, if any of you are Pink Floyd fans, you're familiar with the picture that shows white light coming through a prism spreading out and making a rainbow. I would it to you, but I'm sure I'd have to pay Pink Floyd money. All right. And the problem is, is that different color lights are diffracted different amounts by a particular lens, or a particular medium. So the problem is, how do I get all the light coming from a point to land on the same place in my image? And this is shown here where they're showing you the, the red and blue lights as converging to different places. And, here's an example at a corner, and when you blow it up, you can see this red line that's between the, the blue sky and the, the white building. Where did that red line come from, there's no red there. Well, that's chromatic aberration. By the way, Photoshop knows how to take care of that too. That one's a little bit more of an approximation, but you can remove the chromatic aberration of your lenses that way. Last one we'll talk about is what's called vignetting. If you use certain cameras at certain aperture settings. Not all of the rays that are hitting from, say, the middle of the image, coming through the middle are being caught at the corners. So here we have a, a lens imaging system and we're going to talk about that in just a minute, and we've got some point. And the ones that would be near the edges goes, goes through a couple different lenses. Only some of them hit the imaging medium, and so what that causes is that you get these dark areas around the, the edges. And that's called vignetting.

18 - Lens Systems

The question is, what do you do about these effects? And the answer is you spend money. What I'm showing you here, and I'm assuming I can show this to you because you can just go and get it, and it comes directly from the literature, is a beautiful lens, a Nikon 24-70 zoom. Now you know that I don't own Canon equipment. Well, actually I own some Canon. I own some equipment of every brand out there. Well, any brand that has a lawyer. I own some of your equipment, okay. Here's showing you the elements, that are in this zoom lens. And you'll notice it's not just one thin lens that's moved in and out. It's made up of 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 pieces of glass that are moved in different ways depending upon whether you're changing the zoom length or you're changing the, the distance in focus. These days it's a, it's a combination of computer design, plus knowledge on the parts of optic engineers about how to do this. And what this does is, it helps remove vign, reduce vignetting, chromatic aberration, geometric distortion and other kinds of perturbations to a perfect pinhole image that we like to see.

19 - End

So that ends this lesson. Gives you a little bit of discussion of about how images are formed. This is reality. But in general lenses are designed to make it look like as if you had a perfect pinhole that happened to give you really bright pictures that were very clear. That's really good for us because the mathematics of a pinhole camera are very well understood and are very clean. They're a little bit, shall we say, complicated than simple algebra. But they're, they're very easy to model. So, when we do the mathematics of how images relate to the geometry of the world, we'll use the

pinhole model. But everything that I've showed you here is how we build lenses that give us this illusion of having a very powerful pinhole imaging system. See you next time.

3A-L2 Perspective imaging

1 - Intro

All right, welcome to Computer Vision. We're going to talk today about perspective imaging. Before we talked about sort of different geometry and configuration of cameras. And today what we're going to do is we're going to do the math behind that. In particular, even though we talked about all these different problems and different ways lenses work, etc, today we're going to sort of retreat to the pinhole model, sort of take the academic route of take a simplified model. So we are not going to talk about things like focus or other non-pinhole effects. Remember pin-hole camera, all the rays are in focus. At some point we might talk about fixing lens distortion, which is something you don't get in a real pin-hole camera. But the reason we're doing that is to try to make our images be more like they were generated from just some really uber pin-hole camera.

2 - Coordinate System

So fundamental to our notion of imaging is our projection operation. And today we're going to talk about modelling projection. So first we're going to talk a little bit about the coordinate system. What we'll see here is the coordinate system we're using for our pinhole a, approximation. The first thing is that we're going to put the optical center of projection at the origin. So here's our little COP, center of projection. And that's also going to be the origin of our camera coordinate system. Also with our center of projection at the origin, we're going to use a standard x y coordinate system for the stuff we're going to show here. So now we're back in the land of mathematics. So x and y are like regular xyz, so x will go to the right and y is going to go up. Later when you convert this to images you're going to have to worry about making that flip. Now, here's something a little strange. We're going to, as we were talking about before in cameras, normally a light comes in, goes through the focal point of the lens and then hits the image plane. But in our mathematics here, what we're going to do is we're going to put the image plane here, in front of the coordinate system. As if the light sort of magically knows how to pass through the image and hit the coordinate system in just the right place. So why do we do this? Well, because it's mathematically convenient, because this way our images don't get inverted, y stays up, all right? Before we had to worry about things being switched left and right and up and down. Well now, x stays to the right, y stays up. And we don't really build a camera with the image plane in the front. We put the cam, the image plane behind, and then we have to rotate things back to get y up and x to the right. So for the math here, we're going to put the image plane on the right. Now, one of the implications of this x y coordinate system, in order for this to remain a right-handed coordinate system, so you remember x, y, that this way is z. What it means is that z is now pointing in from the camera, not out to the world. So, here, here's z pointing this way, all right? Sometimes this matters, sometimes this doesn't. You see here we wrote down that the world's position of some point was x, y, and z. And we didn't say negative z. Well that's because it's a z value, which could be negative 20. And we can go back and forth on this. But basically, in order to keep a right-handed coordinate system and everything sort of not have to worry too much about how we do the matrix multiplication and stuff like that. We have that z goes negative the further away that you get. So what's going on here? So what we have here, we have our world point here. All right, so here's our point in the world, and the ray projects down, to the center of projection. Normally our image plane would be back here, but we're going to say. So, the idea is that, that ray intersects the image plane at this point, which we're going to call x prime, y prime. And it's written as minus d because the idea is that the distance, like the focal length distance, the distance d from the origin to the image plane, that's a positive distance. So what that means is that the location of this point is going to be x prime, y prime, minus d.

3 - Modeling Projection

Using this projection from the rule point to that image plane, we get projection equations. And we can compute the intersection, that is where x prime y prime is by looking at similar triangles, kind of just like we did for the thin lens. So basically this triangle and the projection triangle here, these are similar triangles. And by the similar triangles, basically get that the x, y, z location, the location of the point, is now mapped to this point here which, by similar triangles, is minus dX over z, minus dY over z, and minus d. And here you can see that negative d issue, alright. So basically what it says is the x is divided by z and multiplied by d, by whatever that, essentially, focal length scale factor's going to be. So, using that similar triangle and knowing what the location on that point on the image plane is, well the question is, what is the location in the image of that point? Well, to get the location of the image, we just throw out the last coordinate and basically it says. Not basically, exactly, it says that x prime, y prime, which is the x, y location in the image, is just minus d, X over Z minus d, Y over Z. Normally in class I'd say, so any questions, but Megan understands everything totally, and you guys are all brilliant, so I'm not going to worry about it. By the way, one of the obvious implications of this dividing by Z is what? Well, it's that objects further away are smaller in the image which is something we all know anyway, right? Take an object, pull it further away, and its size and the image gets smaller. And that's because you're doing perspective projection and the points are mapped to the values that are proportional, X over Z and Y over Z, as Z gets bigger, X and Y get smaller with respect to 0, which we're putting at the center of the image. Do you hear that? 0 is at the center of the image under this. So not at the top left hand corner. So you may have to worry about that later too.

4 - Distance Quiz

Time for a little quiz. So when objects are really very, very far away. The real X, that is how far off from, from the center they could be, might be huge, as the real Z. Could be miles. So if I move the camera, the origin, those numbers hardly change, right. The location of the real x out in the world, the real layout in the world with respect to this point, and it's miles away. If I move this a few inches, nothing actually changes very much. So this explains which of the following? A, why the moon follows you. I hope the moon does follow you. It follows me, but maybe I'm special. Why the North Star is always North, that's b. C, why you can tell the time from the sun regardless of where you are. And d, all of the above.

5 - Distance Solution

Well the answer here is clearly d, and the reason is, is that basically when I move my camera, since essentially cap X, capital X, capital Z don't change, all right? Where the image, where the point of the moon lands on my, on my image plane, it doesn't change either. Now it'll change if I rotate, of course, but we're just talking about translating. Now, pretty soon I'll translate right out of the field of view and Megan's going to get unhappy because I'm going to come back here. But if you've ever been driving along and you see the moon and it just sort of follows you, okay? That's because the X and Z, well the Z's not changing at all if I'm going this way. And the X is changing maybe by miles, okay? But the moon is, what is it? It's like 200, I don't know. It's really, really far away. Somebody look up the, the, I think it's a quarter million miles, something like that. Anyway, a lot's, lot, a lot shorter than the distance between here and Roswell. So, as I go all the way from here to Roswell, the X hardly changes, the Z hardly changes, so where it appears on my image doesn't change. And basically, this is why things that are very far away don't move as, as I move. The only thing that matters are the angles, so this is true for the sun or the North Star or these kinds of things. And that's, and, and it's because of this perspective projection.

6 - Homogeneous Coordinates

Now, we get to the ugly stuff. Homogeneous coordinates. It's not ugly, it's beautiful, for some twisted notion of aesthetic. So, that operation where we take x over z and y over z. Is that a linear transformation? Well, the answer is no. Now, before we said that well, division's a linear operation. Sure. Division by some constant by a, 1 over a, multiplying 1 over a, that's a linear operation. But, here and I take x, y, and z, okay? I actually have to pull out that z and divide the x and the y by it. And so, if I've got another point with x1, y1, z1, I divide by that z. So, it's not a linear operation. I have to essentially change what's going on. So what we're going to do is we're going to do a trick. And the trick is, we're going to add one more coordinate, all right? And we can do this in either two dimensions or three dimensions. In two dimensions x,y becomes xy1. X,y,z becomes xyz1. And this bottom coordinate, which right now here is written as a 1, later we're going to, they'll, it'll take other values, that's the homogeneous edition, the homogeneous coordinate. Sometimes it's called the scale factor but I, but I think that's inappropriate, it should just be viewed as the homogeneous coordinate. Or the homos, homogeneous component of the vector. So, converting from regular coordinates to homogenous and back again is pretty easy, right? If I have the homogeneous coordinate x,y,w I get the nonhomogeneous by just dividing by w. So 2dx over w and 3d x, y, and z, each over w. So now you see why converting xy1 to nonhomogeneous I still get back the x and y as I had before. But if that last homogenous coordinate is not 1, then I'll get a different value. By the way, this makes homogeneous coordinates invariant under scale. If I were to multiply the entire homogeneous coordinate by some constant, say a. So, I have ax, ay, aw. When I do the conversion back again doing the division, the a just cancels out.

7 - Perspective Projection

So the really cool thing about doing this is that now we can take perspective projection and make it a linear matrix operation while we're working in homogenous coordinates. And we don't have to worry about the nonhomogeneous part until the very end. So that looks like this. So here I have a three by four matrix, and you notice it's 1, 1, 1 over f. And I multiply that by my homogenous 3D coordinate, xyz1, and I get out this three dimensional homogeneous coordinate, right. So it's 2D homogeneous coordinate 3, with component x, y, and z over f. Okay? And I can just keep this around as a homogeneous coordinate until I actually need to use it in an image. And when I use it as an image, I do the division of the top element by the homogeneous element, and I would get f times x over z and f times y over z. Which by the way is sometimes called u, v. And those are the coordinates in the image of some point x, y, z out in the world projected through a projection with a focal length of f. And so our projection operator was this linear matrix multiplication, and as we know, we all love linear matrix multiplications, because we know how to do them a lot. And invert them and all other kinds of stuff. And it's really the basis behind projective geometry, and the projection mathematics that we'll be using for a bunch of the course actually. By the way, f, from now on, is for focal length. Before we're talking about that distance D, so here our focal length is the distance from the center of projection to the image plane, and it is not the f of the aperture that we were talking about last time. I probably shouldn't have even reminded you, because it's probably months ago since you watched that lecture. But f here is always going to be focal length, which is the distance from the center projection to the image plane. So how does scaling the projection matrix change the transformation? Well, here what I've done is I've scaled my projection matrix by f. I could have scaled it by something else, but I'll, I'll scale it by f. So now, instead of having along that diagonal of 1, 1, and 1 over f, instead, I just multiply by f and I have f, f, and 1. Multiply that by my three dimensional homogenous coordinated vector which has four dimensions to it. Now I get out fx, fy, and z. And then when I do that division, what do I get, the same thing I got before. Okay? So it was invariant.

8 - Project a Point Quiz

Try writing some code to project a point from 3D to 2D. Make sure you use the matrix operation just described. Let's say you're given the coordinates of a point p and a focal length f. Write a function that returns the location of the projected point on the 2D image plane. Here's what your function should look like. You can test your code by passing in a value for p and f. Let's say the units are in millimeters. You can display the result using disp. Try out different coordinates and focal lengths. Note that we will test your function against the known set of inputs when you hit submit.

9 - Project a Point Solution

So, the first thing to do is define your projection matrix. Note that it has a size of 3 cross 4. That is three rows, four columns. Next, we convert p to homogeneous coordinates and transpose it. The resulting size should be 4 cross 1. That is a column vector with four rows. It is easy to append an element to any vector. A single quote after a vector transposes it. Finally, we can apply the projection transformation, and this ends up being a simple matrix multiplication. Note that we use star, not dot star, because we are not doing element-wise multiplication. Remember to convert back to non-homogeneous coordinates when returning the value. u equals x by z, and v equals y by z. And that's it. Here's a quick check to see if the function's return values make sense. Say we keep the focal length at 50 millimeters. Then we know that any point at 50 millimeters should be projected at the same x, y coordinates. Let's try this out. This should be true regardless of what the coordinates are. Let's try something else. Similarly, points that double the focal length should be reduced to half the size. See, perspective projection is not too bad.

10 - Geometric Properties of Projection

All right, let's just talk quickly about a couple of the geometric properties of perspective projection. What this figure is trying to show is a few things. The first of all is that points go to points. So that seems pretty clear. I have some point out here. And I'm going to put that point sort of out on my ground plane, right? So I'm, here's my center of projection O. I'm looking at some point on the plane. And here's that point in the image, right? Because it's just the single ray that are, that moves through. Something else that's true in perspective projection is that lines, so here's a line here, project. So one way to think about it is here's my origin, think about it going through here, and, and coming through there, and it sweeps out this whole plane. And that plane, intersects this, the plane made by that line on the ground and the point, the center projection intersects the image plane, along this line. So what that's saying is, not only do points go to points, but lines go to lines. And if lines are going to go to lines, and points go to points, then polygons go to what? They go to Mercury. No polygons go to polygons. All right? Not rocket science but it is this invariance of lines in prospective projection. Lines mapping to lines that we leverage quite a bit.

11 - Parallel Lines

A cool thing in perspective projection and something that, that you already know is that parallel lines in the world almost always meet at a point in the image. So here's an example of that's two parallel lines out on the ground plane and they project to two lines in the image. And you can see that they meet at this point, sometimes called the "Vanishing" point in the image. And the, the canonical example of that is always railroad tracks, where the, the tracks themselves all converge to some point. So we can sort of intuitively think about that. But what's really cool is you can see that directly from the mathematics of it. So let's think about what a line in 3-space is. So a line in 3-space can par, parametrized by a single value a single parameter. So in this case, we're going to use t. So we have x of t, y of t and z of t and the idea is as t changes from maybe minus infinity to infinity or zero to however you want to do it, you would move along that line. And a line in 3-

space starts at some point in 3-space. I'm saying here, x0, y0, z0. And then its moved along some vector. So right now, we're using the vector a, b, c. So a, a in the x direction, b in the y direction and c in the z direction, scaled by the value of t. So now, let's take a look at where all the points on the line lie in the image plan. All right? So that's for every x, y, z, what's the x prime and y prime for the given t? Well, we learned that the equation of the projection from x, y, z to x prime is just fx over z and for y, it's just fy over z. So here, we just substitute it in the expressions from the other side in terms of t. So that's where all the points on the line go, but what happens as t gets really big? In the limit as t goes to infinity and I'm going to assume for now that c is not zero, we'll talk about that in a minute. Okay? The limit of x prime is just fa over c. The limit of y prime is just fb over c. What's missing? What's missing is x0, y0, z0? In other words, it doesn't matter which point in the world this line starts from. I can start from here. I can start from here. I can start from here. As long as the points are all moving in the direction to a, b, c, they're all going to converge at this point in the image. And that's what were, that's why it's the case that all parallel lines in the world, so it's lines that are parallel in the world will all converge to a single point. So the railroad tracks and a sense along the railroad track, since they're all parallel in the same direction, they all converge to the same point. So what does it mean that c is zero? So c is zero means that the z value doesn't change. So assuming I'm holding my camera like this, so my image plane is a, a vertical plane, so that z is perpendicular. If the z value doesn't change, it means the line in the world is parallel to my image plane. Doesn't get further away or closer, it stays parallel and those lines will all stay parallel lines. And that's why I said that almost all pairs of parallel lines in the world converge to a point. If you have parallel lines that are vertical or aligned with the image plane, they don't converge.

12 - Vanishing Points

These points are called vanishing points. Here's it demonstrated that each set of parallel lines, that means they're all in the same direction, meet at the different point. Another thing that's true is that parallel lines that are all on the same plane, they all converge to colinear Vanishing points. And you know this as the horizon. So here is one set of parallel lines in the world. And they converge to a point here. This is another set of parallel lines. And they converge to a point there. If I had parallel lines going away from the camera, they would converge like this, all right? And the horizon is this line right there. And that's because all of those parallel lines that I drew, they were all in ground plane, so they all converge at vanishing points on the horizon. By the way it turns out getting the vanishing points consistent, when you're putting together an image. Like, you're taking a piece of a building here, and putting something else together there, as opposed to actually taking a picture, is actually kind of hard. So if you want to try to find a fake image. That is, something that's been Photoshopped, put together. Go to your local supermarket tabloid. You can often find pictures where the vanishing points don't seem to do the right thing, and that's because it was made up of parts of an image that weren't taken with a camera aligned with those parallel lines all in the same way. A little bit more, maybe some of you actually took an art class, before you decided to lose your soul and become an engineer. And maybe they taught you about 3-point perspective. Well and here's a drawing of 3-point perspective. And basically what they're talking about is if I've got a cube in the, in the, in the air. That defines three sets of parallel lines. Like the left face, the right face and then underneath. And you could see that drawn here and that's where it comes from, is that parallel lines converge to different Vanishing Points. Here's another example I think, I think this is a Zisserman product, I'm not sure, so just a really quaint old building. And there's Vanishing point from the right, from the power lines there, another Vanishing point on the left. And without talking about others, you can see where the horizon is, okay, right across there, that's where all the lines for example, on the ground would converge.

13 - Human Vision

We're very sensitive to this structure of parallel lines and what they convey to us. Now, most of you have seen this Muller-Lyer illusion, right, of which one of these lines is longer? And you probably all know that well, actually, they're the same length, and you probably saw that back in grade

school and it's kind of cool. Let me show you another version of it that shows you why this can be an interesting effect. Okay, so here we have a picture, okay? And you can see the red arrow in here and the same red arrow in there. And let me take that away. And now the question is, which of those is longer? And that one's a lot harder to, for you to say that, you know what? They're actually the same length. So you go back there. In fact, when you pull them away, when I stare at them, they look like they're different length. But of course they're actually length. And that's because you're using this perspective projection as a way of saying that in the real world, okay, the distance from here to here is more than the distance from here to there actually. All right? That is, this is the height of the whole wall. This is just the height of the ticket window. So, it looks the same on your image, but your brain automatically wants to undo that projection transformation.

14 - Parallel Lines Quiz

All right, little quiz. What determines at what, what point or which point in the image parallel lines intersect? A, The direction of the lines in the world. B, Whether the world lines are on the ground plane. C, The orientation of the camera. And d), (a) and (c).

15 - Parallel Lines Solution

So this one is probably bugging you just a little bit, so lets just go through them, all right. Let's talk about a. All right. Well, of course, a has got to be true. The direction of the lines in the world, of course the talk we just showed, the maps shows that, you know, where they intersect in the image comes from there. And then it says whether the world lines are on the ground plane. Well, all world lines that are on the ground plane, all intersect, parallel lines, they all intersect at what? They intersect at that horizon. But, now here's where it was a trick question. I said, what determines at what point, in the image parallel lines intersect? I didn't say what line, I said at what point, so that's why b is not true. Now c is the orientation of the camera, so you can think about this two ways. One, everybody knows that where a point lands on your image has something to do with where you point the camera. Watch Meghan's nose move. Okay. Yeah, as I move the camera of course things move. The right way of thinking about this is remember the a, b, c which is the direction of the the lines in the world. But that's in some coordinate system, and that coordinate system is defined by our camera's center of projection. That's the x y z center of our universe there. And we're defining all of our points in that coordinate system. And that's why it's a and c.

16 - Other Models

All right, so to finish up the lesson. Besides prospective projection, there are other types of projection operators which in fact are sort of special cases and they're used sometimes. So they're, they're worth mentioning here. One interesting form of special case of perspective projection is called orthographic. And as it's drawn here, basically orthographic is when I take my world and I just smash it right into my image plane. And what's interesting about that is that it actually can be thought of as a special case of perspective projection. But where the distance from the center of projection to the image plane is infinite. And my object is infinite, right? If those both were sort of infinite then basically what I end up doing is the z value doesn't matter because those rays are parallel. So that's why this is also called parallel projection and so xyz just gets moved to xy. This is actually some times used as a model for telephoto lens. Because telephoto lenses are looking at things very, very far away. So all of your rays are almost exactly parallel. Now of course, it has to be scaled so you end up scaling the x and scaling the y. But it's the same scale value for your whole world. I mention that, because we're going to talk about something called weak perspective, in a minute. So basically, when you throw away the z value and then you just scale the x and y by some number, that's this parallel projection. And since it can be, since I said it was a special case of perspective projection, there must be a matrix operation that would recover it. And here it is. And you'll notice, now, that this value is zero, okay? And this value is one. So x, y, z one, the matrix just equals x, y one. And when you divide out by the one, you just get the x, y. So even this

orthographic projection can be represented as the same matrix, as a matrix multiplication when we use homogenous coordinates. Another model is called weak perspective. Sometimes called scaled orthographic but I, that's a bad choice. There's also something called power perspective but what we're going to call weak perspective is following. The idea is that we know that when things get further away, they're going to get smaller. But what we're going to do is, we're going to say that all the points on a given object, we're going to assume they're all at, at some constant depth. We'll say z0 as it's written here. Okay? So for all the points on that object, xyz will get mapped to, fx over z0, fy over z0. So that object will get scaled. But some other object, might be a significantly closer object, it will get scaled by a different value. So basically in weak perspective, what we're saying is that the difference of depth over an object or over the range of an object, is very small compared to the difference in depth from the object to the center of projection. And if that's true, then you can basically say that each object has its own scale factor. Of course, as a special case of perspective projection, we can do this in a matrix form also. So, here we have the equation or, or the transformation we said before, x, y, z goes to fx over z0, fy over z0. So basically, f over z0 is acting like a scale factor, okay? So here we have our projection matrix, which now, instead of having a one over here, has a one over s. And then when I multiply that through on the homogenous coordinates and then divide, I'm dividing by the 1 over s, which gives me sx, sy. So weak perspective gives you this scaled effect of each collection of points that sit on a given object. So those are the three camera projections that we'll talk about. The one that we'll use all the time is perspective, which ends up giving you fx over z, fy over z. And we showed you how to do it in matrix coordinates using homogeneous coordinate system. And then we also talked about weak perspective where you have a scale factor, sort of, per object. Or orthographic, where you basically just throw away the z value.

17 - End

All right, so that ends this lesson on, projection. We're going to use the perspective projection a lot, so get to know and love it. And by the way, way more math is coming, so fasten your seatbelts.

18 - Fun with Perspective

Anamorphic images are ones that require the viewer to look from a particular viewpoint to show the desired visual. Other perspectives usually yield a stretched or distorted result. Artists use their knowledge about perspective projection to create some amazing illusions, like this Lego army designed by Leon Keer. Viewed from any other angle, it looks strange. This art activity popularized by Myrna Hoffman uses a curved mirror to undistort a flat drawing. Can you find other cunning examples of people hacking perspective projection? Share with your peers in the discussion forum. I encourage you to create your own, it's a lot of fun. Check out this excellent instructable from Mr. Grease Tattoo for some inspiration.

3B-L1 Stereo geometry

1 - Intro

Hello, welcome back to computer vision. Today, we're going to start talking about, stereo and this particular lecture is on stereo geometry. In general, the next few lectures are going to be on the relationship between sort of camera geometry and scene geometry. We're starting with stereo because frankly you guys are going to do this, in a problem set and I want to be able to get you there quickly, and also in some sense it is the conceptually easiest way to get started. So stereo is really just a special case of having multiple views of an image, two particular views, and here's just an example of two views of a vase. In fact, we're going to get to this particular figure, oh, a little bit later when we talk about epipolar geometry. But the idea is that there's this relationship between two views, and you can use it to recover the depth that's there. We'll also take a look at sort of

having arbitrary different views, and this is an image that comes from work on using special features to align objects and be able to detect that they're there, but, again, you have two different views and you have to figure out how they relate. This is an example of what you can get from stereos, so the picture on the left is one half of a stereo pair, on the right is a crude depth map, the brighter things are, the closer they are, in fact, we're going to talk about how to compute exactly that. And in general, you can have lots of views, where you have images taken from a whole bunch of places, or a whole bunch of directions, and it's a question of how do these images relate. And this notion of how images relate and how do they relate to the scene, that's what we're going to be focusing on.

2 - Why Multiple Views

So the first question to ask is, well, why multiple views? Obviously, structure and depth are fundamentally inherently ambiguous from a single view. Remember, we're projecting from 3D to 2D? When we do that, we lose information. So here's an example of a man pushing the Leaning Tower of Pisa over to make it lean even further. And either he's very large, the building's very small, or there's an ambiguity when you project from 3D to 2D. Or how about, so those of us who grew up in the States who were a fan of, It's a Great Pumpkin, Charlie Brown. So that might be a picture that looks like this. Which would be really a great pumpkin. But of course, what's really going on here is that the location of the camera makes it such that this picture just appears like the pumpkin is very large. And what's fundamental problem? Well, the fundamental problem comes for, from perspective projection. Here we have an optical center. All right. And we have several points such as P1 and P2 projecting along the same ray. And of course, because they project along the same ray, they land in the same spot P1 prime P2 prime. And so that's where the ambiguity comes from. And the question is how to figure out where P1 and P2 really are. And the trick, of course, is going to be the look at a different camera.

3 - Depth and Shape Cues Quiz

When you look at an image, what properties indicate differences in depth, or provide hints about an object's shape? Think about various features that convey structural information in a scene. Stereo is not the only thing at work. After all, you are seeing these images on a flat display. Type in depth and shape cues you found as comma separated list. Feel free to look at other images or just around yourself for ideas.

4 - Depth and Shape Cues Solution

Some depth and shape cues I could think of are: shadows, texture, focus or blur, parallel or converging lines, lighting, shading, objects occluding each other, and size and scale of objects.

5 - How do Humans see in 3D

So before we do think about how we are going to do this in machine, what about humans? How do humans see in 3D, all right? And frankly, we can actually do this using just one eye before we even get to two eyes because we in it, we look at these different cues. So here's an example. So, when you look at this picture, in fact, when you look at any flat picture on a piece of paper or on a single screen, there is only flat real depth, and you see the depth that was in the scene. So, here we have perspective effect. You assume that those houses are all the same size, and their fading off into the distance is what makes them smaller. So you perceive depth from this perspective effect. Shading, here's a picture on the left and b and c are some of the recovery of using an automatic method to try to recover the depth. And it's using just the shading information. You assume that the skin has a particular type of reflectance and your brain figures out the, the depth in it. By the way, this is why certain kinds of makeup is used, right? By changing the shading on the cheeks, you change the

apparent shape of the face. Even though, of course, you didn't change the face, you just changed the reflection. Texture, here you see a wall and you can tell that the wall is sort of curving around. And that's because the texture elements on it are changing. Those circles with the little lines on them, that's an early attempt at being able to recover orientation from texture. A more recent piece of work done by Loh, here you have a picture. You see the strawberry and you see the texture. And the texture changes as the surface tips away from you. And the system actually works on trying to recover the, the surface height or the orientation from the change in texture. And here's a really cool one. Remember, we talked about that in real lenses there's a single focal plane of depth. That is, there is some plane out there in the world that's a depth, that, that's in, that's in focus on the image plane. And, the more you're away from that particular location, the more out of focus you become. So you might be able to take how an image changes focus as you change the aperture. Remember, a very tiny aperture, everything's in focus. And as you open up the aperture, the depth of field shrinks. And that means that things that are further away from that focus plane get fuzzier faster. So you could actually take multiple images from that same point, change the, the aperture and figure that out. So it's a little bit of a cheat. because yes, it's one eye, but it's multiple pictures, all right? Here's an image of them recovering a 3D to shape and depth from a de-focus. And a last one, and this is something we're actually going to look at a little bit later. And in fact, it's actually related to stereo, is the idea that from a single eye, if something moves, you can tell the shape. Now, here's a picture of a statue. And it's pretty hard for you to figure out the motion that you would use. Mostly, you're looking at the shading. So here's a simpler example, all right? Now, if all our video gods are working right and the web is working correctly, and you've got decent bandwidth, and the moon is in the sun with the seventh house of Ju, whatever that is. Anyway, you should all be seeing a rotating cylinder. Now, some of you are going to be seeing it rotating counter-clockwise and some are going to be seeing it rotating clockwise. And that's because you can't be sure whether the pick, the, the dots moving to the right, are in the front of the cylinder, or in the back of the cylinder. But the idea is, purely from motion, you are seeing this cylinder rotate. It's pretty amazing actually if you think about it, and we'll talk a little bit about structure from motion later on in the class. But the idea is from a single image, but the thing moving, you can get depth. We're going to come back to that in a minute because it's a single image with the thing moving. Well, we can also have two images where the thing stays the same.

6 - Stereo

The general methods of estimating shape from some queue, shading, texture, focus, et cetera. For a long time, that was a big deal in computer vision. It was called Shape from X. Sort of very popular in the late 70s, early 80s, which I know is prehistoric history for most of you. But the fundamental idea was that from an image and from some assumption about the nature of the world, like that those houses are all the same. And therefore, they're probably getting further away. The brain could compute depth and we wanted to build machines that could do that, as well. But we, and a lot of other creatures have two eyes, so that recalls stereo. And in stereo, the key is that the image from one eye is just a little bit different than the image from the other eye. Here I see the right side of Megan's nose and I here, I see the straight part of Megan's nose. Megan's, Megan's nose is pretty straight. Mine's crooked. Anyway, so you can think of stereo as recovering the shape from the motion that's between the two different views. And the idea is to infer that 3D shape. Before talking about exactly how to do that, people have been looking at this ability of the human brain to do this for a long time. And in fact, one of the things, cool things that showed up was stereo photography. So Charles Wheatstone back in the early 1800s invented the first stereo viewer. All right. Where you would take two pictures and then put them through the lenses so that each eye saw a slightly different picture. All right. So what you've got here is this wooden thing that prevents the image in the left eye from being seen by the right. And likewise, the image in the right eye from being seen by left. And there are these lenses here that essentially train the eye right on the image that's right in front of them. People were fascinated by this. So here's a picture of Abraham Lincoln taken in 3D. So, we went from 1838 to 1860. So just 20 some odd years and already you're doing 3D portraits of the President of the United States. So it became a really compelling thing for people to see,

because you take this flat thing and all of a sudden, you would see depth. Here's a stereo pair of Thomas Edison, most invent, famous inventor of his age it says, probably true. And you can view that through a stereo viewer or you could do something really cool, you could make what was called anaglyph stereo. An anaglyph stereo put down sort of red and blue imagery. And what that did was imagine if you're looking at a white piece of paper through a blue filter. Well, everything's blue. It throws away all the light except the blue and the white, white light has blue in it. So if you put blue stuff on there, it doesn't change anything. So it basically, becomes invisible. Whereas you put a red filter in it, the blue looks very dark. Likewise, the you put the red image it becomes invisible to the red filter. And so anaglyph stereo became popular for a little while in cheap three-dimensional movies. There's a picture of those stereo viewers and this is a great picture, because it's a stereo image of kids looking at stereo pairs in the library and this is in the early 1920s. And the thing to realize is it gave you a certain sense of realism. So people used to use it to experience far away places and things that they couldn't see. Otherwise, with a, in, with a reality that they don't get when they just look at an image. So for example, here's this picture of this bridge in India. So all of a sudden, you could feel like you were in India, which was different than just looking at a static picture. And when I grew up, a lots of us had these View Master discs. Okay? Where, where you would put the discs into the little viewer and you would slide it down and you would see the two different things. And, you know, you, you, you've got educational stuff. So when I was a kid, they landed on the moon and they made stereo photographs of men on the moon, which is really cool. Talk about some place you couldn't go. Or much more importantly, you could get Spider-Man, which is really good. And of course, if we had the internet then, we would have known that you could have ordered stereo porn, but we didn't know that at the time. Now you guys, you guys, you get three-dimensional television. So for you, this is hardly new and compelling at all. But it's basically the same principle of making sure that two different images are shown to each of the eyes.

7 - Basic Idea

So here's the basic idea, all right? Two slightly different images. These are two stereo animations taken from this website listed here where we just alternate showing you the left image and the right image. Already, if you just take a look at the fence post here, you can see that these pictures were lined up in such a way that the fence post was sort of right at the same point in the two pictures. And you can see that the stuff that's in front seems to be going that way, stuff in the back goes that way and then back again. All right. Or you can switch them, but the idea is that stuff in front of what you're verged at moves in one direction and the stuff behind it goes back. We'll actually see that when we talk about the ordering constraint two lessons from now. Here's a cool one that was done of somebody diving into a pool. And you can see he adjusted the images, so that at the depth of where the swimmer is the two images are exactly lined up. So the bubbles, the splash moves in one direction and I love that the shadows on the bottom of the pool move in the other direction. So the basic idea is that from two different views from how they move, you get a sense of what their depth is.

8 - Random Dot Stereograms

So let's talk a little bit about stereovision. How do humans do it? And at the very, very end of the course we'll talk about the physiology of the human system. But today I just want to talk about the computation. If you think about it, you can imagine that stereo is done a couple of different ways. One way might be that you sort of find some regions in one image and then you find those regions in the other image and your brain compares them and sees how they've shifted relative to each other and then somehow find, figure out the depth. Another possibility is that maybe you just process the very low level images doing some sort of a comparison and get the depth. So you do fusion first, and then you do recognition. So if theres a question of whether you do monocular recognition and then fuse, or you fuse the images and then do binocular, that is given the fused images to the recognition. So back in the 60s a psychophysicist named Bela Julesz wanted to look at this question. So to test this he made a pair of synthetic images which were obtained by essentially

spraying black dots on white objects. All right? So here's a notional description comes from the Forsyth and Ponce book. Of sort of putting imagine that those are white and you sprinkled pepper or you sprinkled black dots all over the whole thing. And you took one picture from this side and another picture from that side. What you'd get are these two images. Okay? Now, when you look at those two images you don't see anything in there. But if I showed them to you in stereo, which is very difficult for me to do here you would actually see that block raised. All right. So what we can do is we can animate those we can animate random dot stereograms in the same way that we animated the swimmer. I don't have animations of this one, but I do have animations here of this well, if you look at this for a little while I hope you'll see a shark. So everybody see the shark? Okay. So this is the, the heads sort of sticking out. Then it bends around and then the tail sticks out again. So you'll notice, notice I just drew that tail. Okay. But of course the tail is not in any individual picture. It's only in understanding the difference between them. Now this was actually done from what's called an autostereogram website. You can go take a look. There are ways of making these so that if you just, you can use the same picture, cross your eyes, or, or stare at them differently that you'll get different images in the two eyes. But I think you get the idea that from these two images, you can see the depth. And what's interesting is if I present one to one eye and one to the other, you see that depth. So the answer to Ullage's question was whether you process the image first finding these regions and then line them up to do the stereo. Or you first line up the sort of little, local pixel area in order to do the, the stereo, it's clear that you do that first. You don't have, you, that you do the stereo first. You don't have to have any specific objects or region. Sort of a clean way of saying it is that human binocular fusion, fusing the two together, is not based upon matching large-scale structures. Or any individual process of the image is actually based on a low-level process that directly fuses the two images.

9 - Estimating Depth with Stereo

So now, let's think about the geometry of stereo and get to the mathematics. The basic idea is this. If I have two cameras and remember, cameras are defined by their optical centers. All right? And they're both looking at some scene point, if I can figure out which two points in the two cameras are the same point. And furthermore, if I can tell which way, if I know something about which way the cameras are pointed, then I can figure out the depth of that point. That's it. So for estimating the depth between the stereo, remember the shape between the two views, there are two things we have to consider. First is information on camera pose and sometimes referred to as calibration. How are the cameras oriented in space with respect to each other. And the second is on image point correspondences, that is which point corresponds to which. So here you see the red dot in the two images, left and right are both corresponding to that front corner. What we're going to do is we're going to swap that order. We're going to talk about doing the image correspondence first, because that's the what you're going to do for your problem set and it's all, it's the fundamentals between stereo matching. And then we save calibration for later, because calibration is also used for whole bunch of other types of geometric processing that we're going to do with the cameras.

10 - Geometry for a Simple Stereo System

What we're going to do is we're going to step through the geometry for a pretty simple stereo system. The first thing that we're going to do, is assume that the optical axes are parallel. That's these right here, and because the optical axes are perpendicular to the image planes, it means the image planes are parallel. In fact, we're going to assume that they're exactly coplanar. So this image drawn here is as if we were looking down on our camera rig system, all right? The cameras are looking out, okay, and this is the image plane right here, so we're looking down on the plane, and that will let us take a look at how things shift left to right in the stereo imagery. For our geometry here, we're going to assume our cameras are separated by Baseline B and that both cameras have a focal length of f, so f is on both sides. So for the two cameras, they both have the same focal length. Now, let's assume that we have some Point P in this scene, okay. And it's located at a distance Z in the camera coordinate systems. Now remember, the Z is the distance all the way down to the

centers of projection, okay? So that's Z not to the image plane, but to the center of projection. And if you remember, just for mathematical convenience, we put the image plane in front of the center of projection so things don't get inverted. So, that point P projects into both the left and the right images. Now, just to make sure you're paying attention, on the left image, the distance is going to be positive. And on the right image, the distance is going to be negative because our origins are right in the middle. Zero is here is in the middle of the image plane, got it? So, distance is positive in the left and negative in the right. These quantities, xl and xr, are meant to be distances, so they're positive values, but whether they move plus or minus depends upon which image we're in. So here's our point pl, and there's our point pr, and pl, is as I said, this distance xl to the right of the optical axis, so it's positive, and the point pr is to the left. So it's negative xr to the left of zero, all right? So now we can do some simple geometry to figure out the relationships between xs, ps, bs, ps and everything else that happens to be in the picture. We want to find an expression for z, so we start by using similar triangles, just like we did for projection, but these are different triangles. The one's looking down. We have the first triangle, which is based upon the point p and each of the projections. That triangle's shown in yellow. And then there is another triangle that's based upon the center of projections, and the point P. And that one is shown here in blue. See how it faded in. Let me back it up. There it is. That's the other triangle. Well, we now have two similar triangles. So we can write down what the relationship is between the edges. Namely that B minus these distances, so that's going to be this distance, okay, is to this distance, which is just Z minus f, that's same proportion as the total baseline B, I'll draw it here, is to the total Z. Okay, those similar triangles give us that relationship, so very little bit of rearranging gives us an expression for Z that says that Z is equal to f times B over xl minus xr, okay? And that difference between the left and the right amount, that's called the disparity. The cool thing here is that we're computing the distance in the scene based upon a difference between these two. Now you should ask yourself a little question here. What happens when disparity is zero? Well when disparity is zero, that means that some point in the scene projected to the same point in the left image as the right image. In other words, as I moved my, from my left to my right camera, that point didn't appear to move. Well remember, we had a quiz on this. We talked about how come the moon follows you. Well, the moon follows you because Z is so far away that you get, no change in the left and the right. So the disparity is zero, so the depth is what? It's infinite, okay. And that's the inverse relationship between disparity and depth.

11 - Left or Right Quiz

Say, I have two images from a stereo setup. One in which I have a tree on the right-hand side and a person standing in front of it a bit to the left. Stereo images are not very different. In the second image, I have the same tree. But the relative position of the person is a little different. Can you figure out which is the left image and which is the right image? Enter the word left or right in the corresponding box below each image.

12 - Left or Right Solution

Here the left one is the left image and the right one's the right image. From the left image, we know that the person is standing in front of the tree. Now if you imagine the two stereo cameras, then you'll notice that the left camera is viewing the scene from a little bit to the left vice versa for the right camera. Similarly, if we look at the right image and consider the two stereo cameras, same thing here. Now if we know that the person is standing in front of the tree, then the only combination that makes sense is that the left image is from the left camera and the right image is from the right camera. One way to think about it is the following. If you shift your viewpoint to the right, then objects that are closer will move farther to the left than objects that are farther away.

13 - Depth From Disparity

So here we're going to show you some example of disparity, okay? In the left hand side, you'll see, that there is a point, at the bottom of this window, okay? And in this image on the right over here,

and by the way, it took me a while to figure out which image was left and which image was right, and I did that by looking right up here at the trees. You can see that, in the image on the right, that the chimney has moved to the left with respect to the tree behind it, so that's the right image. What happened is, if I go to the same location in this image as in the first image, I don't land on the window. I actually have to move this far, in order to get to that window point. That's the disparity. And in fact, you can build what's called a disparity map, for every point x, y, you can say what's the disparity. And you'll notice something about this disparity image, right, notice that the stuff here is brighter than the stuff there. Well, I just made an image of disparity, and remember, disparity is inversely proportional to depth, so the brighter values are the closer zs. All right. And the smaller values are further away, and, the really small values, which are very dark, are going to be further away yet. Okay, and that's this notion of, from disparity being able to get to depth.

15 - Stories in Stereo Solution

Thank you for your response. I hope you found some examples that involve, estimating depth from stereo images. Or perhaps, novel ways of projecting stereo images so that it's easier to see three-dimensional shapes. I encourage you to share and discuss your examples on the forum.

16 - End

That ends the lesson on stereo geometry. The idea is that, if you have calibrated cameras, if you know the match, then you could compute the depth. So, this has been sort of general. The only math that we did was the relationship between disparity and depth. In the next lesson, what we'll do is we'll focus on actually finding the disparity. Figuring out which point in one image matches which point in the other image. And that's what you're going to be doing in your problem set. Then we're going to work on calibration which is determining the relative views. And by the way, you're going to do that in a different problem set. But, we're going to break these down one at a time. That's it.

3B-L2 Epipolar geometry

1 - Intro

Welcome back to Computer Vision. This is going to be the second in a series about stereo. In particular, we're going to focus on what's called epipolar geometry. We'd said last time that if we had two images and we knew something about the cameras, if we could find correspondences given that known relationship, we could find the depth. And we talked about disparity. And disparity was the idea that the location of a point in an image would change, depending upon the depth of the scene as I move my camera. And if I actually had the disparity I could make a disparity map, which tells me how all the points have shifted. And disparity was inversely proportional to the depth, and that's was this disparity map on the right is. But, how do we find the disparity? Essentially given one point in the image, we need to search for it in the other. But our intuition tells us, if I know something about how the two cameras are aligned, you give me one point in the picture on the left. It can't be anywhere on the picture on the right, it has to be somehow constrained in terms of where it can be on the left. This lesson is going to be about those constraints, and then we're going to assume those constraints when we go about actually finding the points in the next lesson.

2 - Stereo Correspondence Constraints

So, in general, if the case of two calibrated cameras. We had talked about parallel optical axes, all right. So that they were lined up and the parallel, and the planes were, were coplanar, but of course, they can be what's called verged. They could be pointed in some particular way. In fact, they don't

even have to necessarily be verged on the same point. Usually they are because we're used to human systems. But what we're going to do is we're going to talk about epipolar geometry, the epipolar constraint in both of these cases. So let's start it this way. So if I have an image on the left and some point p projecting in that image, the question is where can p prime the image of that same point on the right be? Well, basically the actual scene point could be anywhere along the ray that contains p prime. Each of these red dots correspond to possible locations along that ray of the actual scene point that's projecting to p prime. Now remember, in perspective projection, one of the things we said was that lines project into lines. So the line containing the center of projection and the ray that goes through that point, that's a real line in space, okay? That line mu, in, must project into a line in the right image. So that's the constraint we are going to talk more about.

3 - Terms

So looking at this geometry a little more carefully, the geometry of two views constrains where the corresponding pixel will meet. So the point p that projects into the image, well it could be out here in the world, that red point p, or it could be a p1. Or it could be at p2. Or it could be anywhere along that ray and the left image, which projects to that line in the right image. That line is called the epipolar line. This line is carved out by the plane that king, that contains both image centers and the actual point or just this ray. That plane intersecting the image plane in the intersection of two planes aligned, that's the epipolar line. Now of course the reverse is also true, right? So that same plane corresponds to this epipolar line. Okay? These corresponding set of lines, these corresponding set of epipolar lines, provide the constraint that we're looking for. Any point in this image that's on this line, it's match must be somewhere on that line and vice versa. And this is called the epipolar constraint. In epipolar geometry there are a variety of terms that we use all of the time and you should become familiar with them. First is the baseline, which is just the line, or the ray in space, that joins the two camera centers. The epipolar plane for a given point, I should probably say for a given point, contains the baseline and that point. So this point, this bit, this camera center, this camera center. That defines a plane and that's our epipolar plane for that point. As we showed before, the epipolar line is an intersection of the epipolar plane of a given point with the image plane. It should really say the image planes, because there's always a pair of epipolar lines. Corresponding epipolar lines are, if you have a point on one epipolar line, then, or if you have a point, you figure out its epipolar line and its matching point must be on the epi, corresponding epipolar line in the other image. Finally, we have something that we haven't talked about before, which is called the epipole. The epipole is the inner section of the baseline ray with the image plane. So assuming the image plane is big enough, right, the image plane is small it might not er, but if the image plane were big enough it would intersect the image plane there. So you should be clear, right, that since all of the epipolar planes intersect the epipoles, because it contains this ray, that means that every epipolar line will also intersect the epipoles. We're actually going to see that in just a minute.

4 - Epipolar Constraint

So, why is the epipolar constraint useful? Well, we've already said that basically, the epipolar constraint reduces the correspondence problem to a one dimensional search along the epipolar line. In this particular case, we have parallel epipolar lines, we'll see that in just a minute, but the red dot on the left that's located in the window of this beautiful chapel or whatever that is. Is located somewhere, in this epipolar line on the right, so I search for the middle of the window, I don't have to look all over the image, I'm just looking along the corresponding epipolar line. So what do epipolar lines look like, and we'll consider two cases. There's the general case on the top, where we've got verged cameras, and then there's this special case where we have parallel optical axes.

5 - Converging Cameras

In the verged case, drawn here, you can take a look at the epipolar lines by considering a couple of different points. So remember, each point defines a pair of epipolar lines, and here they're drawn. And so here's an example of epipolar lines of two verged cameras. And what that means is, that, for example, this point here since it's on, this happens to be on this line, must be somewhere on that line, and in fact, there it is, okay, and this point here. Now, only a few lines are drawn here, of course, if I was looking for this point here, there's some epipolar line, because there's some point here that defines an epipolar plane, and there would be a corresponding epipolar line on the other side, we've only drawn in a few of them. So, here's a question for you. We said that all the epipolar lines converge at the epipole, well where's the epipole? Well, let's take a look at just this image on the right. The problem is, is that this image ended here. If this image were huge, this line and then this line, here's where they would intersect, okay? So the problem is, is that the, the image, if you will, is sort of cropped, the image sensor is too small to actually find the epipole. The epipole is a mathematical construct that's the intersection of the baseline with the infinite plane that contains the image plane. So that's why the, the epipole lines look like they would converge, but they they do it off the screen.

6 - Parallel Image Planes

Now let's consider the case where we have parallel image planes. It's a little hard to see here in the picture, but the idea is that I've got two image planes. My centers of projection are sort of out here. And any point defines a plane that intersects these image planes. And it's kind of, it's what's called actually a pencil of planes. These planes rotate around that base line. But we have parallel image planes. What do the epipolar lines look like? Well, they look like this. In this particular case, because of how they're shift, they're these horizontal lines. And can see that for the corresponding three points located on that chapel, they land. Four points, sorry. That they land somewhere on these epipolar lines. So if the epipolar lines are all supposed to intersect at the epipoles. And here, we've got these horizontal epipolar lines. Where are the epipoles? Well as somebody once explained to you somewhere in some metaphysical space, sometimes various hallucinogens were involved it, powerful lines intersected infinity. Okay, so essentially he epipoles are out at infinity. And if, if you're not sure about that travel to the Himalayas and ask somebody high up on a mountain or just high up anywhere. I don't know if I'm allowed to say that. Just ask them about parallel lines. And the idea is that parallel lines here are essentially at infinity. One way of thinking about that is, as the two planes get closer and closer to being parallel, the epipoles keep moving further and further. And so when you take a limit, when they actually become parallel, then the location of the epipole is at infinity.

7 - Two Stereo Pairs Quiz

That brings us to a, a quiz. Let's consider these two stereo pairs. We have the top one, which you know the cameras are verged, because those lines intersect. But the bottom one, I'm going to tell you is from a pair of, an image pair, a stereo pair that has coplanar image planes. And the question is how do you know that image B has a parallel image planes? And I'll give you three choices. A, the epipolar lines are horizontal. B, the epipolar lines are parallel. And C, because I just said, B had parallel image planes.

8 - Two Stereo Pairs Solution

Well, you might think, and I'll go back to here, that it's because they're horizontal that we know that they're parallel image planes. But, actually, parallel image planes mean that the line, the epipolar lines are all going to be parallel. The only reason they're horizontal is because I happen to be holding the camera in such a way, the, the pair of cameras, that the baseline between them is

horizontal with respect to the vertical in the image. Right? So basically if I had held things this way or rotated my one camera that way ins, I would still have parallel epipolar lines because the epipolar lines are defined by the geometry of the camera center and the point. That is the plane, they just have to be in a plane. But I can shift them up and down, I can rotate them. And that will rotate or my parallel lines so it's not the horizontal epipolar lines that tell you that the coplanar, but the fact that they're parallel.

9 - End

That ends the lesson on epipolar geometry. In the next lesson, what we're going to assume is that we know the epipolar lines. In fact, most of what we'll talk about we'll use horizontal, parallel, lined up epipolar lines. And then we'll go about solving the correspondence problem.

3B-L3 Stereo correspondence

1 - Intro

Welcome back to computer vision. Today we're going to talk about, stereo correspondence. Up until now we've defined the epipolar geometry that talks about how the two views relate, and how if you have a point in the left image, then if you know the camera's relation, it's a one dimensional search. And in general, the epipolar lines can be, well, arbitrary, not arbitrary, but skewed in a variety of ways and located. But today, in order to make life easier, we're going to assume a bunch of, simplifying assumptions. For example, we're going to assume, basically the geometry that we drew out last time, of parallel, or coplanar actually, image planes. We're going to assume the same focal lengths of the two cameras. We're going to assume that they're, the cameras are horizontally aligned at the same height. So we're, and that the images are pulled out such that the epipolar lines, are horizontal, and, that they are, in fact lined up, so a xy location in one image is at a different x but the same y location in the other image. So look, that's a lot of assuming, and for real stereo, you wouldn't be able to do that, you'd have to do a rectification of the image first. We'll talk about that, couple lectures from now. But for now, we'll make the assumptions which will allow us to attack the correspondence problem more easily, and in fact, it's the way you're going to be doing correspondence on the problems sets.

2 - Correspondence

So what is the correspondence problem? Well, basically the epipolar constraint constrains what our solution is but it doesn't solve our solution so here we have a bunch of points in the left image and we know they lie along the epipolar line in the right image, but the question is what are the matches? And here in hypothesis 1, 2, and 3, col coded in grey here, you can see these are different possible matches. And so for each point in the left image, we have to decide what the match is in the right image or vice versa. So, to help solve the correspondence problem, what we need are some additional constraints. The epipolar constraint was a hard constraint, that is, it must be true, it's enforced by the camera jet, the relative camera geometry. But there are other types of constraints, which we might call soft constraints. One is similarity, which says that pixels in the left image should look about like that fixes the right image if you have the right match. Uniqueness says that there's up to no, no more than one match at a left pixel in the right image. Ordering says if pixels go abc in the left image they go abc in the right. And disparity gradient is limited. That means that the depth doesn't change too quickly. We'll talk about some of these today but where we're going to start with is similarity. Similarity is essentially saying that the image patch from the left should match the image patch from the right. So, to find matches in the image pair we're going to assume, first of all that most points that are visible in one view are also going to be visible in the other. So we'll go looking for them. And that the image regions for the matches are similar in appearance. So here's how we're going to do the dense correspondence. Dense correspondence meaning we're

finding match everywhere. For every pixel, let's say in the left image, we're going to take a little window around that pixel. We're going to compare it with every pixel, slash window, in the right image. And we're going to just pick the position that is either the most similar, or the least dissimilar, and that best match, that's what we're going to assume is our corresponding point. So, here's a very simple illustration. So I have a scan line across both images, and we're assuming parallel scan lines for now. And I've got my window in my left image, and I'm going to compare it to a bunch of windows in my right image. And the kuh, the matching cost is computed everywhere, and so this is just a not, a notional description of what to the matching costs as a function of disparity. So how far off I am from the, the first match. And of course the idea is, well, we're going to pick the value that has the best score. So, I might use something called sum of squared differences, which is the sum of the squared differences. Wasn't that a surprise? No. What you do is you take the two pixel windows, you overlap them, you subtract them, you square the differences, and then you sum them up. So that would be a measure, and that's why at the top it says dissimilarities. Now, sometimes, let's suppose one image was a little bit brighter than the other image, you know, for some reason the gains weren't set so well on the camera. If I just do direct subtraction, even if the match is exactly the right place. Because of that scaling of the intensity, I'm going to get square difference error, and it might not lead me to get to the best match. Now we've already talked about how to eliminate the problem of scaling. Remember, we talked about doing normalized correlation where we scale the value of the window so it has the same standard deviation? So, you could do that here, too. You could take your window and slide it across, doing normalized correlation instead of sum of square differences. And that's actually a similarity constraint.

3 - Find Best Match Quiz

All right. Let's try to do this in code. I want you to write a function, find*best*match, that takes two arguments, an image patch and a strip, which is basically the same height as the patch, but has a width equal to that of the image. Your function should return the x coordinate where the patch is found in the strip. Here is some test code. Let's load up two images from a stereo pair, left and right. This is a left image and here is the right one. If I switch between the two, you can see some key elements moving. All right, let's convert these to grayscale, double type, and scale them down to 0, 1 range. Here are the grayscale versions, left and right. Okay, let's define the location and size for the patch we want to match. Let us extract this from the left image. Here's what the patch looks like. Using the patch specifications, let's extract a strip from the right image. Note that here, since we want to go across the width of the image, we select all the columns. And here's the strip from the right image. Once you have implemented your function, call it by passing in patch*left and strip*right. You should be able to use the best x coordinate found to extract a patch from the right image. Go ahead and give it a shot.

4 - Find Best Match Solution

Let's use sum of square differences to find the best match. Let's initialize a couple of variables we'll need, and then let's run through the strip, placing the patch at every valid location. We extract the second patch of the same size from the strip at each location. And then compute the sum of square differences, and keep track of the x coordinate that gave us the minimum difference. And that's about it. Let's see how it works. Okay, so the best x coordinate found was 145. The top patch is from the left image. Here's the strip from the right image you saw earlier. And the bottom patch has been extracted from the right image. Notice that the appearance of the top of the house has changed a little bit, but our algorithm is still able to find it correctly. Now, instead of using sum of square differences, you could have also used cross-correlation. Feel free to try out alternate methods.

5 - Correspondence Problem

All right, so let's continue looking at this correspondence problem a little bit more. Here we have two images, this is courtesy of Andrew Zisserman. And we've got an epipolar line, and again, we've got horizontal epipolar lines. So here we have both scan lines, left and right image, and a plot of the intensity profiles. And of course, they're going to look pretty similar, because there should be matches along the way. But you know, there are slight differences in, in exactly what the profiles look like. There's a little bit of noise and there also might be some ambiguities of finding where you match over here. It might be challenging over there. So we're going to look at this a little bit more carefully. We've pulled out a window on the left. And we're going to slide that along the epipolar line that window on the right. All right, let's take a look at a particular window. So I'm showing you the two little bands that are sort of the height of the window. And from the left image band, we're going to pull out a particular point. All right, so here's this point, and its window that's associated with it. So that means we have to pull out the right band, that's the same, it's the epipolar band, if you will, it's the epipolar line, but with the window height. And we're going to slide that window from the left across the right band. And if we do that, and we were doing, say, cross correlation, well, you would see this nice high peak right where the thing is supposed to match. So the window was picked sort of between these two dark squares here and of course the best match is right between these two dark squares just like you would expect. The problem of course is we put a window over a nice, essentially the fact it looks a lot like a surveyor's target of black squares and white squares. What happens if our window falls over this area on the left image band that's pretty textureless? I mean, there's some variation in tensity, it's a little hard to see here. Now if I correlate that with the right band, what do I get? Well, I see a result that looks like this, okay? And of course, the question is, where's the match? Because you'll notice before, I had a nice strong match above that 0.5 value. Now I get multiple matches and I know that the match is supposed to be somewhere where the question marks are. But it'd be pretty hard to justify picking that one over any others. So how could we fix this? Well our problem, of course, is that the window is so small that it didn't catch any sort of, significant texture.

6 - Good Regions to Match Quiz

So which regions in this image do you think are good for a stereo matching? Mark the check box next to each suitable window. Let's assume that our stereo setup has coplanar image planes, and that the epipolar lines are horizontal.

7 - Good Regions to Match Solution

You've seen two of these already. The one with two black corners shutting in, turned out to have a very strong match. And we saw that this area on the wall was essentially featureless. Similarly, any region that is too bright, or too dark, has too little information to be useful. Also, regions that only have horizontal edges going through them, like this, or this one, are not useful in our stereo set up. This is because they could result in multiple matches along the epipolar line. Okay then, which ones are good? Anything with a fairly distinct corner works. So do regions with distinct vertical boundaries. If you consider the epipolar lines going through them, you'll see that there's no other region in the image along that line which matches the region.

8 - Effect of Window Size

You could say, well, instead of using small windows, use a slightly bigger window, so this is question of, how should we pick the window size, and just like when we were talking about scale, there's not going to be any easy answer. Here's an example of another stereo pair, I'm only showing you the left. So if I do a sliding window stereo, well, for a small window size, well I, I get the branches pretty well, of the tree, but you'll notice there is all this I think the technical word is crap,

all over everywhere else. That is you know, this is a disparity image, and, and my ground should be going from near to far, and there shouldn't be all this junk all over it. So what do I do? Well obviously, because of noise and things, I need to make my window bigger to get a more robust match. So if I make my window bigger, I get this nice ground pattern here, right, you can see that it goes from near to far, okay, and the trees back here are farther away. But notice what happens to the tree branches there. Okay? The window is too big, and when you put it over a branch, it gets both the branch and the background, it doesn't know what to do. Just as I said earlier in this course, scale is always an issue, there's no magic answer.

9 - Occlusion

I want to talk just a little bit about two of these other constraints that we talked about for the correspondence problem. And I can talk a lot about them, just give you some insight. One is uniqueness and the other is ordering. So, uniqueness says that there's no more than one match in the right image for every point in the left and vice versa. Same thing. Okay? So why does it say no more than one? Shouldn't it be exactly one? Well, no. The problem is occlusion. And that's illustrated here. Let's suppose I have a green bar in front of a red bar. All right. So. Here's my left image and here's my right image, and these are the pixels being seen in my left image and these are the pixels being seen in my right image, all right. You'll notice, I have two red ones here, then two green and a red, and here I have a red, two green, and two red. And the problem is that these pixels are occluded. And what we mean by occluded is that they're only visible, sometimes they're called half occluded, because this pixel is only visible in the left image. And this pixel is only visible in the right image. Right there. Okay. This happens at occlusion edges. So, if I hold it like this. So, if Megan, if I asked Megan to close her left eye, your other left she can see, she can't see this thing with her left eye. But if I asked her to change her other one, she can see it with the right eye. So, the, the tip of this finger is only visible in, in one of her eyes. So, that's, that's why you don't have necessarily a unique solution that every pixel is matched in every in every frame.

10 - Ordering Constraint

The ordering constraint basically says that if I've got pixels in my left image that go a, b c, they're going to appear in the same order a, b, c in my right image. And that's typically what happens when I look at a single solid surface. So when is this violated? Well, duh. When I'm not looking at a single solid surface. That's SSS, that's pretty cool. So what's not a single solid surface? Well, couple things. First of all, we could be not solid. Now, what does it mean to be not solid? Well, suppose you have a surface that's transparent that has some markings on it. Okay? So here we have this, supposed this as a transparent surface, all right? And we can see these points, all right? Well if they're transparent, they would go A, B, C in that order, but over here, it would go C, A, B, all right? And that's just because we can see through the surface. Well that's really weird and that almost never happens. What happens much more often is what's sometimes called the pen, floating penel, I forget. Whatever it's called. Basically, if I've got a narrow occluding surface, and you can actually do the simple experiment. If you wanted to do a stereo put your two fingers, one in, right in front of the other, and in your left eye, this finger is to the left of this, but in your right eye, the other finger is to the left of this, so it swaps. So if you have a stereo algorithm that's trying to figure out how to do the match that's a problem. I will tell you that current stere, stereo algorithms do a lot of work to handle the occlusion problem, because that happens all the time, because if I've got one edge of a, of an object in front of another object, there are going to be pixels that are visible in one eye or one camera and not the other. Current stereo algorithms are not so great on violations of the ordering constraint. Various scale problems, et cetera so, that's just sort of where the state of the art is.

11 - Match Two Strips Quiz

Let's try to implement a very simple approach for matching corresponding regions from two image strips. First, recall the find best match problem. Given an image patch and a strip, your task was to find the best x location for the patch in the strip. You are free to use this reference implementation or roll your own. Now, your task is to match two strips. You're given the two strips, left and right, as well as a block size b. Note that you are only to consider whole, non-overlapping blocks. This means if your strip is 640 wide and the block size is 100, then you only have six whole blocks. The last 40 columns of these strips are unused. Return a vector of disparity values, one for each block. All right. Here's some code to test with. We use the same pair of flower images as before and extract a strip from each image. The top one here is from the left image and the bottom one from the right. Now we should be able to call your function by passing in the two strips and the block size.

12 - Match Two Strips Solution

So, how do you go about solving this? Let's first figure out what the number of blocks should be. Since we're only interested in whole blocks, we use the floor function to round off. Okay, now initialize disparity as a row-vector. For each block, let's compute the x coordinate of the left edge. Note that MATLAB and Octave begin indexing at 1. Use this to extract an image patch from the left strip. Now find its best matching exposition in the right strip. And finally, compute disparity as the difference between expositions in the left and right images. Okay? Let's run this. Six disparity values, as expected, but they don't make much sense, do they? Let's try to plot them. Okay, so what does the 0 in the first block position mean? Well, it just says that the best match for the block in the left image was found at the same position in the right image. How about the peak at position 2? This says that the second block has apparently moved a lot. You can actually verify this visually. Look at this feature in the left strip and its position in the right strip. Different blocks have apparently moved by different amounts. Now, how do these disparity values relate to depth in the scene? Think of what are the drawbacks of a simple approach like this.

13 - Stereo Results

We've taken a look at correspondence done in a simple fashion. And in fact that simple fashion is what you guys are going to do on your problem set. I want to take a look at a couple of slightly more sophisticated methods that are actually closer to the state of the art in stereo. There are a couple of ways we do that. There's a bunch of stereo pairs out there for which there is known ground truth of depth. So here's one from the University of Tsukuba. It's a well known scene, and the image on the right is the actual depth. And these things can be generated in a variety of ways. You use a laser scanner. Some other method of being very precise and doing the matching. But the idea is, if you're developing a stereo algorithm or you're doing a paper or something, you have a scene for which you can say, I know what the actual value should be. So, the types of methods I was showing you before, which are Window based matching, on the left here we have sort of the best result of a Window based match. And on the right is the Ground truth. And we're using a false color image here. It makes it a lot easier to see mistakes. If everything was just gray values and disparities, it's harder to see how things have changed. And you'll notice that it does an okay job. You know the, the mannequin head is in front of other things, but it really does a pretty crummy job around the edges. This movie camera in the back, whoops bad color, this camera in the back is sort of all eaten up here, never mind what's going on over here. So it has some general difficulties, and what we'd like to do is to see if we can get a lot closer to that ground truth solution. So better solutions do something more than just individual correspondence estimates, all right? Essentially what they want to do is they want to optimize the set of disparity assignments, in some sense, jointly, meaning that I don't just assign this pixel to somewhere and then this pixel independently. I want to think of all of the assignments together in some way. Come up with an optimal solution. And this can be done in two different at, at least two different ways. One is to do a scanline at a time. So we're still going to do a line at a time. But for all of the pixels in that line, we're going to

think about how we want to do the assignment. And then we do that for each scanline. And then even more modern methods, the ones that are, that get even more improvement, they actually do it on a full 2D grid. So we're not thinking just across the scanline but vertically as well. So let's talk about the scanline methods. So here I have, that's Kuva University picture again, and we have two intensity profiles, the left and the right, all right? So just to be very precise about it, we have a 1D signal here, and a 1D signal here, and we have to match every pixel here to some pixel somewhere over here, all right? But what what we want to find is we want to find the best overall match, that is the whole set of matches, we want the best set.

14 - Dynamic Programming Formulation

So the way this is done, is in what's called the dynamic programming formulation. And it's a little complicated but bear with me. So what we have here is, I'm going to pretend that I've got the left scan line signal down here and each of these is supposed to be the pixels. Although, I noticed there are more dots than pixels. So, assume that there's as many squares as there are dots in this grid. And same thing over here. And let's assume for a moment that we know that the leftmost pixel here corresponds to this one. And the rightmost pixel there corresponds to that one, all right? Well then, a solution to the mapping of the left to the right is just a path from that top left-hand corner to the right, bottom right corner. So, there's a particular path, okay? And the goal is to find the best possible path. And best possible path meaning the least cost. Now, when you're located somewhere in this grid, there's only three different ways that this path can go. The simplest way is when you go on the diagonal. Now, when you go on the diagonal, what it means is, if this pixel was mapping to this one, then this one is mapping to the next one. That is, they're staying at the same disparity. So if, if I have to go plus three to match this pixel, then this one's also plus three. And I just keep going along. But, suppose I have some pixels that are visible in one image and not visible in the next. So, here I have an, an area that I'm saying is left occlusion. What I mean by that is visible to the left eye and is not visible to the right eye. So, what that means is that all of these pixels, this range here, are all being mapped to this one location. So you can think of it as that this pixel was mapped to this pixel, but all of these were not seen. So that's a left occlusion. Correspondingly, you can have right occlusions, all right? And the goal is to find the least cost through. Now, when you're going diagonally, you're cost is just the match between two pixels and that can be a match much like we did before. [INAUDIBLE] normalized correlations or sum of square differences, depending upon whether you're doing similarity or dissimilarity. But when you're not going the one-to-one, but you're actually doing the occlusion, you're, what you need to do is pay a price for occlusion pixels. And a variety of methods have been designed that tell you that say how to try to assign that cost. Some of them are Bayesian, some are other ways. Now, for those of you that know about shortest path or Dijkstra algorithms or other forms of solving this. This can be formulated as a dynamic programming problem. And dynamic programming approaches are nice because they're computationally efficient. You don't actually compute all possible paths. And yet you can still compute the best possible path. And the only thing required is that you basically have to be able to evaluate the costs of going horizontal, vertical or diagonal. And here, I've listed some references, including one by [UNKOWN] Bobick, that showed about how to do that. We introduced this thing call ground control points which was a cool way of not being very sensitive to the occlusion cost value that you use. Which was part of the problem of, of earlier methods, but this is in the way backs

15 - Coherent Stereo on 2D Grid

Looking at that scuba university thing again, here you can see that with the dynamic programming we get a lot better solution than we had before. Things are doing better here and little bit, quite a bit better there. But one of the things you'll notice is that you get these streaks. And that makes sense because every scan line is done independently of the other scan line. So it doesn't know anything about the previous solution, so it makes sense that as you change scan lines, you might actually get a reasonable shift. And unfortunately, you can't use dynamic programming to find a spatially

coherent set of disparities over two dimensional grid. You have to do another approach. So in motivating that other approach, let's think about it this way. What defines a good set of correspondences? Well, first of all, going back to our earlier work. We want each pixel to find a good appearance match from left image to right. Or, or right to left. So we talk about match quality. But the other thing is, remember when we were first doing filtering for noise, we said one of the assumptions we make is that two pixels that are nearby have similar, sort of, real values? Well when you talk about depth you can also talk about how two pixels that are nearby have a tendency to have approximately the same depth. So that means that they probably should have about the same disparity. So if, if this pixel is, has to be shifted over four, the one nearby shouldn't shift a whole lot. Okay, and I might want to penalize solutions that have lots of jumps. So the method that we're going to use now looks at both of those things. And it thinks of solving stereo as an energy minimization problem. The first term the energy mini, oh, let me. I should define that here we have i1 is the left image, i2 is the right image. We've got a window in the left, we're going to look at it across the right, but then all the way on the right is a disparity image, okay? So that's the disparity that we compute. And our energy is minimization approach requires two energy terms. The first term is a data term. And the date term just says that summed over all the pixels, that the, the window subtracted from the window on the other image, so this window minus that one squared, should be as low as possible. So, we want to, if we're going to do an energy minimization our data term basically takes a look at a look at the squared of the differences. That's similar to what we were doing before. So one way to think about it is, when we were doing corresponds through den search, we were only looking at the data term. The second term is called the smoothness term. And, you'll notice that I don't have any window here, I don't have any window here, I'm only looking at the window over here. In fact, in the smoothness term, the image doesn't come in here at all, neither the left or the right. I'm only looking at the windows of the neighbors in the disparity image. This is the disparity image here, all right. Now, this row function, this is a function that's essentially a robust norm. So what you want is you, it's okay to make small amounts of change and then as you get expensive, to make large changes, but it shouldn't get even more expensive to make even bigger changes. because that's going to be sort of an adjunct edit occlusion. So that row function's meant just to be a robust. I mean, you can think of it as a square that, that, that maxes out. But that's our smoothness term. And for energy minimization, what we do is we have a total energy which is a blend of the data term, and the smoothness term weighed by these coefficients. And what you want to find, and this is the hard part, is you want to find an assignment of disparities that minimizes this energy function.

16 - Better Results and Challenges

So the bad news is that it's hard to do. The good news, no that's not right. It's not hard to do. You're master students in computer science. Hopefully, some of you have seen the Graph Cuts algorithm. I'm not sure, we may do that later in this class when we do an algorithm thing for segmentation. Graph Cuts is a, is a well know computer science algorithm where you've got two groups of graph, you have a graph and you want to cut it in such a way that minimizes some value. The cool thing was that Boykov, Veksler, and, and Ramin Zabih applied graph cuts to some computer vision problems and in particular, they applied it to the stereo disparity problem. And when you do that, you get a result that looks like this. And this is pretty good. So, on the right here we have the ground truth, all right? And on the left you have a picture that looks pretty close to that ground truth. There's still a little bit of issues. I guess green's a bad choice here. Pick red. There's still some issues around here. And of course a little bit where the, where the occlusion happens. because remember, we've got occluded pixels near the edges. But in general, compared to that correspondent solution that I was showing you before, it's a much better solution. I also point you to this Middlebury Stereo website. There's a whole bunch of ground truth and sort of standards in stereo that's available there. I don't want to give you the impression that stereo is a totally solved problem, there are still some challenges. Textureless areas, it's very hard to know exactly where the match is, and clearly the surface has to have more influence. Occlusions are still a challenge. Violations of the brightness constancy. So, things like specularities whose value, whose position on

the surface move as I move that violates our stereo. Really large baselines. So I've got a camera way over there and a camera way over there. Well now, I've foreshortened things, right? So here I see the front of the camera and over here, probably I'm out of the field. Probably make can't, I'm totally gone. But all I see over there is the side of camera. So if I wanted to do stereo matching that's a very hard thing to do. And then there's camera calibration errors, right? So the, remember the camera calibration defines for me the epipolar lines. What if they're wrong, by a little? Well then my epipolar lines are wrong by a little. So, how do I deal with that? So these are challenges for ongoing stereo research.

17 - End

So that ends the lesson on stereo matching. In fact, that ends all of our lessons on stereo. You're going to implement the simple correspondence method. By the way, for a more advanced version of this class, we ask to do the energy minimization by either implementing or using the library for graph cuts. If you're so disposed, you might find that a really cool thing to do. When you go to implement this simple version, hopefully you're going to be impressed by two things. One is, doesn't work nearly, as well as I implied it work. And I even, didn't even imply that it would work well at all, so that's really going to be. However, your computer will be seeing in 3D. You're going to give it two pictures, we provide the pictures and your computer will recover the depth. You know, up until not that long ago, that was considered magic done by human being's brains that was un, not understood. And now you can do it on your iPhone, if you had two cameras. Anyway, I hope you'll find it cool that you can recover that depth. That ends for stereo. Talk to you soon.

3C-L1 Extrinsic camera parameters

1 - Intro

Welcome back to Computer Vision. I hope it's welcome back because if you're jumping in now you missed some really good jokes and some of the better lectures. the, this one's okay, actually. Anyway, today we're going to talk about extrinsic camera calibration. We'll define what that means in a minute. We took a little hiatus to talk about stereo so you guys could get working on your stereo, and stereo was our first look at multiview geometry, multiview cameras. And we talked about how, in order to do real depth reconstruction, we have to understand the geometry of what's gong on between the cameras, and that's what we're going to start talking about today. So, before the stereo thing, the stereo sections, we introduced a projection, perspective projection. And here is the model that we used. In particular, we had a system where we went, where we had a center of projection that was located at the origin of a three-dimensional camera system. And then we derived from simu, similar triangles the location on the image of the point projected down onto the image plane. And then in order to figure out where the point was going to land on the image, we just eliminated that last coordinate. Now we said that this was a bit of an issue because this division by Z was non-linear. And because we had to pull out the particular Z, it wasn't a constant Z, it was the particular Z. So we introduced this notion of homogeneous coordinates. And the homogeneous coordinates essentially added an, another component to the vector. And if it was 2D, it became a three long vector, 3D became four. And the idea was, that we were going to be able to convert from homogenous to non-homogenous when we needed it. But before we did that, all of our operations could be done through matrix multiplication. Which, by the way made homogenous coordinates, the whole thing, invariant under scale. I could scale a coordinate, homogenous coordinate by anything and when I did the, the normalization, divide by w here. It would go away. And, one of the reasons we did this is we said that perspective projection could now be done as a matrix multiplication. So, here I've written, one, one, and here, we've got 1 over f. And by the way, just to make life easier, I'm using the absolute value of z, so we don't have to worry about z being positive or negative. So, when I do the multiplication, I get this homogeneous coordinate. And when I want to normalize and go to unhomogeneous, I get the u v by dividing it out. But in all of this discussion

about projection, we have the notion of a camera's coordinate system. By the way, I'm going to go like this. And it's not some like, weird curse in Georgia. It's, it's a coordinate system, one, two, three. Okay, so we have an origin and a coordinate system. And we said that we put the center of projection at the camera's coordinate system. And we have the z axis, the optic axis going down the z axis. So to do geometric reasoning about the world, we need to know, we need to be able to relate the coordinate system of the, I guess I'm going to have to do this. We have to relate the coordinate system of the world to the coordinate system of the camera. And, in fact, today, what we'll do is the coordinate system from the world to the camera, and then next, I don't know, today, I don't know when you're going to watch it, next month. The next lesson will be the coordinate system from the camera 3D coordinate system, to the image.

2 - Geometric Camera Calibration

This whole thing falls under the labeling of geometric camera Calibration. In order to be able, for the camera to tell us about things in the world, we need to know the geometric relationship between the camera and the world. For reference, you can take a look at the Forsyth and Ponce book. The sections are, are listed here, and there's a nice description. So as we said, geometric camera calibration's composed of two parts. There's the first part that goes from some arbitrary world coordinate system. You know, put your origin wherever you want it to be to wherever the camera is, and that basically tells you where the camera is in the world and its pose. And then the second one is from the 3D camera to the image plane. The first one is called the Extrinsic parameters. That's the Extrinsic it goes from the world coordinate system to the, to the camera coordinate system. The Intrinsic, which we'll do next time, is from the 3D camera system to the image. So let's talk about camera pose or the orientation and location of the camera frame with respect to the world. In this diagram, this transform T is a transform that goes between the world and the camera system. Okay? And that's what this T with lower, lower w, upper C is going to mean. All right. We're going to talk more about this notation in a minute. The transformation that we're going to talk about is this going from world coordinates to camera coordinates.

3 - Degrees of Freedom Quiz

So that's a good time for a quiz. How many degrees of freedom are there in specifying the extrinsic parameters? In specifying the relationship between the world coordinate system and the camera coordinate system? A) 5, b) 6, c) 3, and it's three dimensional space, or d) 9.

4 - Degrees of Freedom Solution

Okay, well, let's take a look. This slide says for rigid body transformations we need a way to specify the six degrees of freedom of a rigid body. So the answer was six. But why six?

5 - Rigid Body Transformations

Well, you can e, easily think about it this way. Let's define a rigid body as just a collection of points whose relative positions to each other can't change. And, for the mathematicians in the audience, we're going to pretend that's a well-defined statement, okay? Or, you can think of it as a box. So, the first thing I can do is I can located that box in 3D. I can take one point of that box, say the corner here, and figure out the x, y and z location of that box. So, that's three degrees of freedom. Then, I can take some other point on that box, let's say the corner, and I can move it around. Now, I can't change it's location space arbitrarily because I'm holding this point fixed. So, essentially, this corner can move around on the sphere. So, this point here can be moved around anywhere on the sphere. So rot, this is supposed to be like rotated backwards this way, all right? And just the way you can think of on a globe. From the middle, there's a latitude and longitude to get any location. There's two degrees of freedom of a vector's direction. So, that's another two degrees of freedom.

So, we're up to five. And finally, once I have this vector specified, I can rotate, I can spin about that vector. So, the cube here, as indicated by this fancy Microsoft PowerPoint arrow that's in here, we can rotate it about that diagonal. So, that's one more degree of freedom. So, that's why there are six degrees of freedom for a rigid body. I'm going to assume most of you knew that already, but there you go. That's why there's a fast forward button.

6 - Notation

So now life gets ugly. I'm going to be using the notation from Forsyth and Ponce. It, it might not be the best notation, but it is a notation for dealing with quarter transformations, which is what we're going to be doing. So the idea here is that superscripts are going to represent what coordinate frame you're in. So here I have some point P and I've got the A coordinate frame. And the expression of the location of point P in the A coordinate frame can be thought of as a variety of ways. You can think of it as the location x, y, z in the A frame. But if you remember a little but from your, I don't know, algebra, calculus. The right way of thinking about the vector that goes from the origin to P, that's this vector here is, it's got the i component of the amount x. The j component of the amount y, A and this k component of the amount z, A. So a vector is actually the sum of these three components, the i component, j component, k component. Each scaled by the coefficients, x and A, y and A, z and A. Suppose I want to express the location of point P, whose value I might know in coordinate frame A, but I'd like to know where it is in terms of coordinate frame B. Well that's just a translation and it's handled very simply by saying, the location of P in B is just the location of P in A plus the location of the origin A expressed in the B frame, all right? And so that equation just gives us that new offset and this OA in B, that's just a three vector. That's the offset of the origin of A in the B frame. I told you this was ugly, but, we, you know, we have to slog through it. The good news is once again, homogeneous transformations or I should say, homogeneous coordinates are going to come to our rescue. Where translation can be expressed as a multiplication. So we've rewritten this top equation P in B is equal to p in a plus OA in B, as this matrix transformation. A couple of things, first of all, that i, that's a three by three identity matrix. So this is, and since OA and B is a, is a three by one. This is a four by four matrix, which means that this vector down here, that's actually a zero vector of length three transpose, so it's three zeroes in a row. It's actually 0, 0, 0 and this is 1, 1, 1, 0, 0, 0, 0. That's clear, isn't it? That's why we draw it this way. Okay? And just to remind you, translation is commutative. Okay? And you can actually show that in the matrix multiplication, if you wanted.

7 - Rotation

Now life gets uglier, rotation. What I have here is a figure, or I think this is also from four-side composite slides, and what I'm showing you is two coordinate frames, A and B. And you'll notice that A has an I vector, a J vector and a K vector. And B has an I vector, a J vector, and a K vector. And one of the important things to realize is that this P value, the vector from the origin, is, can be expressed in two ways. It can be expressed as some components in the a frame times the x, y, z components, or some components in the B frame with the components in the x, y, and z frame, right? They're the same vector, right? And what's key is understanding that there are these basis vectors and we need to know the amount of component that multiples each of them. What we want to be able to do is say we're going to rotate the frame from A to B, and that's what this says. What this says is given my point described in A, I'm going to have a rotation operator that would give me the P now expressed in the B frame. And R A to B means describing frame A in the coordinate system of B and it says, so if you gave me the location of a point in terms of the components of A, this is and it's only a rotation after applying R, I get the components in the frame B.

8 - What Does R Look Like

So what does R look like? Well there are two ways to think about this, first we'll think about it the hard way. R A to B expresses how each basis vector in A would be expressed in terms of B. So the

first column of R A B is the component of the I vector of A expressed in terms of how much it has in the i direction in B, in the j direction in B, and in the k direction in B. So you can think of it as like the dot product between i A, and each of the components of B, i B, j B, k B. And likewise each of the following columns is done, is, is that way. So, one way of thinking about this is that the columns of R A B are the i vector of A expressed in the B coordinate frame. Then the j vector of A in the B coordinate frame, and the k vector of A in the B coordinate frame. All right, so why is this true? Well let's think about it this way. Suppose I had a point, vector, in the A frame. That was just at value of 1, 0, 0, okay? So what that means it's actually a distance of 1 in the i direction of the A frame and none in the j and k of the A frame. What should the value of that be? All right? And I'll just write that down here. Well, this multiplies this. This to that so. So it would just give me the iA dotted with iB. So the first component of the transformed frame is just the i vectors amount in, in the i direction of B. Well, now lets go through this. We go one, two, three. It's again, is going to get iA.jB. Okay, and again iA.kB. In other words it's just what it says here is what we have to get out if we had just a 1, 0, 0, needs to be how that vector is dotted which each of the components. And that's why this matrix can be thought of as having it's column vectors as just being the each of the basis vectors of the a frame expressed in terms of the b frame. Do you get that? So on the sides you'll have that. So just press the pause button and that way you can. See what's going on. Just to remind you I labeled this that the columns of the rotation matrix are the axes of frame A expressed in frame B. Why? because we just went through all that nonsense showing it to you. By the way, it can also be thought of as the rows are just the column, are just the bases of the B vector expressed in the A frame, right? So here's iB in, in the, in terms of the i component of A. Here's iB in terms of the j. Here's iB in k. So you think about this is that if I were to transpose that vector. So I made the columns the rows, the rows the columns, I would now have instead of RAB, I'd have RBA, all right? This is an orthogonal matrix, right? The orthogonal matrix, all of the rows, all the columns are unit vectors that are perpendicular to each other. So the determinate so the, the magnets of determinate is 1. And it's your traditional rotation matrix and by the way, really important is that the inverse is equal to the transpose. So, if you had a rotation matrix and you want to go back and forth between the two, the inverse and transpose, which realize it has to be. Because the inverse of RAB has to be RBA and we just showed how the, the rows are the sort of the transposed of the spec to the columns.

9 - Rotation About Z Axis Example

So let's take a very simple example. So here I have two frames and I'm telling you that the rotation of A to B is just about the z-axis. So the image on the right, I'm looking down on the z-axis. Now this should look very familiar to you when I ask you, what is the rotation matrix? Why? Because you did this in algebra, right? You talked about just rotation of an angle theta, about the origin when you were doing x, y. And hopefully, you remember something that looked like this. See it says, cosine theta minus sine theta, sine theta, cosine theta. Right? George Thomas who wrote the Calculus textbook that many of us use, taught it to me as Charlies little sister, sock Charlie. And that way you remember where the minus sign goes. anyway, so the point is that this matrix here is just for rotating the x and y, keeping the z constant. So if I wanted to get an arbitrary orientation, basically what I could do is a series of rotations to get things where I want them. Turns out there are many standards about how to do that. One that most of, many of us know in math and computer vision are Euler angles. Euler angles say, you rotate about Z. If, if this was Z, you would rotate, let's pretend Z is up, rotate about Z. Rotate about the new X, and then you rotate about the new again, Z. All right? For those of you who fly airplanes, I think it's heading, pitch and roll? Maybe it's boats, I don't know. Heading, you orient yourself. So, you know northwest. You pitch, that's up and down this way, and then you roll. That's rotation about this way. All right. So you're about the world Z, the new X, the new Y. There's roll, pitch and yaw. There's azimuth, elevation and I guess roll for those of you who are used to launching mortars, azimuth, el, anyway. Basically, there are these three basic matrices, rotation about the X, Y and Z. The order matters, okay? We're not going to worry too much, in fact not at all about getting that order. But what it is, is here are the three rotation matrices written as a function of the, their angles. There's the rotation about x, rotation

about z, rotation about y. I put them in that order. Why? I have no idea they used to be a different order, but, but it doesn't matter. The idea is that you can rotate about each of these different axes. Now, whether you pre-multiply or post-multiply, that's an issue. So do we do the x1, then the y1 and the z1? Or the z1, the y1 and the x1? And that depends upon whether you're rotating in the new frame or the old frame. Then, is theta positive or negative? So you have to worry about these things really well when you do this. And this is why we build spacecraft in simulation before we build them for real? Because when it doesn't work in simulation, the engineer goes, says, I don't know, try negative 20. Because knowing which way your angles go is, is, is a very difficult thing.

10 - Rotation in Homogeneous Coordinates

How about, just an easier way. Once again, to the rescue, is going to come homogeneous coordinates. And, we're just going to assume that we have a rotation matrix, okay. So here, I took that top equation, that the p b is the rotated version of p a, and now, instead of the identity matrix in the top left, and the offset in the right, we have the rotation matrix here, and that's a three by three. So this is zero vector is a three by one of zeroes, so transposed is just zero, zero, zero, this is zero, zero, zero this way. Okay. And that makes, rotation a matrix multiplication. And again, we, we're using homogeneous coordinates. And to remind you, unlike translation, rotation is not commutative.

11 - Rigid Transformation

So now, we could do the total rigid transformations. So a total rigid transformation, if I have some point in the A system, I first have to rotate to get aligned in the B system, and then I have to offset it by whatever the offset of the A system in the B system is, that's what this equation says. Using homogeneous transformation, or homogeneous coordinates, we can do this all in one step. So here, we have a rigid transformation and it's really nice, right? We have our point here. We've rotated it, and then we've translated it. And what that says is we have this single matrix. Right? This part here is a three by three. This is a three by one. This is a one by three of 0s. This is just a 1. So our total transformation matrix is a four by four. And it does the, both the rotation and transfer, and, and translation. Cool, right? It gets better. Thank God. So here I've written what we had before. We have P in the A frame, expressed in homogeneous coordinates. Here is our four by four transformation matrix. Here is B expressed in the B frame. And I'm just going to write this as transformation from A to B. But suppose I wanted to go from B to A? Well, that would be written as transformation from B to A, and I'd have the point P in the, in the A frame. But the way to get that transformation is to just invert the A to B to get me the B to A. And then, this transformation takes the, the value from the B frame back to the A frame. And the idea is that our transformation matrices are, homogeneous transformation four by fours, are typically invertible. And so, once we have one that goes from, say, a camera to world frame, we can go from a world to camera fame or, or, or the other way around. And this invertibility of homogeneous transformations is very powerful and used all the time.

12 - Translation and Rotation

So to review, Translation and rotation. From frame A to B. To express this in the non-homogeneous or regular coordinates. We take the location of some point p in the A frame, we rotate it and then we translate it. In, in Homogeneous coordinates, we write it as this single matrix. Where the matrix has the rotation matrix in it in the top left and the translation vector located here in the right-hand column. And the key is that homogenous coordinates allow us to write this coordinate transforms as a single matrix, but I said that four times already, so you're saying like get on with it already. So, now finally, we can talk about going from World to Camera frame. Here's our equation, using sort of non-homogeneous regular coordinates. Where the idea is, if we have some point p in the world, so it's a point location in the world frame, we have to rotate it, orient it with, to know which way it would be oriented in the camera frame, and then we have the translation from the World to Camera

frame, okay? So, we have this sort of ugly equation that would get us from a point in the world to a point in the camera so that p in the C frame, that's now the point in the camera frame. In homogeneous coordinates, it's just expressed like this. The top left three by three is the coordinate, the right-hand column is the translation. And that whole four by four is referred to as the extrinsic parameter matrix, okay? This is the thing that transforms a point in the world to a point in the camera frame. By the way, that bottom row is not so important unless we're doing inverses, that bottom row is what makes this equation invertible. So when sometimes we're doing projections we're going to use the three by four instead of the four by four, but don't worry about that till the next lecture

13 - Extrinsic Parameter Matrix Quiz

That bring us to an interesting quiz. How many degrees of freedom are there in the 3x4 extrinsic parameter matrix? So 12, there are 12 numbers. B) 6, c) 9, or d) 3?

14 - Extrinsic Parameter Matrix Solution

All right? Well, what's the answer? Well, the answer is still six, remember there were six degrees of freedom? There are, only three angles heading, pitch, and roll. Euler omega, phi, kappa, or whatever. There are, three angles that define that rotation matrix, so that's not nine independent numbers, all right, there's only three angles, and then there are three translational values, and that's why there are still six extrinsic parameters, even though we can use a three by four or sometimes even a four by four. So, we've just basically taken away of turning those six numbers into a matrix form that allows us to apply it to the location of the points in one frame, to get the location of the points of another frame.

15 - End

So that ends the lesson on extrinsics or extrinsic. Not really calibration, because we're going to do the calibration part. So I should change the title of the, that was about extrinsic geometry. Later we're going to do extrinsic calibration where we figure out how a camera is oriented in the world. We're going to have to revisit this whole thing when we talk about mapping from world points to a location on an image plane, all right. But before we can do that, we're going to have to talk about, once I have the location of a point in a camera frame, where does that point end up in the image? And that's the intrinsic and we're going to do that at the next lesson.

3C-L2 Instrinsic camera parameters

1 - Intro

Welcome back introduction computer vision. Today, we're going to be talking about intrinsic camera calibration. Last time, we said, that we're going to do geometric calibration in general, and that there were two parts to calibration. The first transformation is from some arbitrary world coordinate system, to the camera system or the camera pose, and this was the extrinsic parameters, and it mapped from, world coordinates to camera coordinates, or camera coordinates to world, depending upon how you think about it. When we write it as T, W, C, it takes you from the world, to the camera. We expressed it in terms of homogeneous coordinates, where we had a world coordinate p here expressed in that world coordinate frame, and it was homogeneous so there's a one down there. And we pump it through both the rotation component and the translation component to get the three dimensional point in camera coordinates. And, that world to camera matrix and codes what were referred to as the extrinsic parameters or the extrinsic parameter matrix. We also said that, that encodes six degrees of freedom, three translation, three rotation. Today, we're going to talk about the second transformation which goes from the 3D camera coordinates to the

2D image coordinates or the 2D image plane. And these are referred to as the intrinsic parameters, and we'll again come up with the intrinsic parameter matrix.

2 - Ideal vs Real Intrinsic Parameters

So, you might say, woah, didn't we already do this? We did the ideal perspective projection, where we said that some value u was just going to be the focal length times x divided by z, and v was, was y divided by z multiplied by f, as well. So you might ask, aren't we done? Well, no, because that would be in some idealized world. The first problem going back to here is, f might be in, you know, millimeters, so we might have a ten millimeter lens, or a 50 millimeter lens, but the pixels, the screen pixels, they're in some arbitrary coordinate, right? That depends upon exactly how many pixels we get per millimeter in the sensor. So the first thing that happens is that we introduce an alpha that's just going to scale that value, because we don't really know what f is. Now sometimes people will give you an f, a focal length, in pixels, which is kind of a weird thing. But what they're actually doing is, they're giving you this combined value that is sort of the conversion from millimeters to pixels times millimeters, just given to you, and pixels. But basically, because they may be in some arbitrary units, we have a scale factor alpha. So that's one degree of freedom. But, who said the pixels are square? Megan, did you say pixels are square? No. Now, it turns out pixels are more square now than they used to be. They used to be cool and now they're, nah, never mind. Anyway, pixels are more square now than they used to be. The used to be, back in television, more of television days, pixels had the same aspect ratio as actually a television. So a pixel was wider than it was tall, and some other things, they were taller than they were wide. They weren't necessarily fixed. In fact, even some CCD arrays that I calibrated once, it turned out that well, it was almost square. It was like 95% of the height was equal to the width, all right? So because they're not exactly square, you might have a separate scaling factor between the u direction and the v direction. And so now we've introduced beta. So now we have two degrees of freedom. But we're not done. Next, well you remember we put the center of projection, when we were doing the ideal projection. We put the center projection right at the camera coordinates system. As if the image was taken so that zero, zero was right in the middle. But of course, we don't have any guarantee of this, right? The image may have been cropped out of a a section of the window. Or the, the location of the image actual sensor might need, might not be lined up with the optical axis of the camera. So we have two offsets, a u and a v offset, u0, v0. So now we're up to one, two, three, four degrees of freedom. Two scale factors and two offsets. Are we done yet? But wait, there's more. Here comes the really ugly one. We're assuming that the u direction and the v direction are actually perpendicular. What if there's actually a little bit of a skew? So u and v went out drinking one night, and they came back just off a little bit, all right? So that's what's shown in this figure here. The idea is that the ideal u and v are this way, okay? But maybe the sensors actually sampled that way and that way. That is, that the, the actual sampling of u, v is not perpen, are not perpendicular, and they're off by some angle theta. So that's what these equations are showing you here. They're showing you the relationship between the v-prime which is measured and the actual v, the u-prime and the actual u. And so, when you substitute those into those equations we just had, you get this sort of ugliness, okay? So this is the really ugly extrins, sorry, intrinsic parameter representation. And now we have, how many? Well we've got an alpha and a beta, the two scale factors. That's two degrees of freedom. A u0 and v0 for the two offsets. Plus theta, which is the skew

3 - Improving Intrinsic Parameters

This is pretty ugly, and we'd like to make it nicer, and we're going to do that through two ways. First, so here we have those uglier equa, equations and the first thing you'll notice is kind of like before, we're dividing the x's and the y's by z. All right. And so that should tell you that see I've wrote up here intrinsic parameters in non-homogeneous coordinates. Well, guess what? We're going to move to homogeneous coordinates by putting this whole thing in a matrix formulation. So now we can express the whole thing in homogeneous coordinates. Notice that here we have z times

u, z times v, z, so later when we convert back from homogeneous to non-homogeneous, we divide by z, and we get what we want. We have the x y z one over here, and we have this matrix in the middle. So we can rewrite this as, sort of, this very simple equation where we have a three-dimensional point in the camera frames. So remember, we've gone from some world, arbitrary world frame to the three-dimensional frame of the camera. And we go from that to the homogeneous pixel representation, like that, in the image. And the matrix that takes them from the camera to the image, that's the intrinsic matrix. Okay, so that matrix represents the intrinsic parameters, all right. Now fortunately, we can make it look even nicer than this. The first thing to notice is that the last column of K, when I write K as a three by four, the last column of K is zeros. And that doesn't really do very much, so we can get rid of it. And then we can do even more. Here we have our kinder, gentler intrinsics. We can use a simpler notation. Like I said, we're going to remove that last column. And we've gotten rid of the explicit thetas and things. And you'll notice that we have the five degrees of freedom. We have f, which is focal length, a, which is aspect ratio, s, which is for skew, and cx and cy, those are the offsets. By the way, remember I said we can have two different scales, right? A scale for one, and, for u and a scale for v? Or what we can have is a focal length and a relative scale between the two. Normally, we tend to think of it that way, as a focal length. That's the overall focal length of the image. And then, if there is a non-uniform relationship between the width and the height, we include that as an aspect ratio. And that's why there are five degrees of freedom, okay. Now, it turns out, this can get even easier, all right. And the way it gets really easy is we assume a certain niceness of the universe, okay. The niceness of the universe that we might assume is, if we have square pixels, if there's no skew, and if the optical center is actually in the middle, okay. Then we have no a, we have no s, we have no cx, we have no cy. All we have left is f. F is the only degree of freedom left. So when you're doing a calibration, sort of a lightweight calibration, what you'll do is you'll just search for f, assuming that your optic axis is in the middle, assuming there's no skew, and assuming that your pixels are square.

4 - Intrinsics Quiz

Quiz. The intrinsics have the following: a focal length, a pixel x size, a pixel y size, two deg, offsets and a skew. That's six. But we've said there are only five degrees of freedom. What happened? A) Because f always multiplies the pixel sizes, those three numbers are really only two degrees of freedom. B) in modern cameras, the skew is always zero so we don't worry about it anymore. C) in CCDs or CMOS cameras, the aspect ratio is carefully controlled to be exactly 1.0, so we don't model that anymore either.

5 - Intrinsics Solution

Well, the answer, as I said, is a, right? We have, the f can be multiplied by a for the two different scales or we can have the two different scales. But there aren't three numbers there. There's actually only two. And so the answer is a.

6 - Combining Extrinsic and Intrinsic Calibration Parameters

So now we found the intrinsic matrix. And last time we found the extrinsic matrix. So now we can combine them to get the total camera calibration that goes from a world point all the way through the camera coordinate to the image. So we write down our two equations here. We have our intrinsic, p prime is equal to K times the point in the camera frame. And our extrinsic which relates the world point, to the camera point. P in w is the world three dimensional coordinates transformed by the extrinsic matrix, becomes the camera three dimensional coordinates, which is used also in the intrinsic equation. And then that's converted to pixels directly. One thing to note here, is that our world coordinate system, is a four vector, it's a homogeneous. And we get out of four vector, but as we said before, for our K, instead of using the three by four, we can use the three by three, in which case we just use the x, y, z of the point in the 3D camera space. We don't have to use the whole x, y, z, 1. That's basically saying that this K can be thought of as a three by four or a three by

three. And when it's a three by three we don't have to have that 1 on the bottom there. So, putting these two equations together, what we have is we take a world point here, we pump it through our extrinsic. So, this is a three by four matrix, okay, which gets us out of three dimensional vector. And then we pump that through our intrinsic matrix, and that get us our homogeneous image coordinates. So this is a three vector. Remember we said it was z times u, z times v. Look at that. My us and vs look the same. Divided, and z, and then we convert it back. And so this whole thing can be written as a single matrix M. M for calibration in some language. I have no idea why it's called M, but we call it M.

7 - Other Ways to Write the Same Equation

Just want to write that, M in a slightly different way, because it's the way that we're actually going to make use of when we solve for this thing. So, here we have the same equations written out. You see, we have the world coordinates mapped through M, gives us the homogenous pixel coordinates. And what I've done, written here is, I've taken this M matrix, which, remember, is a three by four. And I said that you can think of it as three vectors each of length four transposed as the rows, so each of these rows is a vector. I've also introduced something new here, I don't think I've showed it on this slide before. Remember, this is what we said before, okay? I'm, I'm using s as a scale, instead of z, right, and so we have s times u, s times v, and s. And later to get out u and v, we just just divide by s. And I've put this little operator here, but it's just a squiggle with a straight line underneath, sometimes two straight lines underneath. And it's what's known as projectively similar. So you'll notice that this vector and this vector are exactly the same except for a scale factor, all right. The vector on the left multiplied by s, gives you the vector on the right. And remember, in homogeneous coordinates or using the projective, those two vectors are essentially the same, because when I use their values, I divide out by the left, by the last, component. So, that's what's referred to as projectively similar, so that's also introduced here. So, just finishing, the way I recover u and v is, I divide the dot product, the dot of the vector of the point p in the world with the first row, divided by that dot product with the third row, that's what's right here. And then to get the v value, I do the same thing, but now with the second row, all right. So basically, what we have to find, when we're going to find, when we're going to do camera calibration, is, we have to find those m elements.

8 - Camera Paramerters

So finally, we can talk about full camera parameters or camera and calibration matrix. The camera and its matrix M, and sometimes it's called pi, and that's what I've written here, as we know, is described by several parameters. It's got a translation T of the optical center from the origin of the world. We've got a rotation R of the coordinate system. We've got a focal length and aspect, f and a. Pixels or, or pixel size, sx sy. A principle point, that's the offset x, xc, yc or cx cy. And skew. And in this slide, the blue parameters are called the extrinsics and the red are the intrinsics. And we can put the whole thing together, we want to find this as a sort of a single matrix. M is going to be built up of all of the effects of our parameters, and so that looks like this. Okay? What is says is that M is a combination of translate, this is extrinsic, and rotate. So now we have the point in the 3D camera coordinate system. Project, this is just the extraction of the xyz, and then you pump that through the intrinsics. And how many degrees of freedom are there here? Well, there are 11. Five for the intrinsics, three for the rotation, three for the translation. I'll get rid of all that scribble. That equation there, that M, that is the full camera calibration matrix.

9 - End

All right, that ends this lesson. What we've done so far is we've derived how the extrinsics and the intrinsics are combined to form a single calibration matrix. And this matrix maps from some arbitrary world coordinate all the way down to a pixel value. What, you know, some x, y, z in the world, what UV does it end up in the, in the image. All right? So what are we going to do in the

next lesson? Well, the obvious thing in the next lesson is, how would you go about finding that matrix, all right? And basically, the requirement's going to be, if I gave you the location of some 3-D points in space, and I gave you the location of those points in the image, I should be able to recover that camera calibration matrix. And that's what we're going to do next time. And then, a couple of times after that, we'll say, well if we're less interested in going from a world coordinate system to a camera coordinate system, but instead, we're interested in going from one camera coordinate system to another, remember that stereo thing? How would we do that calibration? But first, we go from the world to the image.

3C-L3 Calibrating cameras

1 - Intro

All right. Welcome back to com, Introduction to Computer Vision. Today finally, we're going to get to talk about calibrating cameras. Where we're going to be calibrating with spectra 3D world. So finally, we can get to the parameters of cells. If you remember last time, we solved for the full projection equation. That was made up of really a composition of the translation, rotation, projection and intrinsics. For the rest of today, we're not going to worry about the fact that M has this internal structure, till maybe the very end. We're going to think of M just this way. That is M is going to be this matrix that's going to take your world points and homogeneous coordinates. And eventually, get you out your image points also in homogeneous coordinates. So what we're going to try to do is find that M.

2 - Calibration Using Known Points

Fundamentally at the heart of calibration is this idea of having some points whose three dimensional location in the world we know, and that we identify them in an image. That is, we can find the correspondence between this point in the image is that point in the world. And then we have to compute some sort of a mapping from a, a scene to the image. And one thing you can do is you can actually put an object like this in the scene that has a bunch of points on it. And you've measured or you know something about the shape of that object and you know where all the points are. Another way of doing it is to make use of, I mean, it's sort of the same math, but it's typically referred to as Resectioning. Which I'm pretty sure is a term that came from photogrammetry, which is a science that predates computer vision by about a century, that talked about going between images and three dimensional world, typically used for mapping, cartography, and stuff like that. So in Resectioning which is what we're going to do here. The basic idea is we're going to have some known points. So on the right here we have a picture of one of our labs. And one of the times one of my colleagues was using a theodolite. That's the thing that surveyors use in order to measure things out, you know, for property or houses. And we set up the theodolite in the lab and we established a world coordinate system. And you'll notice we put down these markers. These are actually sort of printouts of what surveyor marks look like. And what we did was we measured the three dimensional location in the world, with respect to this coordinate system, of those points, all right? So what that does is that's going to give us a set of points, X, Y, Z. And then of course, given an image like this one, we can find the location of uv in that image. Notice again that I'm using the homogenous version of the points. Here I'm calling it w. So this is w, u scaled by w v scaled by w, and that's w. And this was that original equation. So clearly, given enough points in the world and the image, I should be able to calibrate, recover the calibration matrix. And here's how.

3 - Direct Linear Calibration Homogeneous Part 1

All right, so first, some notation. As before, we've got our world times our m is our projective equation. And notice we've got two versions here. And again, what we're saying here is that uv1 is projectively similar to wu, wv and w. And in fact, in order to get back uv1, I could take wu, divide

by w in order to get that. And that's actually what's written on the bottom. All I've shown is that ui, that's for point i, I have to take the product that's the first row divided by what becomes the third row. Same thing for vi. So the point here is that I get a pair of equations for every point. For every point i here, I have an equation that's, that goes from x, y, z to ui, and x, y, z, to vi. So what we're going to do is we're going to have to solve for this m matrix using a, a whole bunch of points. Just continuing, I just slid those equations up, and now I've just carried through, just expanded it through. And one thing you'll notice is that every term has an m in it. And for those of you who know anything about linear algebra, that should start to worry you just a little bit because typically when I want to solve a linear set of equations, I hope that there's some term that doesn't have a variable in it. And we'll see that in just a sec. So, I have this pair of equations for each point, all right, writing it again like that. So now the question is how best to solve this, all right. I'm going to write this a little bit different way. So here I have all my variables, and here for just one point, I have my coefficients. So notice my coefficients involve capital Xs and Ys and Zs. Those are the values out in the world, along with the v's and the u's, which are the points in the image. Now, stretch way back into your memory from linear algebra, and you right, might remember that usually you were solving equations of ax equals b. And you could have more equations than unknowns, and you'd solve to a least square solution. But there was a particular kind of sol, of set of equations where the right-hand side was referred to as a homogeneous set of equations when they were all equal to zero. And that's why this is a homogeneous set of equations, not just homogeneous coordinates. These things are all sort of related. And when you get really, really old, you'll figure that out. But for now, you can just remember that these are homogeneous equations. So the question is, how do we solve that? Because obviously, there's a trivial solution. What's the trivial solution? Well, if a times x equals 0, and I'm looking for an x, well no matter what a is, x equals 0 would be a solution. So I could just say all the m's are equal to 0. And that would not be a very satisfying solution. So the question is, what's the best way to solve this?

4 - Direct Linear Calibration Homogeneous Part 2

So as I said, this is a homogeneous set of equations. Now, when you have a homogeneous set, like Am equals 0, and let's suppose you have a lot of equations, well, clearly what you'd like to do is you want to place some constraint on m. because I don't want to let it be 0. But then given that constraint, I want to find something that sort of computes the smallest square value of A times m. And that's what's written here. So I'm going to try to minimize the magnitude of Am. Now, m is only valid up to a scale. And we know that for a couple of reasons. Right? One is you can just take a look at these equations here. Right? Since, the everything's equal to 0, I could just scale m. Or I can go all the way back to my perspective equations. because clearly if I scale all those ms by a value, when I go from the homogenous to the nonhomogeneous, when I divide uw by w, that scale factor will go away. So m is only defined up to a scale, that matrix, all the little m's. So since it's defined only up to a scale, let's just assume it's going to be a unit vector. Okay, so it, it's magnitude is going to be length 1, and the question is, what's the best unit vector m to minimize the magnitude of Am? I'm going to tell you the solution here, and then I'm going to show you that it's the solution. And some of you will have seen this before, and some of you won't. The solution to minimizing Am, the magnitude when Am as a unit vector, is, as you all know I'm sure, because you've studied this all the time, you want to take the eigenvector of A transpose A as the smallest eigenvalue. Of course, you say. I know that all the time. I'll show it to you in a minute. And by the way, this only works when you have six or more points, and that should make sense right? How many degrees of freedom are there in m? There's 11, because it, there's 12, but only up to a scale. Well, I get two equations per point, so I'm going to need at least six points. But of course this thing works a lot better the more points you have.

5 - The SVD Trick Part 1

All right. So let's go through the solution. So here I've written as if I only had two points, point 1 and point n with a whole bunch of points in the middle. Okay. So I've got the matrix A times m

equals 0. How big is the matrix A? Well, it's 12 across. Right? Because there are 12 ms and how many rows does it have? Well, it has 2 times n, where n is the number of points. So it's a 2n by 12 matrix. M is just and m by 1 vector and here's the 0 vector of length 2n. Okay? So now let's talk about solving that. One step at a time here. All right. So here's our goal, right? We said, we want a unit vector that minimizes Am. A unit vector for m that minimizes Am. So let's do what's called the singular value decomposition and I'm going to assume that you've seen that before, at least some place. But the idea is that any matrix can be decomposed in to this UDV transpose format where V is a real orthogonal matrix. So in our case, it would probably be a 12 by 12. U is going to be a much bigger matrix. It says, orthogonal, but I should, can be more precise. Each column's orthogonal to each other, so that would be, in this case a 2n by 12. And most importantly, D is a diagonal matrix. So it only has values in the diagonal and it is customary to write it in decreasing order of absolute value. So from the biggest D to the smallest, all right? So we're going to write A is equal to UDV transposed. So minimizing Am is the same as minimizing UDV transpose m. Got it? So far nothing significant other than you have to use your singular value decomposition code from Matlab. Here comes the first little bit tricky part, that is that the magnitude of UDV transpose m is equal to the magnitude of just DV transpose m. Why? Well, these are just unit vectors. Right? The U's and V also for that matter are made up of orthogonal unit vectors. So multiplying them doesn't change the magnitude of the matrix and that's why these magnitudes are the same. And it's also why the magnitude of m is equal to the magnitude of the transposed M. For those of you who think about such things, V in this particular case is a orthogonal. In this case, 12 by 12 matrix. You can think of it as like a rotation matrix. So when you rotate a vector, you don't change its magnitude. So the magnitude of m is the same as the magnitude of V transpose m. So that means instead of minimizing UDV transpose m, we can just minimize DV transpose m, subject to, instead of being subject to m equals 1, subject to V transpose m equals 1. Got it?

6 - The SVD Trick Part 2

Now you might ask, why are we doing that transformation? Why are we doing that transformation? Here's why. It's going to let us do some very cool substitution. So we're minimizing DV-transpose m. Let's do a substitution of y equals V-transpose m. So now we're minimizing just Dy. Subject to the magnitude of y equals 1. Right? because V transpose M is y. Its magnitude has to be 1. Okay. So y is a unit vector. Why? Why not? No, never mind. Because it is. All right? But think about D times y. All right? Dy right there. Remember, D is a diagonal matrix whose elements are in decreasing order. So, what is the smallest that Dy can be? When is it the minimum? Well, it's going to be a minimum when y puts all of its weight, remember y is the unit vector, just in that last element, all right? So the best y for minimizing Dy given that D is a decreasing order diagonal matrix is just 0,0,0,0,0,1 okay. That's the y that minimizes Dy. And, since we defined y to be V transpose m, then m is equal to Vy. Why? Well, remember I said that V is orthogonal. Right. So when V is orthogonal its transpose is its inverse. So if y is equal to V transpose m, the inverse of that is just V, so m is equal to Vy. So now we've solved for m in terms of y, but, V and y, but remember. Y is just 0,0,0, all the way 0,1. So that means that this equation just pulls out the last column in V. All right, because if I take a matrix, all right, and I multiply it by some vector that's all zeroes down to 1. That 1 just multiplies the last column. So I just pull out the last column.

7 - The SVD Trick Part 3

So, m equals Vy is the last column. And, here's something, just because of what we said before, these singular values of A, that's d, well the singular values of A are the square roots of the eigenvalues of A transpose A, and the columns of V are the eigenvector of A transpose A. So, I could show you this, it's actually pretty, well, I was going to say but you know, if I just write out this, since A is we know is written to UDV, transpose. Okay. A transpose A, is just that transpose, so that's VD transpose U transpose. Well, okay, U is an orthogonal vector, so U transpose is also U inverse, so that's an identity, okay, so that goes away. D transpose D, well D is a diagonal matrix, so D transpose D is just, I'll, I'm just going to call it D squared, it's just the squared values, okay?

And V transpose V. So, I'm just, so I'm just going to write that A transpose A is equal to VD squared V transpose, and that is the equation of the eigen-decomposition, where V is the, eigenvectors of A transpose A. And by pulling out the very last one, I'm pulling out the eigenvector with the smallest value of A transpose A. Just to recap. Given that, what we have is Am equal 0 for some long A, what we do is we find the eigenvector of A transpose A with the smallest eigenvalue, and that's our m. That is the set of values that create the calibration matrix for us. Cool.

8 - Direct Linear Calibration Inhomogeneous

How about another way, okay? I'm going to show you another approach. Which in some sense is easier to understand, but it's actually not as good. That's why I showed you the other one first, all right? So here I'm using the same equation again. Now using uv1 as being projectively similar to m times XYZ1. But remember, if m can be changed by a scale value without affecting anything, then I could divide all the values by whatever that was on the bottom right-hand corner of m. And what I would get out would be a 1 here. So I can just put a 1 in the bottom right-hand corner of that matrix. And the reason I do that is now my equations for Ui and Vi become this. All right? And what you'll notice is I now have terms when I multiply out. This, the 1s and the denominator multiply Ui and Vi, which remember I, those are knowns, because those are the locations of the points in the image. So I now have terms that don't have the variables in them. That's why this is referred to as the inhomogeneous solution. And one of the reasons this is not as good is suppose the original m23 was supposed to be much, much smaller than the other values, that is close to zero compared to the others. Well you've just set a number that was supposed to be really close to zero to one, that's dangerous with respect to numerical stability. So in general and, and also in terms of overall minimization, doing the singular value decomposition or doing the eigenvector finding on a transpose a, is, is the preferred method. And, by the way, it's one line in MATLAB, so you might as well just do it.

9 - Direct Linear Calibration Transformation

So what I just told you is referred to as the direct linear calibration or transform or transformation method. It has a couple of advantages. One is, it's pretty simple to formulate and to solve. Like I said, it's literally, you just take your points, and you make that a matrix. You then do some SVD. You pull out the column. Bang, there's your m. Now these methods are referred to as minimizing an algebraic error because essentially what we did was we set a m. We fixed m to be a unit vector and then algebraically forced found the m that gave us the smallest algebraic magnitude of a m. There are some disadvantages to this method. First is that it doesn't directly tell you about the camera parameters. But we'll get, we'll fix some of that in just a minute. It's also approximate because that m was based upon a particular matrix based upon pure extrinsic and intrinsic projection. What if we had something like radial distortion that we could actually model, but we couldn't model within our projective transform equation. How would we do that? Suppose I actually knew the focal length, right? I went out and bought a really expensive lens with focal length 3.746586, really precise millimeters. I don't want my m screwing with that. It needs to stay that value. I want to come up with the minimum solution using that precise value. And then finally, another problem, I put down mostly, I think it's just another one of those problems. It's not actually minimizing the thing that I really want minimized. It's minimized this kind of cute algebraic trick that I had. So, what is it that I actually want to minimize? Clearly, the amount of weight I put on every holiday. But besides that, what? Okay, that was not even a funny joke, but it, it was going too long without any jokes.

10 - Geometric Error

So what is the right error function? Well, we refer to this error function as the geometric error, and it works as follows. Let's suppose I have some points in the world capital Xi, and here they are shown in yellow. And furthermore, let's suppose I have an image and I know where those points are in the image. And those are the, the red Xi's here and these are these little red points here. Now,

when I specify a camera projection, some sort of an m. Right? What I'm saying is I know where those capital Xi's, the real points, are supposed to project to the image plane. And what I want to do is I want to find the M that gives me the smallest distance between the predicted X size and the ones that I actually found. And that's what this equation here says. What it says is we have an error function that is the sum of the distances between the observed points, those are the red Xi's. And the predicted image locations, the white Xi's here of, of where the, the particular M predicts that the points in the world would land in the image. And what I want to do is I want to find the M that minimizes that value. And now what's kind of cool is, I might have a more complicated mapping from points in the world to my 2D points that include things like, modeling radial distortion, but fix for me the focal length. The idea is I'd have some great big nonlinear function that does the full projection. And what I want to do is I want to minimize this error function by min, by manipulating the parameters of that big nonlinear function. And you might do that using your favorite Levenberg–Marquardt or some other way of doing nonlinear minimization. So if you read about this in a book and the, it says, the Gold Standard. Well, the Gold Standard of this type of work is the book on Multi-View Geometry by Hartley and Zisserman. What they refer to as the Gold Standard of calibration is, Given some number of points, more than, greater than or equal to, but should be greater than 6, where you have 3D to 2D point correspondences. The way to find, they refer to it as the Maximum Likelihood Estimate. We could talk about why that's the case later, I'll tell you right now. You're assuming that your the place that you located the points in the image was perturbed by a little bit of Gaussian noise. So you maximize the likelihood by making that as small as possible, and you want to minimize the square error, if it's a Gaussian error. So the way you do that is as follows. You first compute a linear solution. Now Hartley and Zisserman talk a little bit and, and this is for those of you who are going to be more involved in doing some of this. What you might want to do is, you know, if you're measuring your points out here in meters, you might have some big numbers or small numbers depending upon your coordinate system. Your size of your image, you, you may have different pixel values. They do a normalization, so that everything sort of goes between zero and one to try to get numerical stability. So when they do that, they, they create new vert, new external world points and new image points. You can do that or not. Then what you do is you solve the thing I showed you before. The direct linear transformation using, typically, the eigenvector approach. That gives you a starting point for the M matrix, which gives you a starting point for your nonlinear function. And then what you can do is you can try, u, u, using the M, you project the points. And then you can minimize that total distance that's what's written down below using, like I said, your favorite nonlinear method. After you've done all that, if you've done the normalization, you have to denormalize or, or, or translate back from the, the normalized methods. So that's how you get out your M. And that's what they refer to as the Gold Standard alg, of, of calibration. We only use bronze, which is why in your problem set, you're going to use just the direct linear method. And you're going to see that even without, I, I will tell you this, that the first time we did this we had people do the normalization and it turns out the normalization by just some, but not a lot. When we do it this time, you can use the normalization or not.

11 - The Pure Way

So now we didn't have M. Now we know, that M encodes all the parameters, the extrinsic parameters and the intrinsics. So we should be able to find things about the camera from M. For example, the focal length which should be an intrinsic. Or, how about the camera center? In order to be able to do the projection, the camera center, remember the translation vector that translates from the world coordinate to the camera center. That, if we just knew that translational value, that would tell us the location of the camera center in the world coordinate frame. So we should be able to get that directly from the end matrix. So I'm going to tell you about two ways of doing that. Sort of the pure way,which is beautiful, and then the easy way. So we'll start with the pure way. Here's a slight change in notation just to pull out our parts. Okay. So we have our matrix M, and I'm just going to assume that M, which is a 3 by 4. Is going to be made up of this 3 by 3, which I'll call Q and b is just this 3 by 1, it's just a, a vector. Okay, so we've got Q and b. I'm going to claim that the

center, C, which is a point, which I'm going to represent as a vector, is in the null space. Of the projection matrix. And what that means is that if you multiply M times C, it equals zero. And if you found such a C, that would be the camera center. And you should just trust me. Okay, maybe you don't trust me. Let me show you why that's true. All right? It's really pretty simple. Except drawing it is painful, all right? Let's suppose I have some camera center C. And I've got some plane. And I have some point P that's out here. So I have a ray that goes from P to C. And I'm going to have a point X that's somewhere on that ray. Okay, so that's where X is. All right? So that's all that this equation says. That X is equal to something that's a blend between p and c. Alright, lambda times P, one minus lambda, times C. So given some M, that's the camera, the projection. All right? Of X is just MX and just by linear algebra, that projection is just lambda MP plus 1 minus lambda MC. Now for any point, P, all the points on that ray have to land at the image of P. So, any point P, all the points on this ray, have to land at the same point on the image, okay? So, any point X along here will land at that same point, no matter what lambda is. Well, if this equation has to be true, no matter what lambda is. And it has to land at MP because lambda might be one. That means that this value, MC, has to be equal to zero. So if C is the center, MC equals zero. And that's why, I could, I said before that the center C can just be found by finding the null space. Of the projection matrix M. That wasn't too painful.

12 - The Easy Way

But suppose you actually wanted to like do something useful. What would be the easy way of doing that? Well the easy way is I just give you the formula. So, if M is equal to Q b as I said before, then, the center is just found by taking this quantity above that quantity in a vector. So this is a, remember Q is a three by three, so minus Q inverse is a three by three, b is a three by one, so this is a three by one, and then I add the one there, that is the vector that is the camera center in the world coordinate frame. So if I made, you know, the corner of that table that you can't see, there's really a corner over there, if I said that's the origin, right, and this way is x, and, this way is y, and that way is z, and then I have a camera here. And I, took a look at a bunch of points here, and I gave you the value of those points in this coordinate system, and then I located those points, in my image plane. And then I did all that math to compute that M vector, and that, that M matrix, and then I pulled out the three by three and called that Q and three by one and called that b. And then I took minus Q inverse times b over one, that, vector would be the actual location of this camera in that coord, world coordinate system. Pretty, pretty cool.

13 - Multi Plane Calibration

Finally, and this is the half bit that I was telling you about, there's an alternative way of doing calibration, which is pretty cool. And in fact, it's really what everybody is using today, because typically you don't have all these points. And it's a way of getting the intrinsics, and then if you know where you put these checker boards it's extrinsics. Basically what you do is you take a checker board and you move it around. And you can end up calibrating the camera, with respect to that checkerboard. The reason everybody uses that, these days, is that there's. First of all, you can print out a checkerboard and it the checkerboard has to be on a plane, well that's easy you print out a checkerboard. You gotta make sure your printer doesn't screw things up, you gotta make sure that it actually comes out that squares are squares. You mount it on something that's a nice piece of rigid cardboard or foam core. And you don't have to know the positions and orientations in order to get the intrinsics of the camera. And later you can get the intrinsics as well. And why does everybody use this? Because the code is easily available online. OpenCV supports this directly. We list here a MATLAB version of it. If you go to Zhengyou Zhang's web site, he's like doctor calibration. He did spectacular work for us, thesis etc, in calibration. I'll also tell you that there's something cool you can do, and in fact, the a student of mine, Kelsey Hawkins did this. And I, I think the code is made publicly available. If you take a checkerboard and you mount it on the end of a robot arm. And remember, a robot knows where its arm is in terms of its coordinate system. If you also know let's say the offset of that checkerboard from the the end effector, the hand. Then what you can do

is with a camera you can just move that checkerboard around using the robot, and of course we know the 3D location of the checkerboard at every point in time that we take an image. And so, by taking those images of the checkerboard, you end up calibrating the camera to the robot's coordinate system. So, it's a cool use of doing this directly within robotics.

14 - End

So this concludes the lesson on calibration. In the homework, you're going to use the theory we developed for direct linear calibration, and we'll give you those images that have the three-dimensional locations of the points and the two, and you get the images. And you have to solve for the calibration. And it's, it's interesting that you can actually recover that relatively precisely. In the next lessons, what we're going to do is we're going to move away from the notion of rigidly calibrating the camera to the world. And instead, we're going to talk about the relationship between multiple cameras, specifically starting with two cameras. And that's going to eventually get to the point, where we were talking about in stereo, that if you know the relationship between cameras, that you can know where the epipolar lines are. So basically, if I gave you some corresponding points, you should be able to solve for the relationship between the cameras that will allow you to find those epipolar lines. And that's what we'll be doing over the next three or four lessons

3D-L1 Image to image projections

1 - Intro

Welcome back to intro to computer vision. Today we're going to start a section that's really on mapping from multiple, across multiple images. Image to images. And so I'll refer to it as n views. Mostly we're going to think of about two views, but a lot of this stuff extends to having multiple views. Today what we're going to do is talk about image to image projections. because last time we talked about calibration. And calibration was all about going from the world to the image, and we made a stop. We go from the world 3D coordinate frame, to a camera coord, 3D coordinate frame, and then we do a projection from a 3D camera coordinate frame to a 2D image frame. And we built that m matrix, and we talked about if I gave you 3D points in the world and I gave you 2D points in the image, how you could figure out that that relationship is. For the next several lessons, we'll be talking about mapping from one image to the other. And this is going to be important both for stereo and for other applications where you have multiple views of, of the same scene.

2 - 2D Transformations

So here, I have just I think this is a figure from Forsythe and Ponds. Maybe it's from Salinski. I hope their lawyers aren't watching. It's somebody. And basically, it talks about how you can transform an image. And so on the bottom left hand corner, you have a square. And you can see, you could just translate it. They, they call it Euclidean, but that's just a rigid transformation where you have the rotation translation. You have similarity, affine, and projective, and we're going to talk about each of those in terms of the mathematics. So let's take the simplest example. So the simpi, the simplest example here is translation. So we have an x and, and by the way in, in this cool little figure. So the original x and y is the green box, and take our x and we just add some translation vector to it. And we saw that it was an easy way to describe this using homogeneous coordinates where we just do xy and 1, and we add 1 to the end. And then we can just have a transformation matrix here, which is just made up of an identity plus the translation vector that we add from over here. And to keep the whole thing in homogeneous coordinates, we would just say we would make a, instead of a 2 by 3, would make this a 3 by 3. So that 0 vector now is on the bottom row. And what we do is we get from one homogeneous vector to another. Okay? So that would be translation. By the way, because these are now just matrix multiplications, we could chain these transformations.

3 - Special Projective Transformations

So we also have talked about projective transformations, and for, remember for 3D, we said that we used 4D, four length vectors, and we had four by four. Well for 2D images, it's just a three by three matrix applied to a homogenous coordinates. So here I've written this as I have an original vector, xyw, w being the homogenous coordinate. I apply my three by three, I get a new x prime, y prime, w prime. And of course, to go to the nonhomogeneous, I have to divide through by the w prime later. So using this, sort of, general framework, we can look at all of the transformations. So let's go back to translations. So translation in this framework is just the identity matrix up here, the translation there, the 0, 0, 1 there. And we go from xy1 to x prime y prime 1, and in fact, we didn't have to do any division because we maintained that one. And that's drawn with this cool little picture here, where we're just take this square and we could just translate it around, nothing changes orientation size. And translation preserves the lengths and areas of, of objects. So as I move them around the angles stay the same. Even the orientation stays the same. And, because this is a, a projective transform, lines remain lines. And lines remaining lines is going to be key to everything we do for at least five lessons. Well, four or five lessons. All right, so that's translation. A rigid body transformation could be represented this way as well. It's called Euclidian or, or rigid body. And now instead of having just a pure identity matrix, we have this rotation matrix, so we can rotate in the plane and we can translate. So we have three degrees of freedom here. And again we've maintained a rigid body, so lengths and areas are all preserved, angles are preserved. Lines are preserves, but of course the location and the orientation can change. Another one which people don't think of quite as much is a sim, it's called a similarity transform. In a similarity transform we have translation, here, the rotation by the thetas, and you notice everything is scaled by a. Okay. So that's an expansion, that's a scale. So this translation, rotation, and scale. So how many degrees of freedom of that? four. Right? Two for translation, a rotation, and a scale. Now, one that we use actually a modest amount in computer vision is what's referred to as an affine transform. Okay, and in an affine transform you'll see that we have six degrees of freedom here. And I've just written them as a linear transformation. A, b, c, d, e, f, okay? And again, we go from xy1 to x prime y prime 1. And what an affine transformation can do is essentially map. And we'll look at this later. Any three points to any other three points, all right? And what you do is you're, you're essentially now being able to translate, rotate, scale, and skew the image. And there's skew in both directions. So there's six degrees of, of freedom. You still maintain parallel lines. So the edges of the, this square becoming edges of a parallelogram. Okay. And ratios of areas remain the same. So if I had one area here that was half the size of the area to the left, then its transformation would be half the size over here. But the area itself can change right through scaling. All right. And again lines are mapped to lines. Notice that we don't have to do the division still.

4 - Projective Transformations

Remember, these are homogeneous coordinates, so here I wrote it instead of w but with s's. So it's full homogeneous coordinate system, which will go from the homogeneous through a linear operation to a scaled version of the new x prime, y prime. So, the full transformation is referred to as a general projective transformation, or a homography. And a homography still maps lines to lines, because remember we talked about prospective transforms always maintain lines, and things that are inside of each other stay inside of each other. But this is what's typically sometimes referred to as perspective transform. So the idea is if I had a plane and I was very close to you and I tilted it this way. And we'll do more of this later. The edge that's near you will seem longer than the edge that's farther away from you. And we'll prove later that the transformation of a plane can be represented this way. Now of course, because this is a homogeneous transform, I can scale everything by constant and it has no impact. So I can make this last degree of freedom be a 1, and I'll get a slightly different w, but it doesn't matter, because I'm going to divide everything through by w anyway. So that 8 degrees of freedom, that's the full, it's referred to as the homography.

5 - Transformation Quiz 1

Now I want to do a little series of quizzes that relate, essentially, the transformations to points. So here's quiz number one. Suppose I told you the transformation from image A to image B is a pure translation. How many pairs of corresponding points would you need to know to compute the transformation? So I say pairs of points, meaning pairs of corresponding points, meaning, I've got a point in A, and a point in B, and that's a single pair of corresponding points. So the question is, how many pairs would I need in order to do tran, figure out the translation, a, 3, b, 1, c, 2, d, 4.

6 - Transformation Solution 1

All right, well the answer is b, one point. Because translation is a one point transformation. If you tell me where this bottom left-hand corner, better make it bigger, big point, bottom left hand corner is over here, I knew where, I know where the rest of the image, goes. Couple of ways of thinking about that, one to think about is, there is only two unknowns, and every pair of corresponding points gives me two equations, it tells me how the two xs align and the two ys align. So, each pair of points gives me two equations, I have two unknowns, so I only need one pair of points, got it? Cool.

7 - Transformation Quiz 2

Quiz two. Two, my finger's a little slow, two. Suppose I told you that the transform from image A to image B is affine. 'Kay, now you have to go back through a few slides, remember what affine is, go ahead. Okay. How many pairs of corresponding points would you need to compute, the affine transformation. Same set of answers, a 3, b 1, c 2, d 4. How many pairs of corresponding points to do the affine?

8 - Transformation Solution 2

Well, the answer is, a, had to go back and see, because it's a three-point transformation. And it's written here, there are six unknowns in the affine transform. Okay, it's linear. Each pair of points gives me two equations, I need three pairs of points, which is what we said before. I could take these three points and know where these are. And then, everything follows from that.

9 - Transformation Quiz 3

Suppose I told you the transform from image A to image B is a homography. Homography. Your word processor part doesn't know about homography. Mine didn't, it keeps changing it to photography and other things. Homography. How many pairs of corresponding points would you need to be able to compute the transformation? Same set of answers. A, 3, b, 1, c, 2, d, 4

10 - Transformation Solution 3

How many points? Well, the answer is 4, which is d. All right. Why? Because we have, well wait a minute, it looks like nine unknowns, right? It's three by three, but no, you've seen this before. It's only matters up to a scale factor, right? I can scale all those entries in the matrix by a constant and when I map it back out to the non-homogeneous coordinates, when I divide it out, that scale factor goes away. So there are eight unknowns, eight degrees of freedom. Again, I can take four points, the four points of a rectangle, pop them down here, they don't have to be four points of a rectangle, it's just easier to see. And once I've mapped four points to four points, assuming I don't have any weird degeneracies, then I could compute, the homography. What do I mean about weird degeneracies? Well here would be one. We said, that points that start on a line, have to stay on a line. So, if I took four points that were on a line, and then I mapped them to four new points that

were not on the line, the system should explode and it would, and your computer would melt and all that kind of stuff, because, there is no linear solution of this projection that would map four collinear points into four non-collinear points. But in general if I have four points distributed spatially and I put them down and I map them to four other points distributed spatially then I'm able to compute the homography. Many times through this class I'm going to ask you, how many points do you need to compute a homography and I hope at least you'll scream four. I'm amazed when I do this in class, I say how many points you need to do amography? And they don't know, and it's only been like eight minutes since I did this, so pay attention.

11 - End

Believe it or not, that's the end of this lesson about mapping from image to image. It's a short lesson, I know, I needed a break, or you needed a break, or somebody needed a break. Anyway, what was happening was I had these lessons and they were going on for like, 12 hours, and I said, that's probably not good. So, in the interest of biobreaks everywhere, we decided that we would shorten, some of these. So, so that ends our sort of image to image transformation using the projective, transforms. And now, we're going to continue our conversation about projective geometry, and, it'll get a little bumpy, so you know, take a breath; we'll charge ahead. The cool thing is, the next thing we're going to do is make serious use of homographies to, do something cool that will, wow and amaze your friends. assuming, you have relatively nerdy and boring friends. otherwise, their going to tell you to get out a little bit more often. anyway, talk to you soon.

3D-L2 Homographies and mosaics

1 - Intro

Welcome back. All right, so today we continue on our tour through multiple views and views, and today we're going to talk about homographies, which we learned a little bit about last time, and a cool thing, mosaics. In particular last time we talked about a variety of projected transformations that said there were some special cases like translation similarity that did not divide out by that last time. But then the general case had eight degrees of freedom. Eight coefficients and you did do that division to go from homogeneous to inhomogeneous. And that last one, the one that used all eight, was called a homography. And we're going to use that extensively today.

2 - Projective Planes

This was the description of Projective Transformations for 2D coordinates. It's 3 by 3 applied to the homogeneous. So xyw gets mapped to w prime x prime, w prime y prime, w prime. And to go to non homogeneous or inhomogeneous, you divide through by the, the w. So now I'm going to do just a little bit of the projective geometry that relates to the homogenous coordinates we've been talking about, and we'll do that a lot more in the next lesson. So what is the geometric intuition behind using homogeneous coordinates? The bottom line is that a point in the image can be thought of as a ray in projective space. So what I mean by that is so every point x, y, on the image plane. Let's change the color here so you can these. So here we have this point and we're talking about x, y. Can be thought of as on this ray, intersecting an image plane that set a value of z equal to 1. By the way, and I stole this slide from someone, I can't re, somebody was very clever here. They wanted positive z to go along the direction of the camera. So you notice they have x this way. This is negative y. So it means positive y is down, and if x to the right and positive y is down, x to y, means z goes that way. It was a way of keeping a right handed coordinate system. Don't worry, you will screw this up. That's okay. I ju, in case somebody saw the picture and wondering why it said that, that's what it is. Back to our regularly scheduled program. All right. So if this point on the image x,y,1 is just this ray intersecting the image plane. That means that any point along that ray is on that, it projects to the same point of the image plane. And the points along that ray are the scaled

sx, sy, and s, if you multiply by scale. And remember in homogenous coordinates, sx, sy, and s, you divide by the s to get out the new x, the new y. That's the relationship between that homogeneous transformation dividing out by the third coordinate. And this notion that a point on an image is essentially equivalent to a ray in space, coming from the center of projection and intersecting the image plane. We're going to make way more use of that in our conversation next time. We're only going to use it a little bit this time.

3 - Image Reprojection

So given that little bit of information, we can now talk about image reprojection. Here's the basic question. How do we relate two images, so I have two images, projective plane one, projective plane two, where the center of projection of the camera was the same. 'Kay, so I've got the camera, and it's center projection is fixed, and I take two images. So the question is, how do I map from projective plane one to projective plane two? Well here's the answer, basically think of that image or the point on the image as we said is it's actually a ray in space, and it's a ray from the center projection, through the image plane intersecting the image plane. And then in order to figure out how to map from projective plane one to projective plane two, I basically have to figure out where that same ray intersects projective plane two. So if I knew where that plane was, then I could just figure out where that point is. All right? And so that would be, you know, right here. 'Kay? So what should be clear is, that in order to figure out where this ray was intersecting projective plane two, I didn't actually have to know where in the world that ray hit. I just need to know the camera center, the location of the point in image plane one that defines the ray, and then I intersect the ray with projective plane two. Where it hits in the world is irrelevant. What that means is, instead of thinking of this as a reprojection from 3D to 2D in figuring out how it projects to some other 2D plane, when I'm going from two planes that are from the same center, I really want to think of this as a 2D image transformation. A 2D, it says here, image warp from one image plane to the other. I don't have to worry about the third dimension, the 3D world. So a very cool application of this is what's referred to as image mosaics, or sometimes called image panoramas. So would you like to know how to make an image panorama, sometimes called an image mosaic, which is way easier to say than panoram, ra, pa, pan, panama. Maybe they make them in Panama. I don't know. We're going to make panoramas, and we're going to use homographies to do this. All right, so here's the basic procedure of how to make a panorama. You're going to take a sequence of images from the same position. So you might say, you might ask, well if it's from the same position, don't I get the same picture? No, you're going to rotate the camera about its optical center. So the position stays the same, but the direction that the camera's pointing can change. Then what you're going to do is you compute a transform between the second image and the first, or the first and the second, and then what you do is you transform that second image to overlap with the first according to the, the, the correct transformation. And then what you'll have to do is you'll have to blend those two pictures together to make the mosaic. And of course, if there are more images then you have to repeat. But wait a minute. How come I don't have to worry about the 3D world in order to put together a panorama of images that are taken of this whole big world that's got different depths etc. What about the 3D, 3D geometry of the scene? Why aren't we using it? Well, you can probably guess it's that thing that we said before. About that the, when you're projecting from one image plane to the next, if they're taken from the same camera center, the 3D world is irrelevant. So I don't actually have to know the scene structure in order to paste together these pictures to make a panorama.

4 - Natural Geometry

So there's a cool way of thinking about image reprojection, sort of a natural geometry. The first is, think about take one picture, another picture, another picture and you'll see here that all of these pictures were taken by rotating my camera. And I've drawn it as if my camera's center projection was perfectly aligned. Of course it's never perfectly, but pretty close. Although you can buy little tripods that make it easy to rotate the camera right above the focal point of the lens. In fact, if if some of you have a really good camera it'll show you where the film plane, or the sensor plane of

of of the camera is. And then if you know the focal length of your camera, of your lens, sorry, you. Let's suppose it's a 50 millimeter lens, you would measure 50 millimeters from that point, and you know that's the focal point of my lens. And you could set your tripod up so that the axis of rotation is right underneath that, and that's how you'd actually do that rotation. So anyway, so we show here that we have three images taken from three different orientations. Then, what we want is, we're going to project all of these images onto some single mosaic projective plane. All right? So here we're showing projecting image one, projecting image two, and projecting image three. All right? And you're, so the idea is that you're projecting everything down onto a common plane. And of course, there is some overlap. Which is actually a good thing, because in order to figure this out you're going to need overlap. But of course you're also going to have to figure out a) how to transform these images, and b) how to blend them. So this is way of thinking of mosaics is that these are images that are reprojected on a common plane, and the mosaic is formed on that plane. Before we go further, this only works for sort of a limited field of view. But you might be thinking to yourself, wait a minute, can't I take pictures all day? Spinning around on my tripod, now you get to see how much hair I don't have on the back of my head. I know you've always wondered about this. Anyway, it feels like you should be able to paste together pictures going all the way around. And you can. And in fact Photoshop and even little cameras that you buy do this for you. They're not actually projecting it onto a plane. They're projecting it onto some other surface like a cylinder. And then you unwrap the cylinder. And you get a panorama. It turns out that's not actually a real picture. It's sort of a, I mean it's a real picture, it's made up, but, but it's not a particular scene. Whereas when you do a panorama onto a plane, it's as if you had a super wide field of view camera. So it's, it's two different kinds of things. But in the, in the, when you do the really wide ones it maps it on to a cylinder. So, how are we going to do this, all right? So here's an example. Again, courtesy of Steve Sikes. So what we're going to do is, we're going to take these multiple pictures. These are from the Great Wall of China. Thank you Steve. And, you know, what's drawn here is this idea that we've taken all of these pictures from the same center of projection. So, how do we do this? Well, as we talked about before, the projection between two planes, shown here as, P prime equals H P, is just a homography. Remember that from last time? A homography that allows us to map from one plane to the next. In fact, a little bit later I'm going to show you a, a nice little proof that says a mapping, a projective mapping between any two planes can be represented as a homography. But here we've represented it as just H going from one set of points to another.

5 - Homographies

So let's now compute this homography for the, the Great Wall of China. Actually Steve did this a really long time ago, but I'm going to pretend that we're doing it right now. So in order to compute homography, the first thing we need, is we need sets of corresponding points all right? So on the left, we have the original points, x1, y1, x2, y2, all the way up to xN yN. We have a bunch of points, and on the right, we have the corresponding points in the second image. X1 prime, Y1 prime, X2, and for now, we're going to assume that these points have been found by that tried and true method of paying somebody else to click on them. That's called Amazon Mechanical Turk or something where, or maybe you're an undergraduate, and your advisor's beating you up, and so you have to do all the clicking, or maybe you're doing this for a problem set for a computer vision class and you have to do the clicking. But basically is somehow you've identified corresponding points between the two images. So, what we have to do now, is we have to solve for this equation, right? We know that the relationship between p and p prime is going to be this homogeneous coordinate, transform, or homogenous, homography H. So how do we solve for H given some number of points? Well, there are two ways. The first way is the non-homogeneous. Boo, hiss, all right. Going back here, even though there are nine variables in the, in the, matrix, we know there are only eight degrees of freedom because it's only known up to a scale value. So we can set, for example, the last entry to one, all right? Well then, we can set up a linear set of equations, Equals b, where, the vector now is of h where it's going to be a, b, c, d, e, f, g, h. But this h isn't the same as that h. This is the h that's all the elements of the big matrix h, and this is dumb little h. Okay, so we got a through f, pretty smart, and dumb little h, and then g is in there somewhere. You, you get it.

There's eight unknowns. Okay, if I knew the first eight letters of the Greek alphabet I would have made them that way. But basically, I've got eight unknowns. As long as I have four points, I would have eight equations, why? Because I get an equation for the X, an equation for the Y for each point, corresponding point. But I really because my location, locating of the points won't be precise. I would be better if I had a whole bunch more points then I would do a least square solution. And you would solve for the minimum of the magnitude of Minus b squared, the minimum of that square using least-squares, and you've known how to do that since eighth grade, I hope, ninth grade, twelfth grade, I don't, somewhere along the way. If you don't know what I'm talking about [LAUGH], go learn a lot of linear algebra before we take the rest of this class. All right, but this is the non-homogeneous way. Thinking about this set of equations we've already seen this, right? Remember when we were doing the extrinsics and we had a whole bunch of equations, two N of them, and we had am equals 0, because we didn't change it. So we can do just like we did before. You multiply this through. You divide out by the, by w, and you'll get two homogeneous equations for each point, right? Now instead of m, we're solving for h, and we have a whole bunch of terms all linear in terms of h, and it equals 0. And we have 2n of those equations where n is the number of points. And how do we solve this? Well you use that singular value decomposition just like we did last time. Or you could take that a transpose a matrix, solve for the eigenvectors. Pull out the eigenvector with the smallest eigenvalue and that was our solution. Remember we showed that it was the same thing. That's the way for solving for the homography coefficients. And by the way, that's the cool way, because you can impress people. because you can tell them to use singular value decomposition, which has so many syllables in it, they're sure you're really smart.

6 - Applying Homography

Once we have solved for the Homography, we have to be able to apply it. So what apply it means here is I might have some point in image one at some point at x, y. And using the Homography, I can pump it through, get the x prime, y prime, w prime divide through by w, and I get the new x prime, y prime that tells me the location. And I have to essentially transform that first image to the second image by moving that point to that location. And I could do that for every point in the left image, and I would have transformed it to the location of the points in the right image. By the way, we sometimes refer to this transforming of the image as warping the image. In fact, we're going to talk about how you do image warps in a little bit or s, or yeah, image warps. It's actually going to be interpolation, but that's giving away the secret. All right. So once you do that, you can take, for example, these four pictures of the Great Wall of China. And you can wave your magic computer vision wand and voila, out comes what looks like a spectacular wide angle view of the Great Wall of China. Thank you, Steve Seitz. Now you'll notice that there are some distortions here. And one of the things I want to point out to you is that there were some distortions in the original image like you see here some of the lines are bent and that's to do with distortion on some of the lenses. But in general, lines stay lines when you project them onto planes. And, that's a pretty cool picture. There are other applications of essentially mosaicing pictures together. Here's one that was used for the comment is content-based coding, which is stolen from somebody who stole it from somebody who stole it from somebody else who's now lost forever. But basically, the idea here was what we do is we have this tennis player. And you can see the tennis player's being removed. And with the way that was done is, you essentially take all of the pic, pictures and it's a camera, a TV camera that's rotating about its focal point, so you can line up the pictures. Now of course, they line up pretty well everywhere, except where the player is. And so you can use a particular method called ransack which we'll talk about, oh, I don't know, probably nine years from now. To figure out what the most likely relationship is between each of the pictures, line them up. And anywhere that the pixels are changed a lot, you say, I'm going to throw that away, subtract that out. And that removes the player that's moving around and what you get out is this nice, big background, which is as if you had this really wide field of view. Okay? And then if you wanted to have the player running around on this background, you could just play the player, moving around on this different background. Lots of people have used these types of mosaics in order to be able to show you cars traveling on a really big area. So it's taken from an airplane with a narrow field of view, but it sweeps across. And

it's basically building up this big background to do what's referred to as content-based manipulation, content-based coding.

7 - Homography Quiz

Okay, time for a quiz. Simple quiz. You're going to love this one. We said the transformation between two images taken from the same center of projection is a homography, H. How many pairs of corresponding points do I need to compute H? A, six. B, four. C, two. D, eight

8 - Homography Solution

Right, and everybody's going to say 8, because it's 8 degrees of freedom. No it's 4. It's 4 pairs points, each pair of corresponding point gives you two equations, so you'll remember that.

9 - Homographies and 3D Planes

There are two more things I want to show you. One is just some nice little mathematics to explain to you about homographies and planes, and the second is some of the cool things we can do with homographies and planes besides just, mosaics. So, let me show you a simple proof, if you will, or a demonstration, about this eight degrees of freedom being a mapping between any planes. So, let's go back to our calibration from the external world to the internal. So you remember this? Right, X, Y, Z, 1 is mapped through a 3 by 4 homogeneous matrix. Or I should say it's a matrix we're doing this homogeneously. To get out the projective similar UV 1 over there. And remember we talked about solving for all the 12 M's using lots of extrinsic points. But suppose. Just suppose, all the points in the world were lying on a plane. So you might remember from your algebra that simple equation of plane. aX plus bY plus cZ equals d. Right? So that's the equation of a plane. Basically what it says is it, it gives me the normal to the plane and where along the, that normal the plane has to be. What this equation means is that I could solve for Z in terms of X and Y and a, b, c and d, and I could plug that in. And that's what we did here. Here we removed Z, and put in its equivalent expression in terms of X, Y and the a, b, c and d which define the plane. And so, now, it's the same end matrix that we had before. But we've gotten rid of the Z. But wait. If I have only X, Y and a constant in this term for Z, I can put all the influence of this column, 02, 12, and 22 of m, I can put that in these other. Coefficients. Right? I don't actually need this column, since that's just multiplying the terms that have the XYs and the constants in them. I can put all the effect into those other three columns. So when I do that, it looks like this. Okay? And you'll notice I now have a column of zeroes here. Okay, so that's really not doing very much. I just have these three columns multiplying X, Y, and 1. All right? But, that's just a 3 by 3. That's my homography. Okay? That's why the mapping. From any plane to any plane can be done by a 3 by 3 from a location on 1 plane to a location on the image plane? And so basically between any planes. In fact a way of thinking about that is this, right? Mapping between planes is a homography. You give me four points on this plane and four points on that plane, I now know how all the other points on the plane map to each other. And whether it's between a plane in the world and an image plane, or just thinking about it between arbitrary planes, that's the warping that it takes in order to do this homography or projected projection warp. Between the planes. And mapping between planes is something we do a lot in Computer Vision. Which is why these homographies show up quite a bit. We're going to do even more sort of sophisticated way of thinking about lines and planes in the next lecture. Where we do even more about projective geometry. Got it? Cool.

10 - Image Rectification

What else can we do? Well, if I can warp between planes, arbitrary planes, that means that I can take planes that, say, are in some particular orientation, and think about what would they look like turned in a different orientation. So here we have a si, a situation where we've got a plane, that's the

side of this building, I'm assuming the building is more or less planar, and I'd like to rectify that. That is, I want to remove the slant. So you see this window here. All right, you see how if I were to extend out, these are vanishing point lines. All right, so these are not parallel, they're going to converge, you can see that here. They'll converge all the way out there. All right, but if they were, if it was frontal parallel, then they would remain as parallel lines. Well, if we could compute the relationship between this quadrilateral, which is essentially like a trapezoid, between that and a rectangle, we would be able to rectify the entire building. And in fact that would look like this. At which point you're supposed to say, ooh, that's cool. Ooh, that's cool. Very good. So here's the corrected image where now everything is rectilinear. And in fact, if you take a look at these windows. In fact notice that these are sort of narrow windows here, and wide windows here, and narrow windows there. If you want to think about that from a measurement perspective, you can see that the, the windows that are sort of two panes wide were narrower than the ones that are three panes wide. Even though clearly this is wider than this in the image, but when I rectified it I was able to measure the whole thing. Show you another example of that. So here's a picture. Suppose you want to measure something, let's say you start with on the floor, okay, so something about across what's on the floor. Well, so they have tiles, and those tiles are actually rectangles. So I could unwarp this picture so that those tiles are now squares. Because they're, let's say nine by nine inch squares, which is some number of centimeters. 9 times 2.54, it's 18 to 4 and a-half, I don't know, 23, 24 centimeters? Anyway, I can rectify that image and then I can go ahead and I can measure in that new image directly about the tiles. likewise, if I wanted to measure something on the side of this wall, and that's a planar surface also. It happens to be very skewed. I could rectify that. And here, I've taken that wall, and I've spread it back out to being rectangular. Actually, I didn't, the person I stole the slide from did. And I can now measure on that. And what kind of warp is this? What kind of transformation? Plane to a plane. That's right, it's your friend, the homography. Just to show you what I was talking about from not from quadrilaterals to rectangles, here on the left hand side I've identified four points. And you can see those four points are the four corners of a two by two tile matrix. But of course, the points aren't rectangular because of perspective foreshortening, right? But I can warp that to being a regular square or rectangle. And the rest of the image will come along for the ride.

11 - Football Example

Turns out that there are some other rectangular grids, it's really nice when somebody spray paints the rectangular grid. Here you're seeing a movie, of a American football play, in fact, this is a college football play. But you'll notice they're running around on a plane because to play football, on bumpy things is hard. All right? So since it's a plane, we can rectify that to a different plane. In particular, what about the plane that you would see if you were looking sort of overhead? Okay? So what we've done is, this is just the same frame, mapped to a different view, where we've taken these lines which converge and we've made them all parallel. All right. Now you'll notice that all the points go to the right place, except, the players of course, right, they are not on a plane. Just their feet are on the plane, so their feet go to the right place, but the players themselves are stretched out, because the players are not on the plane. But what we did is we applied the entire homography, the, sorry. We applied the homography to the entire image, so everything on the football field went to the right place, the players went to a slightly different place. But you'll notice, if you look at this, you could see exactly what's going on. Here you see two videos playing at the same time, one is the play as observed by the camera, the other one is this rectified thing, a little bit of shaking is going on here because this was done all automatically. And by the way, this is like one of the few results I'll show you that was actually done by a student of mine. When we were doing that interpreting American football plays, and this way just a pre processing step. But the, we're able to recover the projection. So if you need to save money on a blimp, this is what you do so you can see an overhead view. In fact in some sense this is easier because you see the side of the players, not just their overhead. So I probably should have patented this and made a billion dollars and then I well, didn't happen.

12 - Forward Warping

We've kept talking about warping images from one to another. That is we take an image, we pump it through is the phrase I'm using or we transform it by homography, which means we have to go and transform this image. We have to warp this image. So the question is, how do we do that? The answer is, there are two ways. The forward and the backward. And theres another word for those, the wrong way and the right way. Image warping can be thought of this way. Suppose I have an image transformation and I have some source image f of x,y and I know how that maps to x prime y prime. Or, or another way of thinking about it, I know how that maps in the other image I'm going to create some new image g and for every x prime y prime in it, or call it a and b or ij. What value should go there. And this t of x,y is the transformation that says, given x,y in one image. That value should go to x prime y prime in the second image. So forward warping works as follows. You basically send every pixel f of x,y. So I've got the first image it's at x,y. I send it to its corresponding location in the second image. So I have some pixel here, a little red dot. I know where it goes and I put that same color that was down on the first image down in the second image. And that's okay but you might ask the question, well, what happens if this pixel lands between pixels in the new image. Lets take a look at that. So. Here's a, a picture and I credit Alyosha about this. And in this picture, let's assume that the lines are exactly, you know, y equals 2, y equals 3 and x equals 14, and x equals 15, so the intersections of the lines are the precise pixels. So if I have a pixel here and I map it across. I may land somewhere between a pixel, okay? And somehow, what I have to do is, I have to take the color that came from this location and I have to distribute it among all the pixels that are around where I landed. In graphics, that's known as splatting, that's the technical term. I mean, I guess it's the idea if you threw some paint down there and all the paint had to drain out to its neighbors, where would that paint all land. And that's sort of an annoying way of doing things, and that's why I said that, that was the wrong way of doing warping.

13 - Inverse Warping

The right way is what's known as inverse warping. Given some pixel in x prime, y prime, in the new place, transform back. Figure out where it came from. So now x y is the inverse, T inverse of x prime, y prime in the first image, right? So I have to figure out where it came from. So I have a pixel on the right, I figure out where it came from in the left. Now you have a different question to ask. You no longer ask question, what happens if I get sent to a location between pixels. I now have to ask the question, okay, what happened if a pixel comes from between two pixels? Well, what you have to do is what's referred to as interpolation. Interpolation basically says, if I have some set of pixels, and I'll think of them as being at integer locations. So I have some pixels here at the corner and I wrote that as, i j, i plus 1 j, i j plus 1, and i plus 1 j plus 1, so those are the corners. If I want to find the value of something that is say an a distance in x, between the two pixels, and b distance or b percentage in y between the two pixels, what's the value? The general process of determining what the value of the function should be, between points that you know, is referred to as interpolation. Here we're back to thinking of our image as its functions a little bit, so we have a value of the function at each of those integer locations, and we need to guess the value of the function at this location that's somewhere between the points we know. That general process is referred to as interpolation. The simplest interpolation is called nearest neighbor. Nearest neighbor means, well, just like it sounds. I land some place in my grid, and I say oh, you you're the closest value. I'm just going to pretend that I'm that value. So this quadrant would all have one value. This quadrant would have another value, et cetera. It's not a very clever, not a very smooth way of doing things. A much better way, and perhaps the simplest computational way, is called bilinear interpolation. Bilinear interpolation basically says that I figure out the value in the x direction along with both the top and the bottom. And I do that by linearly interpolating between those two values, and then I would interpolate between those values again using a different proportion, in this case b. But, because these are all linear operations, you can just write it as one single linear operation, and you get this formula here. And it's very clean and very easy. There are more clever ways, by cubic, where you take, instead of just fitting linear things, you fit cubic splines between them. But,

basically, the idea is that you're, you're interpolating. And what's nice, now, is that you don't have any holes in your picture, right? You went for every pixel in the target image, figure out where it came from in your source image, interpolate that value, and stick it in the target image. So there are no holes in your warped target. By the way, it won't surprise you that Matlab and OpenCV and Octave and all of these, support this interpolation generically. in, in Matlab, you can take a look at this function called interp2, that's two-dimensional interpolation. And it will allow you to do this.

14 - End

All right. So, just to review a little bit, we talked about sort of practical application about how to make a mosaic or a panorama. We said you take a bunch of pictures from the same center of projection, same camera position. You compute the transformation between each of the images. Say the first and second, right? What's that transformation going to be? Well. Same center projection. It's going to be homography. You then use that homography to transform, say, the first image to align with the second. There'll be some overlap, so you'll have to blend them, and then you can continue to do that. So, that was for the, the cool application. But the theory, and this is really important part, is that homographies map from planes to planes. And this allows us to compute what a different image plane would look like from the same center of projection. So we can build mosaics by projecting a set of images that are all taken from the, the same camera center. And in, in general, what we're doing is we're going to be, we're working more in this projective relationship between image planes. So that ends the lesson on homographics and mosaics. That was kind of the cool part with a very gentle introduction to projective geometry. Now we're going to do a little bit more serious, or next lesson I'd say, we're going to do a little more serious conversation about projected geometry and the duality between points and lines. How often do you get to use the word duality in a, at a party? I mean, seriously. We're going to talk about the duality between points and lines, and that will help us get to relationship between sort of arbitrary cameras that you might, I don't know, want to do stereo matching between, or some other alignment between those views.

3D-L3 Projective geometry

1 - Intro

Welcome back to Intro to Computer Vision! The last time we talked about homogenous transforms, and said how they were a form of a projective transform. And we showed for a high chromographies are your best friend. They allow you to map from one plane to another and to be able to map image planes, and that allowed us to do panoramas and other kinds of cool things. Today what we're going to do is we're going to get a little heavier in the projective geometry stuff. It's actually a short lesson. And we're doing this for two reasons. Well three reasons. One is no self respecting computer vision person should not know something about projective geometry. So, you can say you know something about projective geometry and impress your your parents. More importantly. Or, or maybe significant others if you're dating a nerd. More importantly we're going to use in a little bit when we talk about the relationship between multiple views. Because projector geometry is just a very convenient way of representing the projection operator. Which after all is how images are made.

2 - Last Time

All right. So why projective geometry? I'm going to show you some of the math. All right. So, what is the geometric intuition of using homogeneous coordinates? Well basically, the idea is that we're going from a per ray in space is mapped to a point in the image. So here you see a ray that intersects the image plane at z equals 1. And the idea is that every point x, y, so every point x, y on that image plane at z equals 1, is represented by this ray. So you see this ray here, right? And it intersects the image plane at x, y, 1. But any point on this ray, sx, sy, s, would in, intersect the when

projected through the center projection, would intersect the image plane at the same location. And that's what it means to be projectively similar. Okay? And that, that's the, the relationship between, sort of, these rays and the points in the image. So, we already know about sort of going between homogeneous coordinates and regular coordinates. So in 2D, for example, we take a 2D point, x, y, to make it homogeneous, we just add a 1 to it and become a homogeneous coordinate. That last coordinate can be thought of as the thing that tells you about the scale. So when you go from homogenous coordinates to regular you just divide by that scale value, that's what's shown here. But homogeneous coordinates actually allow us to do a lot more in particular, they're going to give us natural way of thinking about lines, and this time we mean two dimensional lines. So here we have our standard equation for a two dimensional line, ax plus by plus c equals 0. And what's nice is, in homogeneous coordinates, it can be thought of as just this dot product, essentially, right? It's a, b, c transpose times x, y, 1, or I should say multiply by x, y, 1. And, when you think of the line that way, a, b and c can just be thought of as being the component of the, the unit vector in the direction of the normal between the origin and the line. And then every point on that line when you dot it with that normal, has to be a distance d away so that, so that would be minus d. But the thing to think about also is, it's saying that the normal, a b c is perpendicular to the point x y 1, or to the ray that defines x y 1. So what's this deal having to do with perpendicu, perpendicularity? Get that out one word, right? And in fact this perpendicular relationship is key to, to using projected geometry to define our, our points and lines.

3 - Point and Line Duality

So to understand this project, this perpendicular relationship, let's look at the projective space. So what does a line in the image correspond to in projective space? Well, a line really is defined by this plane of rays that's intersecting the image plane and that plane's defined, like all planes are by a normal. And this is that same normal, a, b, c. So all the rays that are perpendicular to that normal form the line, because the line is really just a plane in this projected space intersecting the image plane. Cool, right? It gets better. Oh, and we can write this very simply in the same vector notation I was showing you before. So essentially, in fact, you know, what, let's, let's say, l transpose p if I wanted to write it that way. Right? So that's in vector notation. And the thing to think about is that in projective geometry and in this homogenous coordinate system, a line is also a homogenous 3-vector. So a point is a homogenous 3-vector, but a line is also a homogenous 3-vector. In fact, this relationship that a point is a 3-vector and a line is a 3-vector is what's going to get to this duality between points and lines. And as we've said, a line is a homogeneous 3-vector, which is defined by that normal to the plane and it is perpendicular to every point p that's on that plane. And what does it mean to be perpendicular to a point? Well, in projective geometry, being perpendicular to a point means that you're actually perpendicular to the ray that defines that point. And that's what gives us some of that duality. Given these points and lines are both three vectors and we have this duality. Suppose I have two points, p1 and p2. Now remember, points are actually these rays, p1 and p2. How would you solve for the line that goes between them? Okay. So you know in algebra, if I gave you two points, you would figure out your formula for line. Well, in projective geometry, it's actually pretty easy. Since l here is just perpendicular to both p1 and p2. Well, you remember how to get a perpendicular vector given two other vectors? You just take their cross product, okay? So l is equal to p1 cross p2, that's it. No problem, okay? And l is the normal to the plane. So that plane is the line, so the way I get that line is I just cross the two points. Very easy. Let's just keep going. What is the inner section of two lines? Okay. So remember now, I've got a line l1 and l2 and remember that they're actually planes and what I'm showing you here are the lines that are made up of the planes where the planes intersect the image planes. So the l1 plane gives you this. The L2 plane gives you that, okay? Well, just a little bit of thinking here and you realize if the point p is perpendicular to l1 and point p is perpendicular to l2, okay? Then what I need is a line that is perpendicular or, or, and need a point whose ray is perpendicular to l1 and l2. All right? Well, that's easy. So just as we had before in order to find the intersection, in order to find the line that spans the two points. We took that cross product to find the line. In order to find the point that's the intersection of two lines, I just take the cross product again. So the points and lines as I said, they

form a dual in projective space and that means really given any formula, I can sort of switch back and forth between what are the points and what are the lines. So when I showed you that l transpose p equal 0. And I said, l was the line and p was the point. Well, that's just because l is for line and p is point. If this was some other language where l stood for point and p stood for line, you'd all be really confused. But basically, what it would mean is that I was thinking of 1 as the line and the other as the point. We're going to make use of this not in the next lesson, but in the one later when we talk about fundame, fundamental matrices. Where just to, to give you forward looking a little. Remember that if I have a point in a stereo image, it has to appear somewhere on a line. In the other, that's the epipolar line and vice versa. These pair of epipolar lines. So projective geometry will make it real easy to go between points in lines.

4 - Homogeneous Coordinates

So just reviewing in homogeneous coordinates, given two points, p1 and p2, if I ask you to find the line, no problem. Just take their cross product, l is p1 p2. No algebra required while doing cross-product. No thinking about lines, equations or something Mrs. Murphy taught you in eighth grade about how to go from. No, you just take the cross product. Likewise, if I have two lines a1x, b1y, c1 equals zero. A2x plus b2y plus c2 equals zero, and I want to know the intersection of those two lines. That actually took a little bit of work in Mrs. Murphy's class, right? If I gave you the equation of two lines and I told you to find their intersection, you had to think about, well, I have to sort of make them equal and solve these two sum. No, you just cross them. You know why Mrs. Murphy didn't teach you that? She didn't know. It's a long story about algebra teachers. Anyway, cool.

5 - Point and Line Quiz

Here's a quiz. How can I tell wether a point p, is on a line, L, in the image, okay? So I gave you a line and I gave you a point, how can I tell? Well, that's easy, I can just a, take the cross product and see if that's zero. Okay. B, no, no, no, no, no, you want to take the dot product and see if that's zero. C, check they're both three vectors, right, so I can add them, and check them if the magnitude of the sum is greater than 1.

6 - Point and Line Solution

Thinking about it this way, some point p, in the middle there right. That's projective ray, is in that plane. And it's therefore, perpendicular to the original line. So, basically, you would just take the dot product between p and l, and if that's zero, then that point p is on the line. So the answer is b, I think, I think that's what I said before, isn't that right? Isn't that right Megan, b? Yes. Yes. She's right, she's paying attention.

7 - Ideal Points and Lines

To finish our little tutorial on projective geometry, talk about something that's really more of a vocabulary issue. There's this notion of Ideal points and ideal lines. So, Ideal points come this way. We said that points are the intersection of a ray from the origin, hitting the image plane. Okay, fine no problem. And all the rays will hit that plane, unless what. Unless the ray is parallel to the image plane. The ray is parallel to the image plane, where does it intersect the image plane? We answered this before we said you had to go, they intersect out in infinity, remember you had to go ask somebody high in the Himalayas about why that's true? That's the second time, that joke was so good the first time I thought I'd use it a second time. Anyway, but the idea is that this notion of an ideal point is essentially a point in infinity. And the way you get a point in infinity is that you have z, the, the, the last term is x, y, 0, scale being 0, gives you a point at infinity. And if you want to think about that, what happens when you divide by zero, x and y blow up, and, and they go to infinity. So that's an ideal point, and the problem with an ideal point is that the ray that's supposed

to define it is parallel to the image plane. Well, what about a line, where the normal that defines the plane is parallel to the image plane, and that's shown here. It's a little difficult to see but here's our normal, right here. And you'll notice it has a z value of 0, okay? So it's parallel to the image plane. Well, that's actually okay, what it means is that the plane that it defines is also going to be perpendicular to the image plane, which means it's going to go right through the origin. So, the ideal line corresponds to the image that go through the origin, or the principle point. And that's typically in the middle of our images, but remember, you, your images might be offset. So an ideal point is off in infinity, an ideal line goes through the origin.

8 - 3D Projective Geometry

We've been doing mostly our projected geometry here thinking about images, so we're talking about 2D projective geometry, so our homogenous coordinates are three long. Well you can naturally extend this to 3D as well. Remember the equation of a plane AX plus BY plus CZ eq, plus D equals 0. So the homogenous coordinate is now just a four vector, wx, wy, wz, w. But of course we've already seen that right? When I, when we first introduced extrinsic coordinates and we talked about combining rotation and translation, we introduced this notion of the four vector. And so in the two dimensional case we had a duality between points and lines, because they were both represented by four dimensional vectors. In 3D, we have a duality between a plane, whose normal is a, b, c, d. Okay, that's the dot product. Is represented by a four vector and a point. So instead of points and lines like it is in 2D, it's points and plains in 3D. And the projective transforms, instead of 3 by 3 are just 4 by 4. So I wrote here that P prime equals TP, okay. Well, that's just the 4 by 4 transformations that we had seen before. So we've, since we've already seen that, we're not going to talk much more about it. Going forward, like I said, we'll talk about fundamental, fundamental matrix, we'll talk about projective geometry in the 2D case using the three vectors.

9 - End

All right, that ends today's, relatively short lesson. We've just, a flavor of projective geometry. There are entire, I want to say books, there are entire lifetimes dedicated to projective geometry, mostly in France. They've got this thing about it. But in particular, out of France, came from [FOREIGN] group and others was the application of projected geometry to computer vision. And it really revolutionized some of the, the work that was done in terms of how people thought about stereo and other types of relationship between views. What we're going to do, is we're going to discuss, we're going to use this a little bit in our next two lectures where we talk about what's called the essential matrix, and the fundamental matrix. Those are a little more intense, so hang on.

3D-L4 Essential matrix

1 - Intro

Welcome back to Introduction to Computer Vision. I'm Aaron Bobick still by the way. I normally when I introduce something I'd say, who I am, but I guess you know. Okay. So last time we talked about projective geometry and sort of this relationship between lines, points and planes and, and using sort of the projective relation and homogenous coordinates as an easy way to describe these things. Today, we'll use this a little bit to talk about the relation between two calibrated cameras. Next time, it will be uncalibrated. Just ta contextualize it again a little bit, we have these projective transforms that are these matrices. These are 3D matri, three by three matrices that operate in two-dimensions that can transform from an image in a variety of ways. For 3D we talked about it being four by four. We also talked about transformation that are called Homographies that would map between planes. This is going to become very important not in this lesson, but the next one. Because we're, we're going to be talking about recovering homographies that map between image

planes. We also introduced projective lines and we talked about that a line in an image plane was just the intersection of an, the image plane and a plane through the center of projection. That plane would be defined by a normal l and that's why both a line and a point are 3-vectors in this homogeneous or, or projected space. And we showed some really cool things, we showed how the equation of a line is represented by the normal components in this projective space. And then a little bit of some cool operations that if I gave you two points, defined in terms of projective coordinates, you could find the line that connected them by doing the cross-product. Likewise, if I gave you two lines, you could find the point of intersection by doing their cross-product. And these are some tools that we're going to make use of going forward.

2 - Stereo Correspondence

So let me motivate the problem. Remember stereo twice good. All right, if you have two views of the scene, and these are two cameras that are not necessarily going to have parallel image planes, just two views. What is the relationship between the location of a scene point in one camera and its location in the other camera. That was what we wanted to know, we wanted to know if we could know something about that relationship so when we went looking for the matches, we could use it as a constraint. Just to animate it a little bit, if we have of points, so the idea is we have a point P, out there in the scene. That point images to some location in the left image plane, center projection 1 here, and that line maps to a line in the center projection number two, and that was what was referred to as an epipolar line with respect to that point. So if you're looking for that point, it's going to have to be somehow on that line. Likewise the ray from the other center of projection maps to the corresponding epipolar line in the other image, its center projection. And those pair of epipolar lines, we showed before, that they're defined by a plane called the epipolar plane, and that plane is defined, well, the simplest way of thinking about it, is that it's defined by this base line vector. So the translational vector between the two centers of projection and this world point. Or you could think of it as being defined by the three points, P, CP2, CP1, that's the epipolar plane. So we have a different epipolar plane for each point out in the world. We then talked about what we call the epipolar constraint. Right? The idea was, if I know about my epipolar lines or I know about the epipolar planes. The epipolar constraint means that if I have a point P in the image on the left hand side. There's only a one dimensional search in the right hand image to look along. That was the epipolar line. For the parallel image planes, everything lined up, it was the same horizontal line. But in general, if I have verged cameras we showed your epipolar lines weren't going to be parallel. And here was this example where we had converged cameras and corresponding epipolar lines. It also allowed us to define a couple of terms. So the baseline was just the vector separating the two centers of projection. The epipolar plane we just talked about, given a world point and that baseline defines the plane. The epipolar lines were the intersection of the epipolar plane and each of the image planes, so that's a pair of corresponding epipolar lines. The epipole of each image was the point in the image plane that intersected the baseline. And we said sometimes you don't actually see that, in fact if we, if we go back to this pair of images here, all of these would have intersected over here. So if this image had been huge, you would have been able to see the epipole, but because it's in some sense cropped out, you can't always see the epipole.

3 - Stereo Geometry

The problem with everything that I just showed you, is it would be very difficult to turn it into code. Why do I mean that? What I mean is, I've been showing you everything in terms of geometry, right? I'm like waving my hands, and we've got points, etcetera, whatever. Computers don't do so well with, waving hands and, despite what my undergraduates say. Okay? What we need to do is we need to go from the geometry, to algebra. Because believe it or not, computers are fine for algebra as long as you know what you're doing. That is, you can convert, algebraic expressions into computations. In fact, we talked about that earlier. But, it's very hard to go from geometry to computation. And the way we're going to do that, is we're going to take the geometry and then convert that to algebra. So we're going to start off, today with a pair of calibrated cameras.

Okay, so here's my pair of calibrated cameras. I have two camera centers, Oc, and Oc prime, and they're both looking at some world point, X. And it's viewed in the two images, and it's seen here. Because these are calibrated cameras, I'm going to assume that I know the translation T, along the baseline. It. Sorry T is the vector that is the baseline right so, if I have to shift you know, two, seven, nine. To get from one camera to the other in XYZ, that's what my translation would be. And I also know R, I know the rotation matrix though, you would have to rotate one camera center to become aligned with the other camera center. That's what it means to be calibrated. So we write this, this way, okay? It says from geometry to algebra, we're not quite there but, but we're almost there. But basically what we're saying is, the location of the point X in the prime frame, can be expressed by rotating X as appearing in the non prime frame and then translating it by T. That's just the equations we had before. I know this is not the homogenous version, it's okay, in a minute we'll make everything beautiful.

4 - Aside 1

Before we make everything beautiful a couple of reminders. Actually this is the, the first reminder. Or an aside, I should say. And it's about cross product. The second one's about cross product, too, but. So the first one is to remember that a cross product. So a cross b gives you a new vector, c. And that vector c has the property of being perpendicular to both a and b. So it's in the direction that's perpendicular. And you may also remember that the magnitude of that vector is equal to the sine of the angles between them. Which should make you realize right away that if I take any vector a across itself, that's going to be zero. Because the angle between a vector and itself is zero, the sine is zero. So, a cross a is 0 no matter what a is. Okay? So a cross b gives you c, c being perpendicular to a and b. And a cross a is equal to 0 no matter what.

5 - From Geometry to Algebra

So here we start our little mathematical foray down into making it from geometry to algebra. So I just wrote what we had before. The X prime, the position or location of X in the prime frame can just be expressed just having rotated the X frame by the R amount and then translating it by T. All right? Great. Let's just cross both sides of that equation by T. So we have T cross X prime on the left, but on the right, we have T cross and it's just, because of distributive property it's T cross RX plus T cross T. All right? And by the way, I, I wrote here that this is normal to the plane. Remember that T cross X prime is going to be something that's perpendicular to both of them, so that would be perpendicular to the epipolar plane that, that we're showing here. We'll, we'll maybe make use of that in a minute, but I just want you to realize that. More importantly, what you should see is that we've got T cross T here. What's T cross T? Not very much. Otherwise, known as zero. Okay? So that just means that T cross X prime is equal to T cross RX, got it? Okay. Now let's take both sides of that equation and I'm going to dot it with X prime. So we got X prime dot T cross X prime equals X prime dot T cross RX. I sure hope you're reading this, because listening to me sounds ridiculous. Okay? Well, let's take a look at this left-hand side. Okay? T cross X prime gives us what? Well, it gives us a vector that's perpendicular to the plane containing T and X prime. So it's perpendicular to T and it's perpendicular to X prime. And then I dotted it with X prime, so what am I going to get? Zero, right? because it's perpendicular and if everything goes well, there it is in yellow. Zero is equal to X prime dot T times RX. Okay? So far so good? Good.

6 - Aside 2

Time for aside number two. This one's kind of cool, actually. It's more about cross products. So you might remember, there's a simple formula for how you can get, a cross product. You basically do IJK, and then you do a1, b1, c1, you, you do your first vector, second vector, and then you do essentially the determinant, and that gave you the vector. Remember that? Remember that? Remember that? Okay. Maybe you don't remember that. That was the formula for cross product. If you had been really smart, you would say, you know, I can just write this as a matrix multiplication,

okay. So I just build this little matrix here from my a. Okay, and I take my vector b here, and so that says that the first component of c is going to be what? Well, it's going to be minus a3 b2 plus a2 b3, and that is the first component of the cross product of a and b. Same thing for the second component and the third component. So we can just write cross product as matrix multiplication. So I'm going to define a little operator, and this little operator is written here. This x, that means cross product okay? So when I do this little bracket a x cross, that means that I'm going to substitute in this matrix, which is sort of the cross product in matrix of a. So if I want to do a cross b I would just multiply this matrix times b. So the notation as I wrote here is a cross b equals and just this bracket and, and here the font's fixed. It's just this bracket a x times b. Something that we're going to make use of, not this lecture but next lecture is and I'll, I'll leave, you know I love it, I used to love it when professors said this, the proof is left to the reader. Okay, but you can actually do this. Figure out what the rank of this matrix is, and if you come out with any number other than 2, try again, because the rank of this matrix is 2, okay. So it's a three by three matrix, but the rank of the matrix is 2. Now we're going to make use of that later, because as you may remember, if you multiply matrices together, ranks can only get smaller. I multiply a rank 2 matrix by some other matrix, I've got a rank 2 situation.

7 - Essential Matrix

Back to where we were. Remember this equation, 0 is equal to X'.(T x RX)? We just rewrote that again. So, now I can get rid of that whole cross product thing, right, and put that cool, little funky cross product notation in there, right, right, all right? So, let me do a substitution. I'm going to let E. Be just this, T cross R, okay, so it's just this substitution in here. Well, as soon as you do that, you realize, I have this really beautiful elegant expression that says X' prime transpose EX equals 0. And E is what's called the essential matrix. And what's cool about the essential matrix is that, it relates the point X and the same point, but it described in the other camera frame. And note that these are world points. In these frames for these calibrated systems. In a little bit we'll talk about the relationship between, world points and the image points. And it's a little bit closer than you might think, because of the projective geometry. In fact, one way of thinking about this really easily is this is equal to 0, right? So what if I multiplied X by some value A? Would the equation still be true? Of course it would. So not only is it true for the point out here, as expressed in this coordinate frame, it's true for every point along that ray, expressed in that coordinate frame. Likewise for every point along that ray. Expressed in that coordinate frame, ooh. That means, it also talks about image points, but we'll get there, all right? One last thing, and this is going to, you're not going to totally get that this time, but that's okay, you're going to get it next time, all right? We said that this works for sort of, any point along that ray. And, so you can think of that as the homogenous representation of all the points along that array. So this, so our E is our 3 by 3 here, times this 3 by 1, so this is another vector here. All right, we'll just call that L, okay. Why am I calling it L? What does L stand for? Well in some weird language, we said it stood for point, but in our language, it stands for line. So remember that X transpose L, or P transpose L, or L transpose P was the definition of a line in projective geometry. So what this means is, if I knew where some point was in the image. So I know what ray it's along in the image. So I know where it is in the image. I could put that point into this equation. And I would have this L, which defines for me a line, in the other frame. Just X prime transpose L equals 0, that was the, the definition of a line in our projective geometry. So in other words, if I gave you a point in one image of a calibrated set of cameras, there's a line in the other image that that point must lie on. What's that line called? That's the epipolar line. But notice, that everything on this side of the slide is algebra, right? Forget all the geometry stuff. So what we've done is we've now converted the epipolar constraint into an algebraic expression. Thank you very much. No. That's what we're going to use, okay? Great.

8 - Parallel Image Plane Quiz

All right, so, you ready? This is actually a good quiz, you know why? Because the quiz is going to make us do some work. So that was fine, what I just showed you for some converged cameras.

What about if the image planes are parallel? What happens? A, well, that's a degenerate case. And we're going to see that in just a minute. B, that's fine, it's no problem. It just means that R is the identity matrix and the math just works. C, I have no idea. Okay, that's usually a bad answer to put on a final, just saying.

9 - Parallel Image Plane Solution

By the way, then and so C is not the answer. I have some idea. So the answer, the rotation matrix is just the identity. So let's work through this a little bit.

10 - Essential Matrix Example Part 1

Let's do the essential matrix computation for the example of parallel cameras. Okay? So, first, what is, so, so, by the way, you recognize this picture? Right, this is the picture of our parallel image planes when we were doing the stereo the derivation of the relationship between disparity and depth. So I have two parallel image planes, nothing is rotated, oh, nothing's rotated, so what's the rotation matrix? Well, it's just the identity. Great. All right, what's T, well, what was T? T was the translation between the two camera centers. Well, in the geometry that we've defined here, the camera center has just been translated an amount B in the x direction, or negative B in the other x direction, doesn't matter, but we'll just say it's minus B, zero, zero, okay? So, T is just minus B, all right? Well, that means I can now compute my essential matrix, because my essential matrix defined before was this weird cross product thing of T times R, R is the identity, so we just plug T into the cross product, little formula I gave you, and it looks like that. So that's our essential matrix.

11 - Essential Matrix Example Part 2

Let's make use of that matrix, all right? What does the essential matrix constraints say? Essential matrix constraints says, that p prime times z times p equals 0. Where p prime and p are the locations of the points, one in the prime frame and one in the regular frame. All right. So, where are our points P? Well, it's in some location, cap X, cap Y, cap Z. Okay. Which is also, in the other frame, cap X prime, cap Y prime. Also Z, because the Zs are the same because our planes are parallel, right? Right. So here are our points written. And the first time I wrote them as we said before, p is XYZ, P prime is X prime, Y prime, and the same Z. But, kind of written kind of cool over here, right? We assume that under normal projection, little x is f times big X over big Z. So I'm just writing big X as Z times little x divided by f. Okay? And that's how I entered in these things here. Same thing with the primes over there. Okay? Let's put those points in our, essential matrix equation. All right. So, I've just inserted them here using our e. And now I've done something a little bit sneaky again. So here I have p prime e p, but you'll notice I've now divided my points all the way through by the, the amount z over f. A constant, right? Z is whatever, s, something, I don't know what the value is, but it's some constant value, and f, I don't, might not know what that is, or I might, some value. And I just divided all the points through by that. Now, why am I allowed to do that? See, in class, everybody yell out the answer. But over the ether. You guys can't say, because it equals 0, dummy. All right? So I can multiply anything I want by a constant and it stays equal to 0, and I just wrote it this way to make it easier to see the equations that are about to happen. So I just divided by z over f. Copy that equation over again, multiply out the first one, so now we have x prime, y prime, f dotted with, or transposed, of 0 bf, minus By equals 0. Multiply it more again, we get Bfy prime equals Bfy. B is a constant, f is a constant of divide through, and it just says y equals y prime. y equals y prime? What does that mean? Oh. That means if I have some point in the image location x y in the regular frame, the line y prime equals y is the line, that this point has to be on. And think about what this is, so a, a points at x y so it's the sum x location and it's y, let's suppose y is 17. The equation y prime equals 17 is what. That's a horizontal line at height 17. So what this says is, our epipolar constraint for parallel cameras, right, is that given a point in the image, the corresponding point is on the same horizontal line. Except last time, I just had to make you believe me. In fact, you didn't even ask the question, because it's very quiet in here, but, it made common

sense that if these cameras were lined up it should be at the same line. Well, common sense doesn't get you very far in mathematics. Here is the derivation, if you will, that from parallel planes you can lined up this way, that your epipolar lines are horizontal. 'Kay? So the answer to that previous quiz of, does this work for parallel planes? It works beautifully, actually. The rotation matrix is just the identity and it derives out that your epipolar lines are in this cast the horizon, the parallel horizontal lines at the same height. Cool huh?

12 - End

Okay, so that's called the essential matrix. And it's maps between calibrated systems. And what was really important about this whole operation is that we converted the geometric constraint that we had talked about back in the stereo days where we were talking about epipolar planes and, and looking for them. We converted a geometric constraint into an algebraic constraint. All right. But, you ask, suppose I have uncalibrated cameras? Geometrically, thinking about, if you gave me enough points that you told me, this point corresponds to that and this point correspond to that. It feels like I should be able to figure out how the epipolar lines are constrained. So what I need to do is to somehow figure out the algebraic relation between images, given those correspondences, and that algebraic relation will give us the relation between a point in one image and its epipolar line in the ima, in the other image, if you gave me enough correspondences. That's what's called a fundamental matrix, and that's what we'll do in the next lesson.

3D-L5 Fundamental matrix

1 - Intro

Welcome back to Introduction of Computer Vision. Today we're going to wrap up our unit on essentially calibration and stereo and we're going to talk about the fundamental matrix. Last time, we developed the relationship between two views of two calibrated cameras where I actually knew the rotation and translation between them. And what was cool is we derived, we essentially went with what the geometric constraint was that whole epi-polar constraint that we had shown very early on. And we derived from the geometric constraint, an algebraic constraint. And in particular we defined something called the essential matrix which related between two views or I should say, world points of two calibrated cameras. So the same point in the scene, the two different views of it, what the relationship was between them. But you might ask, suppose I don't have a calibrated camera. All right, well look, let's think about the intuition. If you gave me two images, and you gave me enough points to tell me that this point is this one, this one, this one, this one, gave me enough corresponding pairs. It feels like you should be able to figure out, say the epipolar lines, that is how the, how the lines map across and what the, what the correspondence should be for all the other points. When I say correspondence, where, where they can be along the epipolar line depending upon their depth. Well, today what we're going to do is we're going to make that intuition real. We'll, we'll do the mathematics of this relationship between two uncalibrated points.

2 - Weak Calibration

Let's talk about it's going to call it weak calibration, and what I mean by that is we're just assuming that we've got two projected cameras. So that they're going to behave like projected cameras, but in particular, we don't know anything about focal length or, or their offsets. But we are going to assume, for example, that we don't have radial distortion. We're going to assume that we don't have a non-uniform stretch of the image. So, but what it's say weekly calibrated that's all we mean. We correct for that. So the main idea is that we're going to estimate the epipolar geometry from a set of corresponding points between uncalibrated camera images. And we'll probably use a large set, so it's going to be somewhat redundant in order to make this thing more robust with respect to noise. All right, so let's get started. So before, from our all our projection stuff, you'll remember that we

had this projection that went from a world coordinate through our extrinsics to our intrinsics to a projective representation of the image coordinate right and so to get the non-homogenous you had to divide through. So it's written here like the it's a scale w times the image point is equal to the world point times our extrinsics times our intrinsics. So phi is just as we did before, we're going to use a three by four version here since K will be a three by three, where the left hand side is that rotation matrix and the right hand side is the transformed translation and, and you'll notice this slightly weird form written here of it's r times t. And all that's saying is that if you know the translation in one frame you might have to translate, you might have to express in the other frame depending exactly on how you're, you're doing this. Don't, don't worry about that detail, okay? So that's phi, our extrinsic parameter matrix. And K if you remember, that's our intrinsic calibration matrix. And the K that I'm showing you here has a focal length and an x scale and a y scale and of course as we said before they're multiplied together. You really only need two numbers you've got f divided by sx and f divided by sy, so they, they always appear in multiples like that. But we have a scale in both directions. We have the offsets. But what we don't have is, we don't have any skew, that is that this value here is 0. That makes this math a little bit easier and as I said before in general for modern cameras skew just isn't an issue. All right. So I'm going to write that this way. The point in the image, and this is a homogeneous point, is a point in the world times our, multiplied by the extrinsic matrix, multiplied by the intrinsic matrix. But this, of course, is just the world point put through the extrinsics, that is the location of the point. It's a 3D point, but it's in the camera frame. Remember that the extrinsic matrix maps us from some arbitrary world frame into the 3D frame of the, of the camera. Okay, so that, we're going to call that pc. And so we can write down this very simple relationship that K, the intrinsics here, maps us from the point in the camera coordinate system to the homogeneous point in the image, okay? So p in the image is equal to K intrinsic times p in the world. It's the point in the world, but expressed in the camera frame.

3 - Uncalibrated Case

Now, this is as if I had fi and I had k, etc. But I said that this was uncalibrated right? Right. So, lets suppose we have a given camera, Pim equals K int times Pc. All right, now see that k, that k intrinsics? That's a nice, assuming we don't have zeroes sprinkled through here on the diagonal, that's a nice invertible matrix. So I can invert that k and that's what's shown here. Okay, and what this says is that I can go from the image back to the point out in the camera frame. Now this should bother you a little bit, just a little bit because it says this, this equation's fine okay. We derived this a long time ago. We said if you gave me the 3D point in the camera's frame, I knew where it landed in the image. But this second equation is saying that I can go backwards. You can tell me where I am in the three, in the image, and I'll be able to tell you where I am in the world. And that seems wrong because that whole projection operator. And then you say, aha! This is not a regular three d point, this is a homogenous point. And remember, where the projected geometry stuff we did a couple times ago? That's anywhere along the ray. And that's correct, right? If you tell me where I am in the image if you just tell me the basic camera intrinsic geometry I now know the ray in space that, that point came from. And that's all projective geometry the homogeneous coordinate is telling me, that if you give me the point in the image and the intrinsics, I can tell you the ray in space. Which is the same as a homogeneous three two the, the homogeneous coordinate in the image. Okay? That's why we can go from an image, to what feels like a world point, but is really just the ray. So, now I can take that equation and I can say, well, suppose I've got two cameras, typically called the left one and the right one. Or the up one and the down one, or the forward one and the back, or just the main and the prime. Two cameras, okay? Here written left and right to be convenient. Okay? Now we have a different internal calibration matrix per camera. Right? They, the two cameras may be the same kind of lens, the same everything or they may be quite different. All I need to think about is that, I, there exists an intrinsics calibration matrix for each one. And by the way, this is uncalibrated, right? Right. So I don't know what it is, I'm just assuming there is one, and that it looks like that one that I showed you before. I don't know what the values of those parameters are. Okay? All right. Now comes the magic, all right? Just repeated both equations here. This says the location of that point in the right frame in terms of the ray. This says in terms of the

left in terms of the ray. Okay? But we know something about the relationship between the, a point in the right and a point in the left if this were calibrated. Right? What would I, what did I know there? Well that was last time. I know there's this thing called the essential matrix that would relate the ray to the point from the right frame to the ray to the point in the left frame. And the relation between them is called the essential matrix. Now, I don't know what the essential matrix is. Why? Because I'm uncalibrated. But I know there would exist one. Okay? So if I assume for a moment there exists one, then I can rewrite this equation like that, okay? So all I did was substitute for the point ray in, in the right frame which was from here and I pulled that down into here, okay? And do the same thing on the left.

4 - Matrix Multiplication

Copying that over. Well, this is just matrix multiplication. I hope you all know that a b bracket transpose is just b transpose a transpose. If not, trust me, all right? So I can regroup using the associative property here. Such that here I have the transpose and that's where this comes from, from, from that. Okay. And then what I can do is I can combine this whole thing, the K inverse intrinsics from the right transpose times Z times K inverse of the intrinsics on the left. Remember I don't know what they are but I just know that they exist. And I'm going to call that whole thing F. And when I do that I end up with this beautiful, beautiful, beautiful equation. It just says that p transpose of the image point on the right, 'kay, so this is from the image now, times F times p in the image on the left equals zero. Okay? And this is a relationship between image points, which I can just write this way a p transpose F p prime equals zero. P being in one frame, p prime being in the other frame. That is the fundamental matrix constraint. That's what's going to allow us to solve for the relationship between one view and another given enough points that correspond. Enough p and p primes that correspond, we're going to be able to solve for F. And once we do that we know what relates the two images, or this can just be written as p transpose F p prime equals 0. P in one frame, p prime in the other frame it's the relationship between the points. Now, with this equation, I'm going to be able to do a lot of things. In a minute, I'm going to show you how we can solve for F. That is, if you give me enough points p and p prime that correspond, I'll be able to solve for F. But the other thing is F is actually very powerful in terms of describing how the epipolar geometry works.

5 - Properties of the Fundamental Matrix 1

All right. If you had actually been here while we were recording this, you would have seen that I just walked around the room 17 times trying to get my act together. because I started to talk about this and my, my tongue just [SOUND]. But, but here goes, okay? Couple things, all right? So, this relationship here, that's the relationship that happens algebraically between corresponding points. p and p prime are the image points of the same point outside in the world. Now, we remember from our epipolar geometry, so like that's over here, that if I've got some point let's say it's right here in the image, I know that that's on this ray somewhere and therefore that point must be on that epipolar line. That was the geometric constraint that we had derived earlier. So the p transpose Fp prime equals 0 is the algebraic one. The drawings that the epipolar lines is the geometric one. But clearly, they relate. So you'll remember from our projective geometry little tutorial that we said that the equation of a line, right, could be written a couple ways. I'm going to write it this way, p transpose l equals 0. All right? Remember l was the, that normal to the plane, and all the points p had to be on that in that plane, and would be perpendicular to it. So notice, here we have p transpose, here we have this p transpose, here we have this l. Here we have that l, that it here we have that Fp prime, okay? These equal 0's align, that's the constraint that has to equal 0. What this means and that's what's written here is that this l, which is Fp prime is actually the epipolar line in the p image, okay. So remember it's mol, I'm going to to erase this, right? It's going to be multiplying a point in the p image. So l equals Fp prime is the epipolar line in the p image associated with the point p prime. Also just reversing this, that means that F transposed p is the epipolar line in the prime image, the p prime image associated with the point p. All right? So what

the fundamental matrix does is it gives us the epipolar line constraint between two views. So if I know what F is I can compute the epipolar lines. In fact, you can just for a given point I just take F transpose p and that's the epipolar line in the other image. And that makes sense because the F is what we said deri, takes all of the, goes through the intrinsics of one, through the extrinsics, the intrinsics of another. And so the only thing I don't know for some point that I'm seeing in the, in my left image is well, since I don't know its depth along that ray, I, it has to be along this line, that was the geometry. So, the fundamental matrix equation reduces that geometric constraint to an algebraic one. The great thing about video's you can go back and take a look at this. We'll also, in the class notes, there'll be some pointers to some texts where you can look at it and learn it from people who are way more articulate than I am. But that's relationship between the fundamental matrix. And the epipolar lines. Go back and look again. Do it again. Okay, so now you're here and now you're really got it, you understand it all, so you should really teach this class but I'm stuck here on this side of the screen, you're stuck there so I'll keep going, okay? Great.

6 - Properties of the Fundamental Matrix 2

All right, now comes something cool, again a little bit complicated. We know that p prime is on the epipolar line, corresponding to p in the other image, Okay. That's what this equation says. If, if, if this equation holds. But suppose p prime was on the epipolar line,. In the prime image for every point, P, in the original image. You might say, well, how could that be? How could that be? Thank you, Megan. Well the way that could be is, suppose Fp' equals zero. Well, in order for a point to be on every epipolar line well there is only one point that's on every epipolar line. That's the epi pole. So the way you find the epipole given the fundamental matrix. Is you solve this equation. And remember this two epipoles. There's the epipole in the original frame and the epipole in the prime frame. And so you just solve each of those equations. Fp prime equals zero. Or F transpose p equals zero. And that identifies for you the epipoles in the image. So this F matrix is very powerful. It tells you everything you need to know or almost everything you need to know. About the way the points relate to another other in terms of their projective geometry, and that lets you do things like find the epipolar lines and the epipoles. Pretty cool.

7 - Properties of the Fundamental Matrix 3

One last thing about the fundamental matrix and we'll look more at this later. The fundamental matrix is actually a three by three, right, because this is the, homogeneous coordinate of an image point, so it's a three vector, okay. But I will tell you we're going to show this in a little bit, that the F matrix is actually singular. And the reason for that is if it wasn't singular, it would map between points and points, but, in fact, it maps between points and lines. So it maps from a 2-D point to a 1-D line, and that's what, we sort of requires that it be singular, we'll do a simple algebraic proof of it later. But it makes sense, because when you give me a fundamental matrix and I give you a point, you can't tell me the point over here, all you can tell me is the line. So we're go from something that has zero degrees from a single point to something that has another degree of freedom, and that's because the rank of the fundamental matrix is only two and not three. Like I said, we'll show that to you in just a minute.

8 - Fundamental Matrix

So now that we have the Fundamental matrix, well, so what? Well as we said, it relates between the pixel coordinates in the two views. And it's much more general then the in, then the essential matrix, because we've removed the need to know the intrinsic parameters. If you told me the intrinsics, okay, then I could figure out the rays that correspond. And if I know the rays that correspond in, in the, the 3D camera coordinates, I can compute the essential matrix. I could figure out the real rotation of translation. But if I have two arbitrary cameras that aren't even necessarily the same, the fundamental matrix allows me to relate them without worrying about the intrinsics or the extrinsics. So the bottom line is if we estimate the fundamental matrix from the

correspondences, then we can reconstruct our epipolar geometry without having to worry at all about the intrinsics or extrinsics of the camera. And that's what's done here. Here you see two pictures. I think this is also courtesy of stealing from Andrew's instrument. And you have some pixels on the left and some pixels on the right and then they were put into correspondence. The idea is I gave you corresponding points, and now the lines that are drawn on here, those are the corresponding epipolar lines. So, in fact if you take a look real quick. So you'll notice that, you know, here's a point right on the bottom edge of that window, and this line lands right on the bottom edge of that window. So these are corresponding epipolar lines. Any point that's along this line, is also going to be along that line. Let's see, does that really work. Let's see, it goes across this chimney thing, yep there it is right there. Okay, so that's finding the epipolar lines by giving the correspondences through solving for the fundamental matrix.

9 - Finding the Fundamental Matrix - Example

So let's work a simple example, all right, of finding the fundamental matrix. And bunch of them out there, one that I'm going to show here is one that's a little bit different, because it's not as sort of trivial. Here's the one of the, the two side-by-side stereo views. We'll do them a little different. And remember, a fundamental matrix gives a corresponce between any two views. It doesn't matter where the cameras are taken, okay? So, for example, my images could look like this. And you might say whoa. Okay, you might not say whoa because that would be only if you were born in the late 60s, or maybe early 60s. But here I have a picture and the white lines drawn on it are meant to be sort of nominal epipolar lines, I took a picture and then the camera moves forward, and then I take another picture. Can I do a fundamental matrix for that? Of course I can. In fact that's what this geometry is meaning to show here, it's a, it's a little hard to see, all right. What I have here is that I took one picture and my camera center was there. And then I take another picture and my camera center is moved forward and if you connect those and slice the image, those are the epipoles. So all the epipolar lines, have to go through the epipole and that's what this is showing you, right. The epipole is right in the middle of my picture. It's not exactly the middle, it's off to, to the top, it's actually to the side a little bit, I'll show you in a minute. But all the epipolar lines have to go through the epipole, and that's another way of saying is. You know, just think about that intuitively, if I move straight ahead and I'm looking straight, the points move along that diagonal out from the center. So that means that if you give me one point in, in the first image, it's going to be somewhere along that line in the second image. And how far it's along that line will be a function of its depth, okay? Because remember, the amount that the points move, between one image and the next, function of depth, going way back. Remember disparity, the closer something is, the more it'll be disparate. Something that's very far away like the moon doesn't change much at all. Okay? So they're going to move along the epipolar lines. So we're going to use this as an example because it'll be very clear whether you got the epi polar lines right or you got the epi polar lines wrong

10 - Computing F from Correspondences

So how do we compute the fundamental matrix from corresponding points. All right, so here I wrote the fundamental matrix equation. And you notice that each point gives us one equation. And that's written here like this. So u prime, v prime, one. That's the projective similarity of the point in the prime, times the fundamental matrix, times uv 1, that's the other point in the original image, equals 0, okay? That's a single equation per point, all right? So, copy it here and I can just multiply this whole thing out, and I still have my one equation per point. And, on the left-hand side. These are my knowns right, so I take the u primes and the u's or the v primes or the v prime and the u or just this individual etc. And here are my unknowns right, the unknowns are the elements of the f matrix okay. So one point, one equation. Well, I just collect n of these. Right? And it's however many number of points that I have. And then I'm going to solve the thing.

11 - The Infamous Eight Point Algorithm

So how do we do that? Well, there was originally something called the eight-point algorithm. It, it was done by, I think it was done by Christopher Hagens and for awhile, it got sort of a bad rep, because people thought it was a very unstable and that's why I put down that it was they infamous eight-point algorithm. Now you might say, well, why can I do with just eight equations, I've got nine unknowns. Well, remember p, f, p prime equals zero, I can scale the f any way I want. Which means, for example, I could set that last value to a one. Not a very good way of doing things, but it does mean that if you gave me just eight points, I'd have equations and because I set the last one to one, I would have eight unknowns and I'd be able to solve it. If I have more than eight equations, right? So then I've got this, you know, suppose I've got n, so I've got n equations with nine values that are homogeneous. Have we seen that before? Yes. Remember when we solved very early on, when we were trying to solve for the total calibration matrix? Between the world and the image and we had the M matrix. So we said, well, there's 12 elements in here, but there's not really 12. Right? There was really only 11, but if we used a whole bunch of points we talked about, you could set 1 element M 1, but that wasn't such a good way. What was a good way the SVD, remember you, you take, you've got this A times M equals 0 or in this case, A times f equals 0. You've take a A transpose A. Take the singular value decomposition or you just find the eigenvectors and you pulled out the eigenvector with the smallest eigenvalue. We've actually seen this now three times. I did it explicitly for you in the very first example, so you can go take a look at that. But that's how you solve it. And by the way, you would typically use more points than just eight or nine or you use however many points you have. So you solve this and what happens? Well, your point matches aren't exact, are they? No, so you solve this. Get an F. With an F, I can draw some epipolar lines. Right? And if I use an, and these numbers were meant to represent coming from that picture, I would see this. Okay? And that's technically called a mess, all right? Because you'll notice that the epipolar lines, they're not all intersecting at that single point, are they? No. Why not? Well, I'll tell you.

12 - Rank of F

So here I have, the same system I had before. Two camera centers. Some point out there. And these are my two image planes, okay? Now, going your way back machine. Well not so far back. Just a few lectures back. How do you map, from one plane, to another plane? Well I told you before that if you have four points, on one plane and four points on another plane. And you knew their mapping, you would then have the total mapping between the two planes, and that was called what? That was called a homography, all right. It took four points to get a homography, from one side. From one plane to the next, all right. So we're going to pretend for a moment, that we know the homography. And just call it H that goes from the plane P to the plane that has P prime in it, okay so we're going to pretend that we know that. All right, and the other thing we're going to do, is we're going to say for some point P here, l prime is going to be the fp polar line. In the prime frame, that goes through this epipole e prime. Okay? So that's the two things I've set up, I'm going to pretend that I know my homography, and I'm going to also say that that's the epipolar line, all right? Now we do a little math. Okay so, l is that epipolar line. So,. How do I find this line, well remember, this is a line that goes between two points. If you remember, in projective geometry, if I have two points, and I want to find the line that goes through them, what do I do? I just take the cross product. That was one of the cool things about the projected, geometry. You gave me two, points. Take the cross product, you get the line, gimme two lines, take the cross product, you get the point that inter, that's the intersection between them. So, here I just wrote that l prime is just going to be the cross product e prime and P prime. But wait, we said, that there's a homography H. Okay, that maps between P prime and P. So I'm just going to substitute H times P for P prime. Okay, no problem. Then I'm going to use something else you learned actually just last time all right. We said that, a cross product can be written as this little matrix operator. Right, right.okay great, so that says. Let's erase that. That l prime is just equal to this cross product operator on e prime times H the hymography times p right, right. But now, I said l prime is the epipolar line in the prime

frame for the point P. That's what this is. Right? This is epipolar line for this point P. All right? Well. We just learned that l prime is just Fp. Right? F is the fundamental matrix times P. We learned this, like, five slides ago. That was the one that we repeat, repeated, like, 17 times. Well. Okay, l prime is equal to this nonsense over here on the left, and it's equal to F P on the right. That means that this nonsense on the left, e prime cross product matrix times H, is equal to the F matrix. Well, remembering a little bit from your linear algebra, that means that the rank. Of this thing, is going to be the rank of this thing. And this, is rank two. Told you this would come up before. Remember I've said to you, and this was an exercise we left for the reader, that this little cross product matrix operator where we took the A and we distributed it amongst the three by three, multiplied it by B for A cross B. I said one of the things that we're going to need later. See I wasn't lying, is that the rank of that cross-product operator is two. So that means, that the rank of F, which is just this matrix times H, is also two. F is not full rank. But before when I was solving for the elements of F, I didn't enforce that in any way. It's a three by three. And whether if you use the svd or whatever you do, it's not going to make it become a, a two rank matrix even though it's a three by three. So we have to fix our F to be a rank two matrix. So, how do we do that? Fortunately that's really pretty easy

13 - Fix the Linear Solution

It's pretty simple to fix that linear solution that we did before. So do what we said before, solve for F. And you can do SVD where that minimum eigenvector or you can use the linear thing where you set it to 1, although I don't like that method. Okay, you solve for an F. Then what you do is you decompose your F using SVD again. This is another SVD, another singular value decomposition. And you're going to get out three terms, all right. Remember SVD gives you UVD transpose, V is the lower-dimensional orthogonal matrix of all unit vectors that are orthogonal with each other. U is also orthogonal but of the full dimension. Now, what did D stand for? No, not dalmatian, not dysfunctional, despite what you might think. D stood for diagonal, okay? So it breaks down, it gives you a diagonal matrix. Now if F is really a full ranked three by three matrix, D is going to come out as a diagonal matrix with three non-zero elements. And D, normal SVD, is produced in a decreasing magnitude, okay? So it goes from largest to smallest. So what we do is, we take the D that comes out,. And I just wrote that as rst here, okay? And t is smaller than s which is smaller than r. And what we're going to do is we're going to create a new D, D hat. And just set that less, that least value to 0. Because it was supposed to be 0, because that would be a rank 2 F would have a r, s, and 0. So we make it 0, great. Once we've done that we can build a new F, which I'll call F hat, using the same U that we got the first time. The same V that we got the first time, but we substitute in this new D, right, D hat. Which is the one that has only got two diagonal elements. And that gives us a new fundamental matrix that has just rank 2. When I apply that to the example that I showed you before, you get epipolar lines that look like this. Way, way better. In fact, you can see here the camera was not moving perfectly straight. It was moving off to the side just a little bit in that direction, and here is the epipole, right? All the epipolar lines go right through the epipole, so we've got the good F, and all is happy in Oz.

14 - Stereo Image Rectification

There are some cool things that you can do when you find the fundamental matrices that relate different views to each other, okay? Because essentially, you can figure out which points correspond to which other points. One, which is really kind of interesting, is, suppose I've got a pair of views of a scene, and what I'd like to do is do some stereo. And I know that if I solve for the fundamental matrix, I could, for every point, I could find the epipole that's along and search along that. But of course, every point lands on different epipolar line. What you can do is what's called image rectification or stereo rectification. Where you take your original image and be, if you find the fundamental matrix, you can essentially unwarp, maybe that's not the right word. Just rectify is the right word. So, it's as if the two images were taken with parallel image planes that are aligned. And why is that good? Because it makes me happy. No, it's good because you can get publi, no, it's good because then it means your stereo solution now is just moving along the, the horizontal line.

So your search is not just 1D, because it was 1D before. But it's a particular known one dimensional search. Namely, every point here corresponds somewhere along the corresponding scan line of the other image. And that was done nicely comes from this paper here shown in 1999, showing you a real example that they did that comes from their paper. Here you see two images taken from radically different views of this very weird sculpture. The lines that are drawn on there are the epipolar lines, and you'll notice they're skewed, and they they go off this way. All right, after you do the stereo rectification by finding the fundamental matrix, you get horizontal epipolar lines, which was the whole point, and allows you to do the stereo algorithm.

15 - Applications

There's some more cool applications. There's a whole bunch of work in building up structures from multiple images taken of that same structure. And there's a system and a whole bunch of work that came out of the folks at Microsoft Research and also University of Washington. Here's an example published at SIGGRAPH. And they built this system called Photosynth, and what you can see here are a bunch of pictures that were taken of Notre Dame. And, what these little things down here are meant to show, are the locations of the image with respect to the structure. And that has to do with relating between the views that you have things. When you do that, you can do some really cool things. They basically launched a site, this whole photo tourism, this idea of that being able to put in your images and this is their Photosynth. And I encourage you to check it out if, if, if, when you're using this we still use computers. In fact, here's a really cool demonstration done by Agarwal in, back in 2009, isn't it. So what you're seeing is, there's a whole bunch of these little dots, these, those are camera view points of the Colosseum, in Rome. You should go see that, see the, I mean, this is nice, see the real one, it's really cool. And this is just, flying you around the reconstruction of all these points based upon the solution to where all these cameras were. And this all relates to, first sun, solving fundamental matrix issues and then getting the full 3D calibration. I don't want to claim that you can do this with just the fundamental matrix, because in fact you eventually end up getting full 3D reconstruction here. But the idea is you have to know how the, the points correspond.

16 - End

Summary slides, I love summary slides. For 2-views and we learned at the beginning, there's a geometric relationship that defines the relation between rays in one view and rays in the other view. And we were referring to that about the epipolar geometry. These relations can be captured algebraically besides geometrically. For the calibrated camera case that was referred to as the essential matrix. For the uncalibrated camera case, it's the fundamental matrix. And the math that we've been going through has been talking about how you can find these matrices. How you can find these relations by just giving me the correspondences between points from one image to the next. All right. This ends the lesson on fundamental matrices. In fact, it ends the entire unit on calibration in stereo. I hope you realize that from these relationships, you can capture the geometry of the scene. That is given some camera views, you can eventually figure out the three-dimensional locations of the points. So this is not image processing, right? This is actually vision where we recover a 3D structure. And if you're taking the OMS version of this, of this class and you're doing the OMS problem sets, you're going to get to know and love these. Well, you're going to get to know. I hope you'll love these processes pretty well. That's it for this unit.

4A-L1 Introduction to "features"

1 - Intro

Welcome back to Computer Vision. With this lesson, we're actually starting a new unit, which I've nominally titled Features in Matching. But really, the, the whole idea of this is to find reliably detectable and discriminable locations in different images. That is, I'd like to figure out a way that,

if I have some points in one image of a scene, I'd like to figure out which points are the same points in another image of the scene. Now, you might say wait a minute, we just did that in stereo, but no, you didn't. You see in stereo, the idea was that we were going to have these epipolar lines along which we would search. But remember we talked about that in order to find those epipolar lines, we need to know the fundamental matrix, and in order to do that, we'd need the correspondences, so this is a way of finding correspondences. That's one way of thinking about it. Couple of resources, Forsyth and Ponce do cover this nicely here in section 5.3 and 4. And Szeliski, which is that other textbook that we pointed you to, also does this well in, chapter 4.

2 - Image Point Matching Problem

We talked about that, there were different ways of transforming images one to another. And you know, so I might have two images that are related by some transformation. And we talked about how if I know the transformation, and I know it go from one image to the next. But suppose I have two images. And I'd like to figure out the transformation that takes image 1 to image 2. How would I go about doing that? Well, to do this, I need matching or corresponding points. So, the way we're going to go about finding corresponding points is, using something that we're going to call Local Features. You know, this is a little 1 and a little 2 there. We're going to put some meat behind defining what we mean to be Local, and what we mean to be Features. By the way, some of you have taken machine learning courses and things like that. And then and, you've seen feature vectors. That's not what we mean here. We're actually going to mean that later in the course, here features are going to be things that we compute about some little local spot, they're related to feature vectors but, but we were here first. All right. Our goal is to find points in an image, that can be found in other images. And by the way, not just found in other images, but found precisely. And what that means is, you know, you, you tell me about, I should be able to precisely determine where that point is. So just saying it's the middle of this gray panel, not so great. But if you told me it's this tip of this spear, that would be better in terms of being localized. Also I want to reliably do the matching, right? So if I find this point, if I find a bunch of other points, I want to be really sure that they say no, it's not this one, it's not, it's definitely that one, right? So this is this idea of well matched points. All right? So why do I want to do this? Well, we talked about things like I might want to recover the fundamental matrix to recover geometry, remember we said that if you give if you give it enough correspondences. In robotics and other computer vision things, we might want to see how a bunch of points have moved from one frame to the next, and in order to do that. I have to figure out which points were, were which. Or, I might want to build a panorama. Now, we've talked about this is general. But here's really a beautiful example. This comes from a paper by Brown and Lowe in CV, in ICCV in 2003. Let me just play that again, because it's so cool. All right, and here what they've done is they've figured out what matching points. How to place each one of the pictures on top of a another? In order to do that they had to find these matching points.

3 - Matching with Features

Let's step through this just a little bit. So when we talk about building a panorama, what do we have to do? Well, we're going to have to sort of align our pictures in order to be able to paste them together. That's what we just, just saw. So to do that, here's the basic process. The first thing we do is we're going to detect some features, okay? And here features just means some interesting points. In fact, we're going to define something called an interest point operator that'll find some points that we define to be interesting. Don't you wish you had an interesting person operator. You could just scan it around a room, and it would just tell you who's interesting? It wouldn't go off a lot in my house. Anyway, no, sorry, sorry hon. So, here we've detected a bunch of points in in both images, and these are going to be our feature points. The next thing we have to do, and, you know, just looking at this, it looks a little bit like a mess. We need to find the corresponding pairs. That tells us how we need to transform one image to the next, and then we can use those pair to align the images. And, if we do them in this particular example. You get something that looks like this. By the way the seam has been left in here just to make it clear how that image is being moved. If you

were actually splicing these images together you would do something you know sexy in order to make sure that those seams became invisible. Now to do this detection and matching and alignment, there are a couple of things that we have to worry about. The first problem is well, I can't possibly match points if I don't find them in both images. And remember I'm going to detect interest points here, and interesting points there, and look for a match. So it better be the case that a point that's been found here to be interesting is also found over there. In this particular case there would be no chance to match. Because I found some set of four points on the left, and a different set of four points on the right. So what we need is what is referred to as a repeatable detector. If I run on this image, most of the points that I find over here as being interesting, I will also find over here, over there and vice versa. So that's the first problem, reliable finding them at all. The second problem is let's assume that these points on the left are somewhere in the right. So I found here it's four points on the left, and then there's four points on the right. Which point is which? Okay, I have to do the match. The way we do that is what's referred to as using a descriptor. A descriptor is just what it sounds like, it's a description of this of a point, or of a feature to be precise. What I want are distinctive descriptors, that is, you know, if I write down the description of each of these. So you can think of the description as being the name, right? So I want this one to be Bob, Ted, Alice and George. That's mixing some metaphors from the sixties, by the way. because if they all said George, George, George, and this one says George, then I've got a French movie, and I don't know whether, what's going anywhere, okay. But if this one says Edward, it better be that one of these things says Edward. Maybe if it just says Ed I'll be okay, but my descriptor can't change that much of the same point in, in the two images. So that's what we talk about it being distinctive. You know, Bob, Carol, Ted and Alice. That's who it was from the movie, right? They have different names, but they also have to be reliable. So Ed might become Edward, but Ed doesn't become Francesca. All right? So Feature points are used for all sorts of operations in computer vision. Image alignment we talked a lot about, 3D reconstruction, Motion, Motion tracking, I'm trying to figure out how either the object's moving or the camera. Object recognition in fact, in a couple lectures from now we're going to take some features on, from a bunch of different objects. Take the features in our image, and figure out up, it's this object that's in that image. All right. Database indexing, Robot, features are used for all sorts of things. And that's one of the reasons why we have to talk about them in the class.

4 - Good Features Quiz

Here is a section of the Berlin wall photographed from two different angles. I've marked some candidate regions in both images. Select the ones that you think can serve as good features. Note that when evaluating the fitness of an image region you're only considering the pixels within it. Let me get you started. This grey patch at the bottom left is pretty bad. This nose in the right image is fairly distinguishable.

5 - Good Features Solution

So, any region with two or more edges coming in at different angles is a pretty good feature, like the fat lip that shows up in both images, corner behind the ear, base of this nose in the left image, green corner on the pink wall, also in both images, and this top section of the wall in this right image. The cropped portion of the eye is also pretty good. So is this tooth that shows up in both images, although it could be hard to match given other teeth that look similar. These two regions involving the cycle and cyclist in the right image are useful as well. However, they're missing from the left image. Now look at the remaining regions. This one with the vertical edge in the left image cannot be located precisely. Same for these two regions with horizontal features. Also, this diagonal could be anywhere along the wall. And even these two with slight curvatures are no good. They could be part of larger curves and would be hard to localize. So what makes certain image regions good for feature matching and not others? Think about any common properties you can see. How would you define these mathematically?

6 - Characteristics of Good Features

What makes a good feature? So here, I have a picture just as an example. The first thing is, it's called Repeatability/Precision. The same feature should be found in several or multiple images of the same scene. Now, of course these aren't all the same picture. Because, if I gave you 15 copies of the same picture. This would be a really boring class, all right? These are actually different pictures of the same scene, so things change a little bit. But despite those changes, I want to get the same points being identified, and that's repeatability, and also. I'd like it to be the active same point. All right, so there's this little red dot on top of the camera, there's one over here too. I wonder which one I'm looking at. See I'm going to make Megan cut back, and forth in her video. Okay, the little red dot. So that dot's a pretty good feature, because there's only two of them that I'm looking at here, so I can find both of them, and then matching will probably be easy to do and then they're very precise. That is I'm not going to drift around, I'm going to be able to find right on that dot. The next thing is Salience or Matchability. This is how distinctive the descriptor is. So each feature should be able to be described uniquely enough that it's easy to find it right. So if I'm looking at a bunch of white dots on a black wall, and they're all the same white dot. That's really a problem, because that's the George, George, George, George, George, right, everybody is named George, I can't find anybody, I just quit and go home, all right? So, you like to have a description that's different, about them. Compactness and efficiency. Another way of saying this is, don't get carried away. You remember when we talked about computing the fundamental matrix, we said well, I only need eight corresponding points, but I mean maybe having a dozen or 20 of them might be better. Maybe even 50, but 50 is much, much, much, much, much, much, much, much smaller than the number of pixels in the image. So, what I would really like to do is, in general, I don't want to have a feature everywhere, although in this picture, there's a lot of them. But you see, even here, it looks like there's a lot of them, but I don't know, maybe there's a 100. But there's tens of thousands of pixels here. So, that's what it means by being compact and efficient. And finally, Locality. Locality is, is not what you think if you think locality means exactly knowing where it is. That was the precision. Locality means that, the descriptor that I compute about the feature, the description, the name, is based upon information that comes just. From a pretty small area around the feature. Now you might think that you know I'd like to take a nice big area to describe it because that'll give me a nice very distinct description. That's like a first name, and a second name, and third name, and a fourth name would be giving me a very rich description. The problem is and I'll try to find, oh yeah it's right over here. So I'm looking at the side of this. Now I'm I hope I'm still in the feed. No, the camera moved! I'm looking. Okay, good. Thank you. So I'm looking at the side of the, of the lens over here, and the problem is if I take too much of the lens to describe what's going on, as I move over here, and I start to get occluded. A bunch of that neighborhood goes away. That's the same thing when we had, you know, a finger getting occluded a little bit by the other. So since I'm trying to find matching pixels from different views, and different views will cause a little bit of occlusion to change, I don't want the neighborhood that I use to describe the, the, to make the description to be too large. And that's what locality means. It makes us more robust to sort of clutter, and occlusion.

7 - End

So believe it or not, that's the end of this first lesson. why, well I want to cut you a break a little bit. Actually the real reason is the next lesson's got, shall we say a, modicum of equations on it. And to do this and then all of that, boy that would be too much, so here we are. But the idea is that we've motivated what it means to be a feature, and a good feature. Features are being used to match one to the next. The two main properties that we talked about is repeated detectability. That is, if I find them over here, I'd like to make sure that I'm going to find them over in this image. And they also should be good for recognition one to another. That is, you can think of them as having a signature. And that signature should be distinct, right? Bob, Carol, Ted, and Alice. Alright. But robust. Right? So Edward might become Ed but nothing significantly different beyond that. Turns out that this notion of being both distinctive and robust sort of works against each other and, and we'll talk

about that. So going forward, we're going to introduce about what it means to be a good feature, what it means to be a good interest point, and we'll break that down into two parts. Detecting of interest points, and then characterizing the descriptor around those interest points. So sharpen your lead pencil. You guys even see lead pencils anymore? Don't use a lead pencil on your iPad. It's probably a bad idea. Anyway and we'll do some interest point operators, and how those function mathematically.

4A-L2 Finding corners

1 - Intro

Welcome back to Computer Vision. Today we're going to talk about finding corners, which might not feel like the most exciting thing in the world, but actually it is fundamental to a whole bunch of computer vision. Last time we talked a lot about the notion of features, and features were sort of locations, in the image that you could find in multiple images so you could match them, whether you were doing, essential matrix finding or some other reason or some, transformation. And we talked a little bit about what makes a good feature. So here was an example of some feature points. The first thing that makes a good feature is simply repeatability. Slash decision, but really repeatability, meaning if I find you in the left image, I better find you in the right image. Or if I have a sequence of images that I'm trying to line, I want to find that same feature point. I want to detect it as a point in as many of them as possible. The other thing is that I should be able to be sort of precise in my description. We'll talk a little bit more about locality in a minute, but the idea is that you know, this is the point, I find it every time, and I find it more or less in the same place, with respect to the scene. The next thing we said bu needed to be important about a feature was the saliency or the matchability and what we mean by that is, that it should have a description that's distinctive. So when I find it in one image, when I look in another image, there aren't a lot of other points that might match that first image, the, the description should be distinctive. Something that we sometimes forget is that one of the reasons we're doing this, finding these features, is that we're going to use this, remember I think I used the term constellation of features. This collection of features, is going to stand in for the image. So, if you remember for fundamental matrices we needed eight points although a dozen or more is better, at 20. But the idea is that this is a number much smaller than the number of pixels, so compactness and efficiency says we should find good features, but we should not find a bazillion of them, or even half a bazillion. We should find enough, but not too many. How's that for a technical description? But the idea is that you're finding enough. But it's much, much less than a number of pixels. And finally locality. And locality means that, the description of the feature is, dependent upon a neighborhood, but not too big a neighborhood. If you remember one of the reasons we said that is that if. If the neighborhood is too big, then if I change my view point a little bit and something's occluding part of that, even though I can still see the feature, the neighborhood might have been partially occluded and then the feature might not be detected anymore or, or not described the same way. So locality means that I want a descriptor based upon a region around it, but not too big a region.

2 - Corner Detection

All right. So, fundamentally to in terms of makes a good feature, we need to talk about finding one and describing one. And today we're going to talk about the notion of finding these features, sometimes referred to as interest points. And these are going to be defined to be the points of the image that would likely make good features, that is ones that satisfy all those criteria we just talked about. So, let's start thinking about an image pattern and imagine you're looking at a large black square against the white wall, okay? Would the middle of the square be a good interest point? Megan, what do you think? Yes? Wrong, no. Look, remember one of the things we need to be able to do is precisely locate this? Well, in order to try to find the middle of some big square, there's like nothing there. It's all black, I wouldn't know where to land. Okay, well, what about along the edge?

Will a point along the edge be a good interest point? Megan what do you think? Yes. No. Look, okay, so if my edge was vertical we're going to do this in a minute, if my edge was vertical I could tell which way to move left and right. But as I move up and down, nothing would really change, right? I just have black on one side, white on the other side, and this edge in the middle, and I wouldn't be able to tell where that point lands. Okay, you ready, Megan? Two strikes. What about the top left-hand corner? Would that be a good point? Yes. Yes! She got it right! Swings and hits a home run, well, a triple. That seems pretty good, right? The idea is that, I know exactly where that point is. If I told you, go point to the left, top left hand corner, you'd know where it was. Whereas if I said, go point to the middle of the square, the exact middle, you'd look at it and say, I, I don't know. You'd have to get Megan, and she wouldn't know either, and she'd say, oh I'm sorry. Anyway, okay, so let's take a look at this. All right. So that this really is the basic idea of corner detection, all right? So here I have a black, sort of zoomed in on a black, kind of rounded square against a white background, where there's brown little square is, right? I could move that square up and down or left and right. And basically the pixels underneath that square wouldn't change very much. That's what is referred to as a flat region. That is that there is no change in the intensity. Well, here's our edge. Okay, so now we've put our square over the edge. So what do we see? Well, it's the same thing we, we talked about before. Now, I can know about moving left and right. I would know exactly more or less where that edge is. But if I were to move up and down. Okay, I wouldn't be able to see any difference, because there would be no change. So finally what do we get to, aha, we get to corners. Okay. And the idea is that in the corner, basically because I have gradience in more than one direction, no matter how I were to translate that square, the pattern underneath the square will change. And so I can know when I've lined up exactly where I was before, right? So there's this idea that we need the gradients to have more than one direction.

3 - Harris Corners

So this basic idea of finding corners in, in thinking about it this way. I don't know if I would say that it was first introduced. But I'll say that it was most well known introduced way back in 1988. I know some of you guys were, like, in diapers. And some of you were, a dream in your parents' eyes. And some of you were a nightmare in your, no, whatever. It's a long time ago, all right? And, the picture here down below is just showing you sort of a mess of corners. And they don't all look like corners, but they, they're places where the gradient varies. And it was created, this paper, the Alvey Vision Conference is what later I believe became the British Machine Vision Conference. And it was done by this, by Harris and Stephens. Now. If you ask anybody, about the corner detector I'm about to tell you about, they'll you that they're Harris corners. Because that's what everybody remembers. I know an awful lot of people in Computer Vision, I don't know who Stephens is. So this tells you something. That if you're doing a paper with somebody, try to get your name first. Unless of course it's your advisor and he's paying, he or she's paying for your education in which case then you know. But, anyway, so I'm it's, it's actually Harris and Stephens and you can impress somebody by saying, Harris and Stephens corners but it's, it's Harris corners. All right. So let's learn about Harris corners. So, Harris corners are. Based on a approximation, model and an error model. And here's how we do. Let's assume, that we've got an image L and right here, right and I is the intensity image. And what we're going to do is we're going to say, if I shift that I by a little bit, by a u and a v and I were to sum up the square over some window. How much error, would i get? K thats thats thats the game we're going to play. So stepping through that, I is our image intensity, so I of x y is just the orignal image. The u and the v is some small shift, so if I were to pick up my window and move it over a little bit, I would now get some different image just shifted by a little bit. I subtract them and square them, and then I, going to, I'm going to sum them up over some window. Okay? So the window is going to be some area, around some point. So that's written here as w of xy. The total value here, E, is sort of for error. You really could think of it as sum of the change where E for error or energy. But the idea is that we're going to sum it up. Around that window. Now of course your window can typically be a square window, like in my house, or if you want to actually work a little bit, or, on a, on a slow computer you make it be a square window. But if you actually want to weight the pixels near the point you're thinking about,

more then the pixels that are far away, you would use a window that falls off in intensity, intensity So like a Gaussian window would be an example. So. Typically your windows can be square, if you need to be really fast, or smooth, if you need to be really good. So, are you fast, or you good?

4 - Harris Corners Illustrated

So now we can start to graphically illustrate this. So on the left, I have an image I of x, y. And on the right, I have the new function E of u, v. Okay? So remember, E is the error. Okay? And what we're going to do is we're going to put 0, 0 sort of in the middle, right? So this goes plus in u and minus in u. This would be plus in v and minus in v. All right? So what if I just leave the square right in the same place? What if I make u and v be zero? How much error am I going to get? Well, none. Okay? And so E of 0, 0 is 0. I'm just subtracting the same image from itself and that's what this E 0, 0 is meant to show. Oh, and by the way, just showing you this little pixel here is black. Why is that black? Because that's zero, right? There's zero error E of u, v. Right? Error because it's the sum of squares could only give, become more positive. Right? So at zero, zero it's going to be an error of zero. So now suppose I move my window just a couple of pixels, like 3 in the u and 2 in the v direction. Now my error going to be whatever it is, right? And over here, right? You notice this is just a gray value, right? Because the value is gone higher, right? So the error grows as I move away and just think for one minute, suppose that picture were perfectly gray, every pixel is the same value. That error would be zero at the middle, at zero, zero and as I moved it, it would still be zero. So that should give you some idea of where we're headed. We're looking for points where as I move in u, v, I get this nice change, no matter how I move in u, v. Got it? Cool.

5 - Small Shifts

So, what we want to do is, we want to figure out, we want to model how this error function changes for small shifts. So we're not going to move you of 97 pixels. No we're going to move, maybe even just a fraction of a pixel or a, or a small shift. Right near u and v being zero zero, that is around no shift at all. And of course what we'd like to see is that we get a significant change even for a small shift that would tell us no matter how we move in u and v, and that would tell us that this image is going to change as I move around, and maybe it's corner-like. So how do you predict the value of a function for small shifts? Do you remember this? Do you remember this? Oh yeah, calculus. Taylor expansions? All right. We're going to do a second order Taylor expansion of E of u v. That's our function here. Around u and v being zero zero. And this second order means it's going to be a small, a quadratic approximation. All right, so strap in, because we're going to go through the math of doing that, and it'll get us to a point where it's very easy to compute whether the neighborhood around a point feels corner like. All right, so you may not remember everything you need to know about Taylor expansions, so I'm going to write this out. So back when you first took calculus, this was the equation of a Taylor approximation, where we say that the value of a function of some small delta near zero can be approximated the value at zero times the first derivative times delta x times the second derivative of delta squared x half of that. That was the Taylor approximation. I'm sure you all have that tattooed somewhere on your body. All right. So it gets a little uglier when we do things in two dimensions. Remember it's u and v. Okay? And we're going to write that out for this expression, this error of u v function, this way. You ready? Okay, don't panic. What we're saying is, that the error function u v for some small uv's near zero, can be approximated by the value at zero zero. The first derivative times the u v, so that's like this d f of x times delta x, except now we got it in two dimensions. So we've got the gradient and the u dimensions, plus the gradient of the v dimension. Plus, we have this quadratic term and in vector notation for two dimensions, that quadratic term is written like this. So these are our second derivatives. Uu vv, this is uv uv, and we just matrix multiply times the uv vector. Hopefully this is somewhat familiar to you from your vector calculus if not, it's just true

6 - Second Order Taylor Expansion Part 1

So now what we're going to do is we're going to step through bare with me this whole second order Taylor expansion about u,v of 0,0. By the way you should appreciate that I am not writing this. So you actually can read it and it will be PDF presented of some sort so you can get it from there or you can go to Rick book it's, it's all there. But believe me you don't want me to write this you just want me to be humourous and light hearted. All right, so here we have our Taylor expansion written just as before. So we're going to knock these derivatives down one at a time. We're going to start with our first derivative, right? We have to do the first derivatives then the second derivatives. So the first derivative we're going to look at [LAUGH], the first, first derivative we're going to look at. Is respect to the u direction. Now u is the offset in the x direction. Okay, just remember that u is the offset in the x direction. And we're going to take the derivative of this function E of u,v with respect to u. Okay, let's see, well we've got this square term here, so that gives us this, we, we raise it to, to single power instead, and then we have to take the derivative of the inside term with respect to u, and, let's see, there's no u's in there, that's go away. So there's a u in here, so I have to take the derivative of I of x plus u, y plus v, with respect to u. Okay? Not me, you. Anyway. Well, remember, u is in the same direction as x, so the derivative of the image in the u direction is the same as the derivative of the image in the x direction. So I need the image gradient I sub x. Got it? Good. I'm not going to do v. V's exactly the same thing except instead of xs, you get ys. Now what we do is we're going to do the second derivative. So we're just going to take the derivative of this with respect to u. Well, there's two parts here. Right? We've got this part, and we've got that part. Right? So the derivative of a product is u times dv plus v times du. Remember? Yes, yes, yes, yes, yes. Except Us and Vs is a terrible choice because we've got different U's and V's. But basically the derivative of the first time the second plus derivative of the second time. The first what is that going to look like? That's going to look like this. Okay? So we take the derivative of a part that's the image and multiply it so that just gives us this twice. And then the second one is we take the derivative of the derivative and multiply that by the original. That gives us the second derivative. -okay? The math's pretty easy, you can work it out. This is the e respect to u, u term, all right? So we've done the first derivative, the second derivative, u and v are the same. The only thing left to do from that matrix, we had e, u, u, e, v, v, we need the the cross derivative and that just looks like this where we take the derivative of,- The previous one of u except to v now. And we get something that looks like this. And you'll notice that this has the gradient in y, the gradient x. Remember x is the same direction as u, y is the same direction as v. And it also has this cross derivative. We haven't talked a whole lot about the cross derivative, and in fact, we're not going to talk even more today about cross derivative. because later we're going to make it go poof! Go away. So we won't even think about it. So now we have our first derivative, u,v is the same. Our second derivative u,u. v, v is the same. And our cross derivative. Right? Right.

7 - Second Order Taylor Expansion Part 2

So here they all are, in their incredible glory. All right? So this equation is ugly. But, we're going to be able to make a bunch of it go away. And the reason that we're going to be able to make a bunch of it go away is. Since we're going to evaluate it at u and v equal to 0, 0 we're just going to make all these plus u's and plus v's. Everywhere be 0,0 okay. Remember we're evaluating our derivatives at 0,0. So all of these terms that have the plus u, plus v, the plus u, and the plus v go away, and you get something much simpler that looks like this. Now that still looks kind of ugly. Okay. Until we start to realize that a lot of this stuff goes away. So first of all what is E at 0,0? What is the error I get if I don't slide the thing at all? It's 0. Great. Next, I've got this funny little term here, it says I of XY minus I of XY. What is that? You betcha. 0. Okay? And here it is again, 0. And notice, this 0 is the one that's multiplying the second derivative term. And what about down below? Another 0. That's the one that's multiplying the cross derivative term. So, when I'm. Get rid of all of the stuff that equals 0, my equations become very, very simple and they look like this. Okay. My main function is 0. My first derivative components are 0. And these components which go in the matrix, they are just these simple terms here, okay? And you'll notice that the only thing in here is first

derivative with respect to x times first derivative with respect to x. First derivative with respect to y times the first derivative with respect to y. And the first derivative with respect to x times the first derivative with respect to y. Say that seven times fast. There are no second derivatives left. So getting ray, rid of all my erasures, we take that bottom part, we shove that into the top part there, because that's the only thing left that's going to be non 0, right? Because this is all 0, and this is all 0 because of these zeros. All right? So when we do that, what does our equation look like? It looks like this, all right, so the only thing we have left is that the valuation of that error function for a small u and small v is approximately, remember it's a Taylor series approximation, is approximately this function, okay. And remember, no more second derivatives, they've been killed by multiplying by these zeros, we just have the squares of the first derivatives, and then the product of the two different derivatives

8 - Quadratic Approximation Simplification

That's still pretty ugly. We can make that look even nicer. All right? Here is a very simple, beautiful way of writing this. That this quadratic approximation, so quadratic, remember we're just looking at the square terms of the u v. Right? It can be written matrix form as u v times this matrix M times u v. This, you know, you could think of this as u v, transpose. All right, where M is the second moment matrix written down here, okay? Again, what is has is it's the weighted sum of the squares and of the product of the two different derivatives. It pays to think a little bit more about this M matrix, because we're going to do it even more in a minute, all right? So, here I've just copied it over, and for a moment, it's kind of a joke. For you guys that're physics geeks, if I say for the moment and we're talking about the second order matrix, you realize it has something to do with moments of inertia. Nevermind. Okay, I'm going to write that M matrix in, without the weights at the bottom. Okay. And when I write it that way, you can see that this is just the sum of ixx, sum of iyy, and the cross sum. That can just be written as the sum, this is referred to as an outer product, right? So it's the outer product of the single gradient, okay. That's what this says. One of these is just the outer product of one vector. It's a two by two matrix but it's an outer product of one vector. In linear algebra you realize that's only a rank one matrix, right. Because let's just take the columns, the columns are just a multiple of Ix Iy. And each column is, is a, is a, is a multiple, multiplied by a different scaler. So when the two columns are scaler multiples of each other, it's not a full rank matrix. So its two by two is actually rank one. Okay, that's what it says here, each is a product of rank one. And then of course, this thing is going to be summed over all of the points within the window. So what's it going to take for this matrix to be full rank, for this to be sort of a well-behaved ellipse? We would actually have to have gradients in different directions over the window to be summing up a bunch of different rank one matrices and that would get us to a full rank.

9 - Interpreting the Second Moment Matrix Part 1

So looking at that we can think about it this way, all right? So, the surface, E of u,v, in this quadratic form, is being approximated essentially by this quadratic or, or sort of parabolic shaped form here, and that's just written the equations out. Now, let's think just for a moment if you were to take a fixed value of E. Okay. So, we're just going to pick some constant value. All right. So, that would look like this. All right. And here I've written it two ways. I've written it sort of down below in sort of the elegant matrix way, which we're about to cover up. because I like inducing pain. And the top way is just the algebraic expression of that expression equal to a constant. And what you can see here is, now you remember, this is the equation of an ellipse in u and v space. All right, so you remember A x squared plus B xy plus C y squared equals D or something like that, was a, was a, was a an ellipse. And if the middle term, the xy term, was 0, it meant that the ellipse's major and minor axes were aligned with the x and y axis? Well, this is the same thing. This is just the, an ellipse in the u v space, all right? So we can draw that out like this, okay? So here is different ellipses for different values of k. So you can think about that the way we're approximating this whole surface is as a quadratic surface that gets bigger and bigger, and each different level is an

ellipse. Okay, so that's what you get when you do the type of second order quadratic Taylor expansion of, of this function.

10 - Interpreting the Second Moment Matrix Part 2

So we can look at this ellipse a little more carefully, okay. Let's first consider the case, where the gradient at every point in the image, is either horizontal or vertical. Sorry, not in the image in the window. So, at each point. It could be horizontal or it could be vertical, but it's not going to be slanted, okay? So some places it'll be horizontal, some places it'll be vertical, but it's never, the, both Ix and Iy are never non-zero at the same time. Well, that would mean that these terms would always be 0, right. Ix times Iy, all right. So, that would look like this. And I'm writing this as lambda one, lambda two, these are the diagonal terms. The off diagonals would be 0. Now, that's a nice full right matrix, right, lambda one, lambda two. Except if one of those were 0, it would not be a very good full, it would not be a full rank equation, right? So for example, if the Iy squareds were all 0 and that there was no gradient in the Y direction, only in the horizontal direction, that would mean that my matrix would only have one of these values, and probably, that's not really a good corner, right? Because if I only have say gradient in this way and not in that way, then I can slide up and down as much as I want. So, something about these numbers tells us how good a corner, the location in the image is. Can you smell the algorithm that's coming? It's not me, it's Megan. In fact, I'll write it just this way. If either of those lambdas were close to 0, this is not a corner. So what we want is we want, is we want to look for locations where both of those are large. Now for those of you who remember from your calculus and and you're learning algebra, of course, we can take this second order matrix, this ellipse, and what we can do is we can diagonalize it through a rotation, and remember we talked about singular value decomposition, well this is not exactly that but it's quite related, where basically, what it says is we can think of M which is that special matrix as being sort of rotated or unrotated by R. Then it's a diagonal matrix and then it's re-rotated. The axis lengths of those ellipses, okay how big those ellipses are, that's determined by the eigenvalues these numbers. Of that diagonalization, and the orientation of the ellipse is determined by that rotation. So here I'm drawing that, like this, so remember, this is a particular ellipse of constant E, constant error. So for places where, it's a very narrow ellipse. That means that the E changes quickly. I didn't have to move very far for the E to get to that value. Whereas, the places where it goes very slowly, okay, I have to move a farther distance, in order for that value to change. And you see how it says it's one over the, the square root? So, remember, if that lando were 0, okey? That would mean I'd have to go an infinite direction, that means it doesn't change at all as I move in that direction. So what I'm telling you here is that the eigenvalues, of that M matrix, are indicative of the cornerness of that matrix. That's pretty cool. So, it turns out, Harris figured this out and they figured out a really simple way of thinking about this.

11 - Interpreting the Eigenvalues

Here's how we going to do. The idea is that we're going to classify our image points based upon computing those, that Harris matrix, and looking at some measure that's a function of its eigenvalues. And the idea is that in a flat area, both of those eigenvalues will be very small. In fact if it was perfectly flat, everything would be zero because the gradients were zero everywhere. And that would mean no matter how I moved, nothing changes, so it's a terrible corner. Likewise if I have just one of the eigenvalues large, that means there's some direction that changes quickly so if I move in that direction, I know things have changed. But if it's zero in the other direction I can move as much as I want. Remember that's what happened along an edge, so when one eigenvalue is much, much bigger than the other then you're along an edge up here. And when is it a corner? Well, when both of the eigenvalues are large. And we'll say that. It says here that they're approximately the same magnitude. Just that they're both big enough, and that the ratio between them should not be terrible, all right? If you're, even if they were both large but the ratio between them was 10 million, then probably your code was wrong. If they're moderately large and your ratio between them is still pretty big, then you have to worry because it means one gradient is much

much more dominant than the other, and maybe the noise is affecting the finding of the corner. So you want them to be approximately the same magnitude. So you might think great, that means all I have to do is find the eigenvector, and the eigenvalues, right? Right. Guess what? You don't even have to do that. This is what was so good about the Harris thing.

12 - Harris Corner Response Function

You have to remember something from your linear algebra. And by the way, I have to look this up every time, so if you don't remember, that's okay. One is that the determinant of a matrix, remember the determinant of a matrix? That's actually the product of the eigenvalues, okay? So lambda one, lambda two, multiply them together, that's the, that's the determinant. With two zeroes in there, it's easy to see. It's a little harder to see in the more general case. The other thing is that the trace of the matrix, right? The sum of the diagonals, is the sum of the Eigenvalues. So the determinant of a two-by-two is trivial to compute, right? This minus that, or however you like to determine some two-by-twos. Trace is easy, sum them up. And the Harris operator was just to compute some function R. You take the M matrix. You take its determinate, minus some constant of the square of the trace. And I'll tell you that empirically, the constant was small number, 0.04, 0.06, written here, exactly how things scale depends a little bit on your, the, the way you do your gradient operators and pixels. Okay? So it's the determinant, which is a product. Minus the alpha trace. So the real equation of what you're looking at is right there. Okay? So taking a look at lambda one lambda two minus alpha one, alpha times lambda one plus lambda two squared. Let's think about this. R depends on the Eigenvalues but we don't actually have to compute them. And that's great, because even in 1980, in the 80s you'd want to compute your eigenvalues, maybe, I don't know. But the idea is that you compute a single number, all right, really fast. All right. So, we've already said that R is large for a corner. And why is that? Well, if they're large then both of the l one and l two are large. Their product is even large and if you take the squares. You know it's l one square plus l one l two plus l two squared. Subtract it off. Yeah but you're subtracting off alpha of it. It's a small amount. Okay? So it's still going to be large. But if only one of the L's is large, then this term goes small, whereas one of these terms is large so the R value is negative, okay. So that you can measure the R value, it's negative along an edge, so R is large for corner, it's negative, along an edge and if the magnitude of R is small, basically that means that everything was small, right? So L1 and L2 is, is small, everything there is small. Well, that's the flat region we were talking about, right? None of the gradients have any magnitude to speak of.

13 - Textured Regions

So let's take a look at this working. Here is some images taken from there's something called, the mpeg flower garden sequence. It was used for demonstrating certain kinds of image processing algorithms. It's just convenient here. So the first thing is, let's take a look in a low textured region, all right? So, there's not very much going on there. The gradients there have very small magnitude. And what's being shown here, both as an image, and as a plot. Is, how does that error function change, as I move around in u and v? And what you can see is, there's no sort of, obvious peak anywhere, be a positive or negative in here. To say that, that that's' the obvious place for, where you would want to find a feature. And remember, we're, we're looking for that. Now, let's take a look at another region. So here we have a point that's on this edge that's where the, the roof line is, okay? And you'll notice that we have this ridge here, okay, of a, of low error. And that's sort of, represented through here and by the way, in this just make the, the pictures easier to understand. The bright thing here is, is, is good and, and black is bad. And you know, you don't want to go this way, but going along this way, you don't know exactly, where your corner is. So, let's take a look at sort of another patch, another place, and here we are right in the middle of the flower bed. Now, both of our gradients are large. Our lamp, the gradients are large in many different directions. So we have a large l1 and a large l2. We get this nice peak down here, and that's shown, as this nice peak here. The idea is that, this would have a nice large, r value, Harris corner r value. That would find you, that says, you know, what? If you give me that point. And I'm looking for that point again

in another image. Any small motion is going to give me a big error change, so that's a good suggestion for, where a corner might be.

14 - Harris Detector Algorithm

All right, so here is the simple Harris detector algorithm, courtesy 1988, still used today a lot. We're going to talk about different versions of it, going forward when we talk about scale and stuff, but for now, we're just going to talk about this. And here's the basic algorithm. You first compute Gaussian derivatives at each pixel. Okay? So remember you need this ix and iy. And we've talked all about how to do derivatives, where you take the derivative of the Gaussian operator, apply that, and you get back, your nice image derivatives. Then, you compute that second order moment matrix M, that M matrix that we were talking about. And, the recommendation is you use a Gaussian window so that you weight the points near the middle more than the points that are further away. And then you compute, that R function on that M matrix, remember you have a different M matrix for every point in the image. Right? Because you take a little window around a point, compute the moment, compute R, move on. So you do have one for every point in the image. So you get all these Rs, you're looking for high values, you threshold that. And then, and we talked a little bit about this when we were doing the edge detection, you do what's called local non-maximum suppression. Right. So I might have a heart, I might be a strong R value, but if there's a stronger R value next to me, throw me away and keep him. Sorry about that. But the idea is that you're looking for local areas that have the highest possible R value. And like I said, this was presented originally in this paper in 88.

15 - Harris Detector Workflow

Just to show you some examples. So, here we have two pictures. I wish I knew where these pictures came from. These are, like, so hysterically cheesy. Two pictures of a toy giraffe. You'll notice that a couple of things. First of all, obviously the orientation has changed a lot. But also the head has been rotated, right? So, here it's on the side, and here he's facing you, and so things have shifted a bit. the, the feet are a little different. Of course, the entire object is rotated and the intensity has changed. So, there's been quite a bit of change done to the image. So, here we go. We compu, we compute our derivatives, we compute our moment matrices, we compute our R functions. We plot them in MATLAB, using a interesting pseudocolor colormap, and you see this beautiful picture. And that's R everywhere. And the next thing you have to do is threshold that. So, here are all the points that R is above some particular threshold. Then we do that trick of the non-maximal suppression, right? So, we only keep points that are locally higher value than anybody else. And this is probably pretty hard to see. Megan helped a little bit by putting this line down the middle. So, there's this constellation of points on the left and the right. You know, if I look at that, I can't really tell it's a giraffe. But, what I can do is, I can show you those red points on the actual giraffe itself. This is a giraffe, right. Yeah, it's kind of a, kind of a weird giraffe. Somebody's fanta, it's like a creepy giraffe. It's like always smiling. Anyway. All right. And here you can notice that we've found a bunch of points repeatedly. So, for example, on his nose, on the right nostril there, there's a point right there. Well, guess what? We found the same point right there. So, I wanted to see how the head was, was changing, right? Points on the ears are there, okay? Points on the back of this spot here are there. Points on the bottom are there. Now, of course, all of these points, they're not there. Well, of course not. There's a specular reflection in the left image that's not in the right. That's okay. Harris couldn't possibly know about that. The point is, ha-ha, that we reliably found many of the same points. And if you remember what we were talking about for doing this interest point, or keypoint detector, we need a reliability and a precision. We have to find the same points and find them in the same location on the scene or in the scene. And that was what would make a good feature

16 - Other Corners

I should mention that, there've been other attempts at not, the other pieces of work on finding corners. In fact, Shi-Tomasi did this nice work on, cornerness, and it, it did something different looking at the lambdas, instead of using that r, and found local maximums. And, I will tell you, if any of you who use OpenCV, they have something called OpenCV Good Features to Track, it's a function, it finds you points in the image, that are good features to, good points to track. How does it work? I'm pretty sure that's the Shi and Tomasi version of this. It's still based upon the same moment matrix, it's just a different use of the eigenvalues. And, there's reports that maybe it's a slightly better way for things that are undergoing certain kinds of deformations. There was another one 2005 Brown, Szeliski and Winder, and I write this one because I've stolen so many of these pictures from Rick Szeliski's book, I feel like I have to put this in there. It again looked at eigenvectors in another slightly different way, and there are other, types of, of corner detectors.

17 - End

That ends our very long lesson on Harris corners. I'm hoping Megan can do some, like video magic. So you've actually just seen three magnificently edited, cut, tight, get, you know, I hear she's friends with John Williams, so there'll be a score in the background. Anyway, but this notion of variation and gradients is very important in general. It's used for feature detectors. But it's, but this notion of gradients varying over a window will be important when we do other issues having to do with motion and stuff like that. But for now, we're just thinking of it as doing point detection. And in the next couple lessons, we're going to take a look at some of the properties of this detector and then we're going to build some descriptions, some descriptors around those detected points. Are you tired? I'm tired. All I've got to drink is water, I hope you can do better.

4A-L3 Scale invariance

1 - Intro

Welcome back to computer vision, today we're going to continue our exploration of future detections. Remember last time we started talking about interest points of ways to detecting locations the image we can get back to. And we, focused on finding corners, and why corners? Well as we talked about. Corners had this nice property that if you put a little operator over, over an area that was a corner, whether you moved it left or right, or however you moved it, you would see a pattern changing. And that was not true if you were over a flat region, or over an edge, or an edge if you moved in the direction parallel to the edge, you couldn't see, any change. And, we derived some, nice mathematical formulations, where we talk about the error as you slid things around. We said how the error was zero, before you do any sliding at all. And that, as you move, the error changes a little bit. And, what we did was, we said, well, can we build a model. Or can we approximate that error, by looking locally at the structure of the image. And then we wrote this out as a expression really just involving what's called the second order moment matrix. That moment matrix being the sum of the squares, of the first derivatives. So I x squared, I y squared, or I x times I y. And even though it was a second order, moment, there were no second derivatives, right? No I, I sub, xx. Cause those all went away when we evaluated around u and v being zero, zero. You can go back and re review that lecture. And then we talked about how Harrison who? Harrison who, yeah who knows. No Harrison Stevens, did this thing where you derive an operator based upon that little matrix based upon it's demonstrative trace. And you could use that to find good points. And we showed this horrible giraffe image and we said, yep, the thing seems to work pretty well finding those points

2 - Harris Detector Properties

What we're going to do is, now we're going to look at some properties of the Harris detector. And actually of, of one or two other ways of detecting some interest points. Remember, the reason we're doing this is to find points that are going to be our feature points. So, let's start with rotation, okay? If I've got an image. So we have here a little. Notional corner. And I would rotate it, how would the Harris operator change? Well, remember that whole thing about the ellipse rotating? Well the ellipse would rotate, right? But it would be the same ellipse, right? Because the variations in the gradients would be the same over the window. They just, the whole thing would have rotated. So basically, the Harris corner detector is invariant to rotation. That's cool. You don't have to take my word for it, you can take somebody elses word for it, no, you can actually study these things empirically, alright. So the idea is you take some picture, you find some corners, and then you change the picture somehow, right. So, maybe you rotate it a little bit, and then you find some corners again, and you measure sort of this repeatability. Right? So the number of correspondences that were found versus the possible ones that you should have found. That is, how repeatable was the ability to find the same corners? And as you can see here, as they varied the rotation of the picture all the way up to 180 degrees, you've got pretty good reviews. By the way, ImpHarrises improved Harris. But you can see that both Harris and the improved Harris are. Pretty invariant with respect to rotation. By the way, these plots come from this very nice paper from Cordelia Schmid and, and folks there. And in fact, here it is, showing you, it's from the International Journal of Computer Vision. I put this, these sort of basic res, sources in here just so you can see where some of this stuff comes from. And if you're interested in computer vision, I encourage you to take a look at what these. Papers actually do, to get a sense of how this stuff got derived.

3 - More Harris Detector Properties

What about changes to intensity? Right. I change the intensity somehow, what happens to my corner detection? Well it turns out, that it's mostly invariant to additive and multiplicative intensity changes. So as it says here. Look, the moment matrix is just made up of IXs and IYs and the products of them, well IX is the derivative in X, the IY, the derivative Y, if I add a constant to my image, nothing changes. My derivatives are the same. So it's totally invariant to changes in intensity shift. I just add a value. What if I multiply everything? Well when you multiply an image, you multiply the gradients, right. Well when you multiply the gradients, you multiply the R value, and that's what's shown here. Here we have an R function with certain peaks that were above a threshold. And if you were to scale the image by a scaler you would just grow those peaks. All right? But the shape would be the same. And the only thing you have to worry about is since you're using some threshold, maybe some points wi-, That were above the threshold or below, depending upon whether you're going up or down, maybe they would now change which side of the threshold they're on. In practice, intensity variation like that is not an issue. What about change to scale? Well, what will be clear is, it is not invariant to image scale. So, here what we have two different curves. Here I have my original curve, like this, and here I've just zoomed out and I have a much tighter curve like that. And the little boxes here, those are meant to be the size of the window. And if you take a look at each one of these, all these points will be classified as being edges, because, essentially, there's not much gradient in the directions parallel to the edge. But, if I zoom out, this is now a very good corner, and so, the problem is, is that. By changing this smooth curve. By zooming out so this curve becomes tighter and tighter and tighter. It goes from being a bunch of edges to being a corner. So in general, the Harris corner finder or corner detector is not invariant to scale and guess what. The data proved that. Okay? Here's a another graph taken from a similar paper. And you can see that it falls off pretty drastically as you get to scale. In fact eve with just a scale factor of two. So I just. Zoom in or zoom out by a factor of 2. I've already lost almost 80 percent of my repeatability. So it's really, really not invariant to scale and we're going to have to do something about that.

4 - Scale Invariant Detection

So let's just think for a minute. Suppose we wanted scale invariance. How could we do that? So to think about this, let's consider some regions, and I'm going to use circles, it's easier to think about, of different sizes around some point, okay? So here on the left, I've got a curve that is the zoomed in version of the curve on the right and I've got some circles around this. You know, when I look at the part that's in this circle and the part that's in that circle, I say, no, that curve doesn't really look the same. If I make the same circle a little bit bigger, no it doesn't really look the same. How about now, no, not yet. Then I say, oh, wait. Now with this bigger size, this piece of curve looks a lot like this piece of curve. All right? So the regions of corresponding sizes will look the same in both images. It's not the same size, I, I call them corresponding sizes. It is, it spans the same amount of the image. So the problem, of course is, we have to choose the circles independently, because I'm dealing with this image and I've gotta figure out what scale to do things. And I'm dealing with this image, I've gotta figure what scale to do things. And I want it to be the case the way the, the red circles are indicating here, that the scale I picked for the left-hand image is this nice big circle and the scale I picked for the right-hand image is this small circle. So, that's what we want to do. We want to be able to do this independently on the, in the images. So, how might we do that? Well, here's a solution. Okay. And graphically, it'll be a little bit easier to see. What we're going to do is, we're going to design some function, okay? And it's going to be a function over that circle, and this is a little bit tricky, we're going to call it scale invariant. And what we mean by that is, the function doesn't actually care how big the circle is, it just cares about, say, the distribution of the values of the pixels within that circle. So a simple example, not one that we're going to use, but it's simplest to understand is imagine average intensity. All right. If I just take the average intensity over a circle, that function doesn't care how big the circle is. It's just going to compute that function over whatever size circle you give it. And in fact, if there were a constant grey value, if you kept making the circle bigger and bigger and bigger, it wouldn't matter. The, the function would stay the same. If you have a function that changes over different corresponding regions, then that function would go up and down as you've looked over those different regions. And if I zoomed out of that thing, it would go up and down again. But now it would go up and down faster, because as I grew my circle, I would be covering more, more of that picture as I move that circle out. Now, I know that was clear as mud, so let's take a look at a picture. All right.

5 - One Method for Scale Invariant Detection

So imagine at a point, you compute one of these scale and variant functions like average intensity, over a range of neighborhood sizes, okay? So, we're, we're at a single point and we're computing this function f over different region sizes. And you'll notice that, well it goes down, and then it goes up, and then it goes down again. So that's my function that I computed over different region sizes, all right? Now, let's suppose in my next picture I shrink the thing down by half. Well now, as I compute that same function over the same region sizes, two, four, eight, 16, right? It's going to take the same shape, but it's going to get there faster. Which is why this function is essentially just a squashed version of the one on the left. But it's the same function, just squashed. Ha. Well, what that means is, suppose we pick the maximum of that function. I'm going to call that S1 in the, in image 1 on the left. And I take the same function and I'm at some other point. I'm same, same function on the point of the second image. If I were to take the maximum there, it would be a different size, right? Because that image had been squished in half. That would be S2. But these two neighborhood sizes would be covering the same region of the image between the two pictures. So they would be matching scales. So, the important thing is, I do each one of these things independently. That is, I do this thing and at this particular point, I find my maximum function my, my maximum of the function. I say that's the natural scale size that I'm going to use. Then I come over here and I go to my other image. And I do over here, and I'm at some interest point, remember we found our interest points. And I look over all these different region sizes, and I say, aha! This is the scale there, and I compute that function, and magically, these two scales correspond. These are the same chunk of image. Here's a very nice illustration of that. Here's a picture of probably some

beautiful church on the inside here, it says white and gray, in some church, in some mountainous area in Europe I'm going to guess. And you'll notice that there's a function here that we're computing, that's a function of scale and here it is, and this is the maximum point. And this circle is meant to reflect the scale size of that maximum. Here we have that same image zoomed way out, computing at this interest point, so we've got the same interest point. So, notice that this interest point is just sitting at the top of this little chimney thing here. We compute the function again, and again, we take the maximum, and that's going to be this circle. And the really cool thing, is that this circle, and this circle, are the same components of the image. Those are the two scales to operate at. If we want to find descriptors that match, that's coming later. Basically, the scale of my detectors in one image should be big and in the other image should be small. Also the scale of my my descriptors and my detectors.

6 - A Good Function for Scale Detection

So I told you before that nobody actually uses average intensity. All right? So what will we use? Well, let's think a little bit about what it means to be a good function for scale detection, all right? Basically what you want is you'll be able to find these maxima as a function of the region size. Okay, so it would be bad if for large variation of region side basically the function stayed flat. That would be bad. Another thing would be something that wiggled too much. It's only three wiggles here. Imagine, there were a lot of wiggles because we'd like to find the maximum and we'd like to find really just you know, one or a really key maximum that we can pull out. So you know this becomes your ideal functions and it's sort of a, a, a cartoon of the function there. So, the functions that people use. What you want is one that, you know, responds nicely to contrast. Right? Because the idea is, you know, you're going to have edges at some scale. Oh yeah, remember that? Right, so edges occur at some scale. And if you were operating at the right scale you'd get this nice sharp contrast. And if you were, as you got away from that a little bit, it would fall, and if you got too far, it would fall down on the other side. So there's going to be something having to do with contrast, or finding points of high contrast. What are points of high contrast? Remember edges? Okay? So in fact, some of the detector operators that people use are just applying, remember kernels? And these are the kernels like when we apply differential operator kernels. We just apply a kernel to the image and use the output. Now one that people sometimes use and we actually talked about this. This Laplacian of Gaussian. When we talked about edge detection we said well, in the first derivative, there is two directions you could do. You could do an x and a y. And then we said, well if you have to worry about the second derivative there's all sorts of challenges. But we mentioned something called the Laplacian of Gaussian. The Mexican hat operator, right? And this was this one and the equation is written here. I actually pulled out my own MATLAB here and made a little 3D plot of what it looks like, and here this shows you. It's an operator which essentially is a second derivative operator, and we talked about zero crossings. Go back and, and, review the property of what we did on edge detection and you'll see it there. So that's the type of operator that you could use. Now, something that we didn't really talk about. Instead of doing this whole Laplacian of Gaussian which involves these sort of annoying second derivatives, there's a cute little trick you can do. And that's called a Difference of Gaussians. And that's what's written here, is basically if you take out of one Gaussian, and you subtract from it another Gaussian with a slightly different, or some factor different sigma, you get a curve. And what's plotted here in the red is the difference of Gaussian, and in the blue is the actual Laplacian. And you see that the values are almost the same. So if we're doing the work here, it's a lot easier since you're completing these Gaussians anyway. So just subtract things. One of the things also is that these Gaussian the the difference of Gaussians to solpacean is circularly symmetric. So it's totally invariant with respect to orientation. Okay so as you rotate your image, nothing changes. And it says that it's invariant to scale. What that means is if I, it's just like it's a scale and variant function before, if I blow up the function and then I make my operator that much bigger. I'll get back the same value that I had before.

7 - Key Point Localization

The way we do that for what's called key point localization and, and key point and interest points are used similarly, key point was the term used by David Lowe and Sift, we'll talk about that in a minute. Here is the general idea. We want to find robust extremus so maximums, by the way, you can also find minimum, so if you remember for the the Laplacian, it goes through zero, goes positive to negative. So we've took the absolute value of that, say the square, the value that's zero is a good point. Anyway you want to find robust maxima and minima in both space and scale. So that's what's drawn here. We haven't talked a lot about scale space. But the idea is, you know, I have an image, and then I could filter that image again and I could filter it some more and you can think of it as a coarser and coarser scale. And, I have some point. And you can think of that point as having neighbors to the left and right. And you can also think of that point as having neighbors up and down in scale, and that's referred to as scale space. A specific way of using that scale space was suggested by David Lowe in the SIFT paper, but at that point there was specific suggestion of. Do this difference of gallium pyramid? What I mean by that is you do it at different scales and over space and find maximum values in all of those. Now, sometimes you might long, land along edges. Okay. So your on a maximum value but it's really just a maximum along an edge. So what you have to do is you have to eliminate the edge points. In fact, if you recall we first introduced Laplacian or Gaussian to find edge points. Right? Even though it was a circular symmetric operator you're able to find these edges. So from way back, from the Harris who said well edges aren't so good for finding you know, specific locations we want it to be more like a corner. So basically what you would want to do is, you would want to find points that have these extreme values, then make sure that, you know, the thresholded, that they were ex, extreme enough, if you will, and then make sure they're not edges. So here is a particular method by which it was done and I, I show you this so you can take a look at the results in a minute. Basically what was done, was you take your image and you deal with it an octave at a time, and an octave is a doubling. So what you do is you make it blurrier and blurrier and blurrier and blurrier, then you take half, then you reduce that down to half the size, and make it blurrier and blurrier and blurrier and blurrier, and what you do is you subtract these off to get your difference of Gaussian images. Then, within each one of these Gaussian images, you're going to compare a pixel and in fact, if I go back to this picture here, I'm going to compare this pixel to each of its eight neighbors and each are in the same plane, and there are nine neighbors above it and nine neighbors below it. And if you're a maximum of that, you're going to say aha, I'm an extrema.

8 - Extrema at Different Scales

So, here you have a picture of Einstein. And each one of these is a different octave. So cut in half, cut in half, cut in half. And then you can see we go from crisp to blurrier, to blurry, to blurry. It's probably the only time Einstein was fuzzy in his life. And then, just as we showed in this picture before. We can subtract example these two, and get that difference, all right? Or we can subtract these two, and get that difference, right? So, these are our difference of gaussian images. And then what we can do is we can take each pixel in one of these and compare it to its neighbors in space and its neighbors in scale. Then when you do all of that, you can find the extrema and it looks like that. So, this is an Einstein picture with really bad dots. But some of those extrema might be higher than their neighbors but not very high. This happens in Colorado. So, you want to make sure that the contrast or that value is higher than a particular amount. So you say, ah-ha, here are places with contrast. But we say, wait a minute, we said before that when we use Laplacian and Gaussians, we might have nice contrast but it might be high just in one direction. So along an edge. So what do we do? We remove the edges and voila, these are our detectors, okay. And what's important here is these are detectors that are robust with respect to scale. Because we pulled out all the extrema at all the different scales. And this was David Sieff detector. So this can be written sort of like this. So for scale invariant detectors for SIFT, here we've shown that you're going to do difference of Gaussians in space and difference of Gaussians you're taking the difference of all these Gaussians and comparing them in space and in scale and you're picking the maxima. And here's a reference

from again the Journal of Computer Vision where he talks about this. There is another approach that's quite related. It uses both things that we just talked about. First, you essentially do the Harris corner detection at lots of different scales. And then you also take a look at the Laplacian in the scale direction. And you find points that are the maximum in space, in the Harris measure. Right, that's the amount of maximum suppression. And our maximum in space in scale in the Laplacian direction. And this is sometimes called the Harris-Laplace detector. The paper here ICCV is one that's referenced. And you know, I can show you here's a nice little demonstration that had, with change of scale that, well, first, you can see the original Harris detector, in fact, does really poorly. That's what we saw before. The SIFT detector, that's the thing in the green line. That was the thing we showed you before, it does way better than the Harris detector. But along comes Harris-Laplace, and it does even better. And so you might say, well, of course we should use Harris-Laplace. I will tell you a lot of people do. I will also tell you that this graph comes from the people who invented Harris-Laplace, so you know, your mileage may vary.

9 - End

What we've learned about today is, how to deal with images that have, large scale differences between them. Frankly, if you have two images that are translated, rotated, made brighter, darker etc. Stretched a little, skewed a little. The scale may not matter as much, but if you had a large scale difference, then what you do is you try to find the same scale around the interest points by that scale-invariant function, and you search for these maxima. All right. So where are we? So we know, how to detect interest points. Here's a whole bunch of them. The next question is how to match them. So I've got points on my left and points on my right. How do I find the match? To do that, I need what's called a descriptor. I need to describe each of those points by looking at the little region around it, and look for that in the next image. And that's what we're going to do next lesson.

4B-L1 SIFT descriptor

1 - Last Time

Welcome back to computer vision today we're going to continue talking about features, or interest points interest point operators key-points, in particular last time we talked in fact last couple of times we talked about how to find interest points, how to detect them. And we talked about the Harris operator and other kinds of ways of finding them. And here, for example, typical picture of finding interest points. In two images, and so this is like about a 500 points extracted with a Harris detector, and you can see that, well there are a bunch of points found. And what's good is that a lot of points, like here on this turret here, is also found on this one, there. So one of the things that we're looking for is remember we talked about reliability, that points that you find on one. Should be found on the other. But now the question is, given that we found points in two images, image one, image two. Image a, image b. Alpha bravo, take, pick your favorite. The question is, how do we match them? How do we determine which points in the left image are the same as points in the right or multiple images. So to do this, we need what's referred to as a descriptor. A descriptor is a description of the neighborhood. Where the interest point is, and then we're, that's going to allow us to match. Now, we need a couple of properties of the descriptors, and what's important is that these properties sort of trade off. The first is that we want these descriptors to be what we call invariant. That is, they should be. Almost the same from one image to the next. So why do I say almost? Wouldn't it be great if they were exactly the same? Well it would be great if they were exactly the same but also there were no duplicates. Remember that silly example I gave a couple of lessons back which maybe you just watched now or maybe you watch eight months ago. I have no idea. Where I said, you know, if we name everything George then it's easy to match them. Except not really. Because everybody matches perfectly. But you don't know which George is the right George. So the way we say that is, we also want our descriptors to be distinctive, all right? Distinctive means that. One point, you know, some particular point here has the same at the, corresponding location as the other

image has one descriptor. But a different point would have a different descriptor. And the reason that's challenging is because these two images are a little bit different, it means even the same point like the top of that Diet Coke cap that you guys can't see that Megan's been drinking because she drinks a lot of Diet Coke. Anyway ,. When I move a little bit, the image changes a little. So, th, the neighborhood is not exactly the same anymore. So the descriptor is probably going to change a little bit, but we don't want to change so much that I can't tell that they're the same, but on the other hand, there's another cap nearby. That I'd like it not to have the same descriptors so that they can be, you know, unambiguously matched. So we have this tradeoff between being invariant, found each time, and being distinctive, one is different from the next. So you might say, look I've got a solution. I've got a simple solution. We know that the Harris detector. Gives pretty good detection and remember we did that whole thing to talk about how to find the right scale. Right so we even know sort of the scale of the window. So, why don't we just do this? Why don't we for every feature we have here, we pick a window whose scale has been determined and we'll just do correlation against all of the feature points. Over here, using their appropriate scale, and then we just pick the best matches, and we go through and through that all the way. That would be a great solution, right? Well, no. If it was, then I wouldn't be here telling you all this. So why isn't this so good? Well, first of all. Just remembering about correlation. We have these two pictures and we might have a different picture over here, a camera over here, and the, fe, features could have rotated. Right? And correlation is not rotation in variance at all. Remember we talked about how. Harris detectors are very robust with respect to the de, the rotation for detection, but to do the matching, we have to be robust with respect to rotation as well. Correlation also is pretty sensitive if I made things brighter or dimmer, et cetera, I would actually change the values. So remember we introduced this thing called normalize. Correlation, well that's not really your savior either, because it sensitive to non linear changes. And actually very slight geometric distortions, can cause your normalized correlation to get pretty bad. So as I move and that bottle cap changes, it's appearance to me just a little bit. The normalized correlation could become problematic. And also this idea that I'm going to take one and I'm going to compare it against all of the others, and then the next one, all of the others. That would be very expensive, so we'd like to do something that's a lot smarter than that. So, this lesson, and the one next, is about, sort of, essentially overcoming these problems.

2 - SIFT

What I want to talk to you today is about what's called SIFT feature detection. SIFT stands for Scale Invariant Feature Transform. The motivation for the development of the, of the SIFT transform. And this was done by David Lowe back 99 is really when the, when the community first saw it and then early 2000s. The Harris operator was not invariant to scale, now we've fixed that later, and we talked about how to do scale stuff. And correlation was not invariant to rotation. So what David wanted to do, is he want to to do better matching, and he wanted to develop a, two things, one was an interest operator, a detector, that was invariant to scale and rotation. And we talked about the SIFT detector last time, right. We talked about these finding the maxima of the difference of Gaussians both in scale and in space, and that by doing that you could determine the natural scale, or I should say, the right scale. But the other part of SIFT, was building a descriptor, that is the description of the neighborhood that was robust with respect to variations corresponding to viewing conditions. So, as you scale the neighborhood where you found the right scale, if you changed the brightness a little bit as you're rotating those kinds of things, to build a descriptor that would be, robust under those transformations. And just inserting opinion here, for these days for SIFT and things like SIFT, the descriptor is the part that's used most. Okay. That's the part that the idea of building these local descriptors was really significant and continues to be used today

3 - Idea of SIFT

So here's the idea of using SIFT for doing things like alignment or recognition of objects. And the idea is this, you're going to say that your image content, whatever your image is, you're going to transform that into a representation. The word I like to use is a constellation, I'll show you one in a

minute. A constellation of local features and that those feature descriptors will be invariant to translation, rotation, scale and other sorts of imaging parameters. So if I take the object and I move it over and I get closer to it and I change the lighting, I would get the same descriptors. And so here's an example of shift descriptors, so you see we've got a truck here and it might take you a little while to really and in fact here I'll show you. These are these little neighborhoods. All right. And these little arrows are just showing you these little neighborhoods. Okay. So we've got ones by the tires, and by this thing, and back here by the logo. What might be a little harder for you to see, is that we have the same truck right here. All right. And the point here is, is that a bunch of the same descriptors. So here's one, and here's another. They were both found in the two images. All right? And if I can find enough matches, then I can figure out oh, this image has an object and that object is also in this objec, in this image,it's just has been rotated, translated a little bit, maybe scaled. And that's the idea about using SIFT for doing the recognition. So if your descriptors were robust with respect to these transformations, you could find them from one image to the next. By the way, here's another description of the problem. This one also comes I think directly from the original paper or, or the second paper. And suppose you want to find this box of crisp almond cookies, because we always want to find boxes of crisp almond cookies, in this basket. And what's really interesting about this grocery basket is it has basmati rice, crisp almond cookies, and the computer vision textbook, because when you go to Publix, you always buy a computer vision textbook. Anyway, the idea is, can you find this box that's actually over here? All right? And the picture that's shown in the original paper which is kind of cool, looks like this. Although cyan lines are the connection between SIFT descriptors found here and SIFT descriptors found here. This is the original image. This is the box located in the basket of groceries. So because it could find all those features it was able to say this object is really right over here. This was very impressive at the time, in fact continues to be sort of a strong way of doing certain types of computer vision recognition

4 - Overall SIFT Procedure

The overall SIFT procedure to recognition was originally defined to have sort of four components. The first two were finding these extrema these, those are those maximums the difference in Gaussians that we looked at. And to detect, you know, localize exactly where they were. So, these days as I mentioned this before, a lot of people use the Harris-Laplace method of doing this instead. Where you do Laplace in scale and Harris in space, whatever it is. Use some method to find your keypoints, and also to determine the scale of the keypoints in the image. Then what you have to do, there are two more things related to SIFT and we're going to do them today. The first is, we're going to define an orientation. That is, we don't know how the image has been transformed and of our little local neighborhood, we want to know what the orientation is. And then, once we have an orientation, we can think of that feature with a spectator orientation and build a descriptor. And by the way, I'm using the phrase here, Keypoint. Keypoint was the terminology used in SIFT. What we're talking about is these detected interest points and, and you're going to use them as keypoints, you're going to use them as features. Keypoint description is the descriptor we're going to use. All right. Here is a, a, a picture taken out of again the original SIFT work. Here is just a, a small image of a house on the left. And what this incredibly ugly graphic on the right is, is just. And, and we've got better ones in a minute, but this was the original. Is just there were all these little keypoints, and you can see these direction arrows. Okay? And those directions are going to be the little local orientation that we're talking about.

5 - Orientation Assignment

So, let's talk about your orientation assignment. Computing the best orientation, okay? Well, that's actually pretty easy. Remember the gradient and we had different operators for doing the gradients? So, what we're going to do is we're going to computer the image gradient out all these points and then, locally, we're going to compute some estimate of orientation. And the way we do that is really pretty simple. So nominally, this is your image area with a little line drawing and it just shown here. But the idea is, you're going to build a histogram. Okay, this is our little histogram where you've cut

the angle of orientation into some number of bins. In this particular case, they're using 36 bins of orientation. You know, which, what's the right number to use is, is a question. All right, and then what you do is you say I've got a peak, and I'm going to say, okay, that peak. That's my dominant orientation. And that's going to be the the orientation that we use. Okay? Once we've picked that, that becomes essentially our new north. Okay? We're saying that's, that's the way we're going to think about our window here. Now once we've done that, notice, we found it in x, y so we've removed the translation problem. That is we know where it is. We found it in scale when we detected the the, the interest point to begin with. We've picked this scale. And now we have a new orientation. So what that means is that whatever descriptors we build are going to be invariant to the transformation in x, in y, in scale, and in orientation. And that's what we wanted.

6 - Dominant Orientation Quiz

Here's the dolphin image we've seen before. Let's say our interest point detector found this tip of the fin to be a possible keypoint. We zoom in on a five cross five neighborhood around the keypoint. Now recall how you computed image gradients. Say using a three cross, three filter. When we place it over this pixel, what is the dominant orientation you see? Right. It's along the diagonal going towards the top left. Let's use this approach to mark image gradient directions for other pixels. To avoid edge conditions, we only consider locations where the filter fits inside the image patch. Also, we approximate or quantize to eight directions for it to go horizontal and diagonals. Can you figure out the dominant orientation of this patch using the image gradient directions? Select the appropriate radio button.

8 - Keypoint Descriptors

So now, the question is, we need to compute this descriptor about this local image region, all right. And remember, we want two things for the descriptor. We want it to be highly distinctive, that is, given two different points in the image, I should be able to come up with two different descriptions. But, we like it to be invariant as much as possible, to just small changes in viewpoint, maybe even large changes in viewpoint. Sir, illumination in other kinds of imaging parameters. So, before we build that descriptor we normalize. What's the normalization? Wherever our little window is, we're going to say, okay, this is our new north. We're going to rotate north to be north, to be up let's say. So we're going to move them, rotate them all back. And the other thing is that we're going to scale it to be some uniform size. So if the, if the, when we did the, the detection to begin with, it said it was a small scale we're going to blow it up to be that size. And if it said it was a large scale we might shrink it down to be some fixed size. So the idea is we you know, pick particular sizes. By the way, getting SIFT to work is painful. And that's one of the reasons people tend to use libraries for descriptors. it, because it's a lot of you know the, the expression, the devil's in the detail? There are a lot of details to makings this work.

9 - SIFT Vector Formation

So here's how we're going to build our descriptor. We're going to build a feature vector. So there's going to be some vector of some length. Okay, and it's going to be based essentially upon properties of the histograms of gradients, all right. So what I'm showing you here is, this is a, a nice picture nomolist, this is pretending that we've got a little grid of 8 by 8 pixels. I'm only showing 8 by 8 because to show you more, would be, too hard. Okay? And, what we're going to do is we're going to take this histogram of gradients, all right? And we're also going to weight it by a Gaussian. So sort of this circle that's drawn out here and then The idea is that we're going to, if, if our center point if located in the middle, we're going to take the gradients in the middle to be weighted more than the gradients toward the outside. So, what we're going to do is we're going to build these little histograms. What we do is we take, say 4 pixels by 4 pixels. Can I count to four? Yes I can. Okay, and we take a little histogram of those orientations and that's done right over here. So this thing has sort of eight different dimensions. And then this four pixels here would be. There. Now, I'm only

showing this as a two-by-two, right? One, two. Well, actually, for real sift, they do a four-by-four, two-by-two just shows better. So just to think about that, we have a four-by-four away. That's 4 times 4, that's 16. Each histogram has eight bins. So that's 16 times 8 that's 128. That's 128 numbers, okay? Those 128 numbers, that's the feature vector that we're going to use to describe this this single key point. This is the descriptor that we're going to use. Because you take these 4 by 4. And now I'm going to try to draw them out and it won't look quite so good, but you'll see what I'm talking about. Okay, so I have 16 different little histograms. Because I stacked them up into a feature vector. I don't add them together, I stack them into a feature vector. There is a certain amount of spatial sensitivity, right? The top left hand, let's suppose the top left hand histogram is the first part of the feature vector. Right? All the way down to the second, the third, the fourth, finally the, the last part on the bottom. So there's a certain amount of sensitivity to the spatial layout, right? So if you've got some orientation of gradient in the top left-hand corner and different on the bottom right-hand corner, you get that spatial layout. But it's not a very strong sensitivity to the spatial layout. Which is good because and that maintains a bunch of our viewpoint invariance. And remember, we have unrotated this thing, with respect to the dominant orientation locally, okay. So our, all of our histograms that are being put down are placed with respect to this normalized orientation and scale.

10 - Smoothness

So now we have this 128 vector. And also remember we've been smoothing by these Gaussians so the middle has more weight. In fact, and I mentioned this before in order to make these things work, we ensured smoothness. In fact I think in, in Davy Lowe's work, they originally talked about sort of trilinear interpolation. What they meant, was that. Points get smooth and they contribute not only to bins they, they actually overlap the histogram bins a little bit. And that way each pixel corresponds multiple times, is, is used multiple times. In different gradients and also they, they vote in to adjacent orientations. The details don't matter, so much. But what it means is, is that if you get a small shift in where this neighborhood of eight by eight, or 16 by 16 pixels lands. You don't get, a large change in the scriptor. And that's really important, because when you re-localize a point from one image to the next, you're not going to land, you may not land in exactly the same spot. So the smoothness allows you to get a very slow change, in the descriptor as you move just a little bit. And that's really important. This also is why I said it takes effort to make these SIFT libraries function. One of the things you don't want to do is remember. The magnitude, of the gradient. I mean I'm not sure I said this. That the, that it's weighted by the, the gradient, it's actually a weighted magnitude gets used in the histogram. Well, a problem with that is, if something has very high magnitude. It's going to dominate the votes, or it's going to dominate the descriptor in a way you might not like. And so, one of the things that they advocate doing is after you rotate your normalize the rotation, clip the magnitude of the gradient to some maximum, okay? So you don't let it get more than a certain amount. Talks here about point two, the which one you use is a function of your know imagery but the idea is that it gets clipped. The second thing that you do in order to prevent any sort of illumination change from really, harming you is that you take this 128 long vector and you normalize it to be magnitude 1. And that way if, if more gradient vectors show up because you scale everything. Essentially things are better behaved, because you're going to be comparing these descriptors one to another. And you want them to be, sort of a similar magnitude. So by normalizing to one, they're all exactly the same magnitude. And it's just the distribution within that vector that gets used to, to compare them. I suspect that has function of exactly how you do the comparison. If you were doing dot product kinds of things, the magnitude normalization might matter a little bit less.

11 - Evaluating the SIFT Descriptors

One of the really nice things about some of this initial work is, they did some pretty extensive evaluation. They would take a whole bunch of images and they would change their, they would rotate them, they would scale them, stretch them, change the brightness, contrast, and, and add

noise. And then the feature point descriptors would get run on the original image and the the, the transformed, the distorted image. And then what you could do is you could say, under some matching criteria, how often does the point, does this descriptor. How often is its best match in this whole other image, the correct one? The correct one being actually comes from the same point, on the same object, in two different images. And that's what they evaluated. So, here's the first graph that we'll, that I'll show you. again, the error measure is the correct nearest descriptor percentage. How often is the one that matches best the correct one, all right? And then they varied a couple of things. The first thing that's easiest to see here is this number n. This was the number of those of the histogram that got used. Remember, I said, I was showing you only a two by two, but they actually used a four by four. Well, one of the reasons they used a four by four is a two by two is here, a three by three was there, a four by four is here, and all of a sudden, you're sort of maxing out. So, it looked like four by four was, was a good thing to use. By the way, this was evaluated by looking at a bunch of images, and 40,000 key point descriptors, you know, a bunch of images. So, they looked at 40,000 different descriptors and the corresponding other image with all those perturbations, and so it's, it's pretty serious evaluation. The other thing that's shown here is, how many orientations in the bins, right? How many, sorry, how many orientations in histogram, how many bins? So, you could have used just four. You could have used eight, you could use 16. Well, what you can see here is four does that. When you go up to eight it gets much better. When you went to sixteen, you didn't do very much better than eight. So, they picked eight. All right, so this is an empirically motivated set of parameters. Without going into too much detail, they also show, this is as they add image noise to the picture, how much percentage of pixels get noise or, or the magnitude of the noise. And you can see that the correctly matched value, again, falls off nice and gracefully, which is what you're looking for. You're looking for it slowly. By the way, going back to this previous one. You know, it's only getting up to like, 50% match. We don't know, we don't need everything to be matched correctly. We need enough of them to be matched correctly. And we'll assume the ones that don't match are all voting for Santa Claus, Mickey Mouse and Donald Duck, if you remember that analogy, right? The ones that don't match, match sort of arbitrarily. So, as long as we get ones that match well then we're doing okay.

12 - Experimental Results

Here's a very nice table of some experimental results. Just shows you the different kinds of perturbations that they did. Okay? And they're talking about different ways of comparing the matches. About whether you just, you know, do you get the right location and scale? And what about orientation? The thing that really matters is that you're doing all these different things. You're looking at changes in intensities, rotations, scalings, stretching 10% noise. And you can see if you do all of those things, depending upon whether you're looking at just location scale or including orientation, you still get above a 70% match. And that's pretty remarkable. In fact, there's a nice figure that comes from here. So, here's an original image, and drawn on on top of this pic, in fact you can almost not see the picture underneath. There are all these feature points, okay? And, what they did is they took the picture and they rotated it 15 degrees. They scaled it down 90%. They stretched it 10%. Okay another, an extra 10%. They changed the brightness down by 10%, changed the contrast, multiplied out, and they added some noise. And then they went back and they took a look at all their feature points. And how well did it match? Well I love this little animation. See the whole thing flies in here and says 78%. That's pretty cool. What that's showing you, is that you have this very robust descriptor, and yet it's distinctive enough that you can find the correct matches. Here's another example again from the paper. What you're looking for are these little things. Okay, so this is a little piece of the house this is a little piece of the totem pole, this is something else that was on the totem pole. In fact, it's a little hard to see, so I put these little colored dots. So what they're look, so we're looking for these patches over here. What you're going to do is you're going to create little SIFT features on all these things. And you're going to look for those patches in the corresponding image. And what are you going to find? Voila! Basically, the thing works, which is not a surprise. It came from the paper where they're claiming it. By the way SIFT is actually intellectual property owned by the University of British Columbia. So before you go

implement it and put it in your iPhone in order to do something clear. Make sure you hire a really good lawyer.

13 - End

That ends this lesson on SIFT descriptors. As I mentioned, getting SIFT to work can be painful, and that's why those of you who are taking the OMS version of this and working on a problem set using it, we'll be giving you some libraries for computing the descriptors. You're going to be on your own, for finding the points, finding the interest points, and figuring out maybe the scale or maybe we'll just fix the scale but the idea is that then you can use the library to do the descriptor. Then, you're going to have to do all this matching. Okay? Now, you know, matching, if you think about that, suppose you've got a thousand descriptors here and a thousand descriptors there, right. That's a million combinations, if you're just trying to compare this one to every each point here to every point there. If you've got 10,000 that's 100 million, that might not be really a very good idea, so you might want to do something smarter. And, what would it be to do smarter? What would be really smart would be to watch the next lesson, so you can learn how to amaze your friends doing this.

4B-L2 Matching feature points (a little)

1 - Last Time

Welcome back again to computer vision. Today we're going to continue a little bit, on we did SIFT feature descriptors last time, descriptors in general. And today, we're going to talk about matching them, but we're only going to talk a little bit ab, abou, matching, because actually there's only a little bit you need to hear. So, okay, so where are we? We, we know how to detect points, right? So we detect, feature points. And we now know how to describe them, this little, notial picture showing you that we get the little feature descriptions, using, the, the, the sift, meth, method we talked about where you get a 128-long vector as a descriptor for each one of them. So now the question is, how do we go about matching them? Which is kind of a, sneak here cause I started off with this slide last time that said we were going to do matching, we didn't do the matching, we just did the descriptor. I'm not even going to tell you about matching again, I'm going to tell you about how to search for a match this time.

2 - Nearest Neighbor

So we already said you could just do nearest neighbor, right? I got a point here. I key point here. I search every key point over there to find out which one is best. And by the way as I mentioned before, those of you doing the problem set will probably end up doing just this. But as we pointed out that could be really expensive. You might have thousands of points here, and thousands there, and that's awfully lot of combinations. So we're going to do something a little bit smarter, okay? We're going to use a approximation to a nearest neighbor search. And the idea is that we generate hypotheses from an approximate nearest neighbor, matching for each of the vectors. And it's essentially in, SIFT uses this algorithm by Beis and Lowe called best-bin, and it's a modification to the k-d tree algorithm. And I'm going to assume those of you who are in computer science, that you have some familiarity with what k-d trees are. It's, it's essentially you can think of it as in a binary world, just binary splitting things, k stands for k-dimensional. Tree where you, you split them along. It allows you to access points in essentially a a log n instead of a something proportional to n. And they did a little modification of that. That allowed you to sort of sort your bins that you're searching for quickly. And two points are important about this. One is that it is. A pretty good algorithm for the SIFT search. And it's reported anywhere from two to three orders of magnitude speed-up. And anytime you can buy two or three orders of magnitude speed-up, you should do that. Now of course, you can't cheat the piper. You know, nearest neighbor is certain algorithm in

complexity. By doing it this way, you find the nearest neighbor, the one that actually matter. On approximately, again currently they'll tell you 95% of the time. So, you have to wonder, you know, exactly what it is. It can't be guaranteed but the idea is that for sort of well behaved systems you're getting in the 90 plus percentile, which is good because if you remember before, we said the descriptors aren't perfect anyway. So some of our matches are going to just be wrong. So we're finding 95% of the right matches, that's going to be pretty good. This figure here sort of shows you a little bit of what's going on. Again, this comes from, from their paper. The algorithm is called Best Bin First. The idea is, if this was your probe point, and you found this point D when you do normal k-D trees. You essentially recourse up the trees and, and, and branch out looking for nearest points, and all that they did is they had a way of sorting through the bins. So they would order the bins themselves, so they didn't just recourse through the tree. And it gave them this speed-up. And this listed here is the, the reference to the paper, and it just makes it easier to find the nearest neighbors. To, usually find the nearest neighbors.

3 - Wavelet-Based Hashing

Another cool thing you can do is do some form of a hashing based method in order to index. So, what I'm showing you here is what's called Wavelet-based hashing done by Brown, Szelski, and Winder in 2005. It comes from Szelski's book. you, you can see it. And, and basically the idea is this, so here's our little neighborhood. Okay? And we apply what are called these Haar wavelets. Now wavelets are just a self-similar form of filters, may be applied different scales. We didn't talk too much about them. That's okay. But basically these are the three filters. And these filters are, you know, sum up plus here, minus there. Plus here, minus there. And then plus here, plus here, minus there, minus there. Okay? So, what you do is, that means you'll get three different numbers, right, from each of the three filter outputs. And what you do is, you can then quantize each of those into ten bins, or maybe ten overlapping bins, all right? So, you have 10 bins 3, that means you have 10 to the 3 different possible values that you can get out for a single application to the neighborhood. And because the bins overlap, you don't have to worry about well, maybe I got out a 4 vs a 4.1 for some value, because the bins don't have a, a sharp border. They have an overlapping border. So then what you do is, you compute these three numbers quantized to over 10. That gives you 10 to the 3 possible bins to pick from. And you go and you get that. So that has, that reduces the set of neighbors that you have to look at, or the set of key points you have to look at by a factor of something less than 1,000, because overlapping bins, but on the, between, again, between two and three orders of magnitude. Pretty cool.

4 - Locality Sensitive Hashing

This is related to the general idea of what's called locality sensitive hashing. All right? So I'm going to assume that most of you, as computer science students, you know, what hash tables are and what hash functions are. And if you don't, you may not understand this. Normally a hash function maps some data structure. Into some nicely distributed space. So you might have a function that maps, a structure to, integers between one to a hundred, Okay? And you can think of those integers as bins. If you give me some new data structure, and I've already seen one of these before, I map it through that function, and I go and I look at the bin, and I only search through the ones that are in that bin, okay? So, if it's distributed kind of evenly. You're going to reduce your search work by a factor of 100, all right, because you this function and, and you reduce it down. But the problem with hash, not problem, the design of hash functions is that they are very precise. They're designed for precise data structures. Locality-sensitive hashing works similarly, but it's a little bit different because it takes the notion of distance into account. And the basic idea is you're going to construct some hash function. Here it's written as g. Okay? And it's going to map onto some output U. All right? And the idea is, and there's some like Greek written here, et cetera. But the basic idea is that if you have two points that are nearby in the original space by some distance measure, then the probability that those two points will land in the same bin should be pretty high. Likewise, if I've got two other points that are far away in the original space, the likelihood that they land in the same bin should be

relatively low. Okay? That's what locality-sensitive hashing is. And when you do that, it means that I can compute this hash function. Go to whatever bit it sends me. And the idea is that. Descriptors that were similar, not exactly the same, but similar will be in the same bin. There was some nice work done by, Kulis and Grauman. The title is Kernelized by Locality Sensitive Hashing for Scalable Image Search. There's a lot of work on kernel functions. We'll talk about them them later. This is not the kernel of a, filter. Sorry, they use the same name. But the kernel functions here are a way of mapping from some dimension, some higher dimension. And then operating on them. And this is a approach of how to do. Applying these kernels to sift features. What. What, what's important is it's the nicest discussion in there about doing this locality sensitive hashing.

5 - 3D Object Recognition

So to wrap up let's talk little bit about doing some recognition using our SIFT features. So basically at training time, you I give you some object. You somehow have to get hold of the, just that object in the image, okay? So what's the easiest way? Well, you put it against the black background. Okay. So you find the outline, everything inside there is the object. And you compute your keypoints, you find your little SIFT descriptors located all over the object. And in here, you find these outlines, okay? But again, there'd be little SIFT features all over them. The outlines are what we're going to overlay when we say that we've found the image. Okay? So at test time, what do we do? Well, we take the keypoints from our test image and we compare each one of them against, let's suppose we're looking for, I don't know, telephones. So we take the keypoints of the telephone, we try to find the best one that matches. Exhaustive nearest neighbor best bin first locally sensitive hash, whatever it is. I try to find the best one and I compare a whole bunch of them. Here comes the trick. Now I want to find a consistent set of matches, okay? So, what that means is that I'm going to do the following. Let's suppose that I believe that the difference between this picture and that picture is an affine transform. Okay. Now you're scratching your head to try to remember that whole affine transform thing, because I'm about to ask you a really hard question. How many pairs of corresponding points do I need for an affine transform? Then answer is not four. Four was a homography. One for translation. Two is similarity, translation, rotation, scwale, scale. It was three for an affine. So an affine will let you do a translation, rotation, scale and a shear. So if I find three good points here and three good points there, that allows me to compute my affine transform. So then what I can do is that tells me where every point from the left image should appear in the right image. So I can now check to see if I take all my key points from my teddy bear, do they all land at matching keypoints when I apply that same affine transform? Okay. We'll actually spend a lot more time talking about how to search for transforms over matches on the next lesson. But for now, you can just think about doing it sort of this brute force way.

6 - Recognition Under Occlusion

So, how well does it work? Well, here's a nice example of finding both the telephone and the shoe. So here is the original phone and here you can see the outline. And here are the modest number I don't know one, two, I don't know. About 15 key points that have been matched. Now remember, you only need three to find the affine, so by finding more than three, you get a much more robust solution to the transform. Likewise, see so here we have our green shoe and here you can see where it's found. And again, you can find the set of feature points, key points, that were matched. But's also pretty cool. I only need to find three points from one object in the corresponding place, and then I can predict where the rest of them go. So that means, I don't have to see the whole object. So here you see an example. Here is the shoe. I'm going to draw inside so you can see this right. But you'll notice it's being occluded by this box of whatever the groceries are. Okay. Only the front part and the back part of the shoe were found. All right, but I only needed three matches to find an affine transform, so I didn't have to see the whole shoe. I only had to see enough of it to do what's called the alignment. Same thing is true of this telephone here. The whole back end of the telephone is missing, but we're able to recognize it by just finding features within the front part. That's pretty impressive robustness with respect to occlusion. Here's another one. Now you can see

the again, David had this thing about teddy bears, shoes, and telephones. I'm not exactly sure why. Anyway here you can see the shoes behind the teddy bear, only the front part is visible but you can see it. Same thing with the phone. So the idea is that as long as I can find these, a small number of these local matches, I'm able to do the recognition. oh, and here's one you, you saw before. Remember the, the, the totem poles? He's at UBC, University of British Columbia. They got a lot of Native American, Indian stuff up there, so I'm guessing that's what this is. And you remember the thing just works.

7 - SIFT in Sony Aibo

There was a more recent application of SIFT. Very explicit. You guys, I don't know how old you are anymore, but maybe you remember the Sony AIBOs, the dogs that used to go around and would recharge themselves and live in your house. It was sort of an experiment that Sony did for a while. It was kind of successful, but eventually they stopped making them, because I think, I think they were learning from it and moving from there. Well, one of the things that the dog could do, well there were two things. It could recharge itself and you could show it cards and it would do the commandment based upon what was draw on the card, what was printed on the card. And the way it did that was using SIFT features. All right? So the charging station had a pattern on it, and as the dog walked around and just did kind of whatever it did, whatever little robot dogs do, it would see this pattern. It would be oriented, it would be translated off to the side somewhere at a different scale. But remember, SIFT is translation, rotation scale, and variant. So it could find the charging station. Also, they had these what were called visual cards. And they were things like this. And it doesn't really matter at all what's on the cards as long as they have gradients distributed all over them. The dog can recognize which card you're showing it. You know, I said it doesn't matter at all what's on it. It does matter a little. It has to not look like anything the dog would see otherwise. So if you wonder why do these cards look so strange, it's because it can't look like my dining room table. Because if it looks like my dining room table, my dog will look see my dining room table, and roll over and play dead or whatever robot dogs do.

8 - End

So that ends this lesson on finding, matches for SIFT, descriptors. And it's really all I've really talked about was some clever indexing and hashing methods that have made it, much more effective as a way of describing and finding, matches. As I said earlier, if you have a really fast computer, you might just do full nearest neighbor. Or, if you had all day to do a problem set, you might just do the nearest neighbor. But if you actually have a real time device that needs to do something a little more clever, then you need something that accelerates that pace. Going forward, what we're going to talk about is to use a small number of local matches, to determine a best global solution. And this should remind you of something we've done before, where we take, sort of, a modest number of local events and find a global solution. Remember that whole voting thing? Well, what we're going to show you is another way that's more appropriate for doing things like I've gotta sum SIFT features. Some of the matches are good, some are not so good, I'd like to find the best global, match, what's the way of doing that and we'll talk about that the next times.

4C-L1 Robust error functions

1 - Intro

And welcome back to computer vision. With today's lesson and the next we're going to close a circle that we started it seems like forever ago. I remember we were talking about trying to compute transformations between images, and we talked about doing panoramas and mosaics, and o, and other ways of doing this, and we said, and also when we're talking about fundamental matrices. Key to all of these things was if we knew the correspondences, you could compute the various

transforms whether it's or something else. Then we talked about competing features, and feature points detections, and descriptions. So today we'll finish the story, where we're going to actually do the approach for doing feature-based alignment. We'll describe it as a form of model fitting, since, really that's what you're doing, we'll show you more about that. And then, we'll talk about a really cool method that let's you effectively find the best global solution, or probabilistically the best global solution, without being too exhaustive, too combinatorically challenging.

2 - Feature Based Alignment

All right, so let's rememb, remind ourselves of the overall strategy for doing feature-based alignment to affine transforms. We know how to compute features. That involves both detection, remember all that Harris, that Harris Laplace stuff, for finding feature points, and for generating descriptors. We talked about SIFT where we generate descriptions around the little local neighborhoods of the points. Second, we know how to find some pau, plausible, or what we're soon going to call putative matches. We talked about doing the vari, the variation on Kd-trees, the Best-Bin hashing methods. Basically ways of saying, well which features are likely to match which other features? I might make some mistakes, but at least they get me some, some reasonable matches. Then we would have to compute a transform that aligns one image to the next. So for we talked about doing homographies or affine transforms. And then finally you would apply that transform. Great. All right, so the algorithm follows this pretty carefully, but you'll see in a minute, we're going to get to a point where we don't, there's a step that we don't know actually how to do. So the first is extract the features, fine, we do that. Second, is compute these putative matches. That's the term that's used. This is that, what I was talking about doing this kd-tree or the best bin. And what you see drawn here, all those red arrows, those are going from some feature in the left to the best feature matching it on the right. Now if this was nice, if this, if everything was right, you'd see this nice smooth transition. But of course what you see here is a lot of stuff. In fact you can see, you know, some of these things tend to be all lined up the same direction, but here's one right, it goes from here to there. That is clearly, I think the technical word is, wrong. So what we'd like to do is, somehow figure out a solution to this that makes it clear that we're going to ignore that one and pay attention to another. So after we find the putative matches, then we have the loop until happy routine. Okay, so loop until happy says, hypothesize some transform T from some of the matches. Actually, this is what's going to be interesting. This sum of the matches, okay? And then we're going to verify the transform. We'll talk about what that means in a minute, and we keep doing this until we think we found a solution that's really good. That's what it means loop until happy. At least today and then you apply the transform.

3 - Feature Matching

So how to get the putative matches. We talked a little bit so far about essentially doing exhaustive search, where you, you know, compare one against every other and you, pick the one that's best. We also talked about hashing and nearest neighbor, techniques. But. You now find a best match. How do we know it's a good one? Okay. So, in this next lesson, not today, we're going to talk about looking at the global solution that's suggested by this match and using that to prune off some, some of these, but even before that, there's something we can do at the local level. And here, here's the idea. So, we don't want to. We're not going to match everything. We're only going to take, match the ones that have good enough matching. All right? So, how do we do that? Okay? Well here would be a simple idea. You find the best match. And you take the correlation. A little patch from here. A little patch from there. And you do it says here SSD. Sum of squared differences. Remember where we talked about that? And we could say, all right, well if that number is less that a certain threshold, we're going to accept that match. By the way, something that I've said before and is implicit here and we'll say it again later. The general idea is I need to find enough good matches. Not all of them. And I need to not find too many bad matches. So the goal is to try to get a bucket of good matches with only some bad matches thrown in. So, going back to the SSD, the

question would be how to set that threshold. So, empirically, you can start to take a look at this. All right.

4 - Feature Space Outlier Rejection

So lets suppose you have a whole bunch of matches and since you, the human are going to click on which one's are correct and which one's are not. Well, actually not you. This is what you're going to do if you're a professor. You're going to find a bunch of undergraduates, okay? And you're going to offer to pay them not very much money and then you're going to offer to give them pizza. Okay? You can get undergraduates to do anything for pizza. Turns out you can get graduate students to do anything for pizza and Coca-Cola. So anyway, so you get a army of students to click on which matches were correct and which matches are not correct. So after everybody's clicked on all these matches, you can actually generate two graphs. What you're going to do is you're going to measure the nearest neighbor squared error. So this is how good is the SSD of the nearest neighbor? And we're going to check it for all of the correct matches and all of the incorrect matches. And so incorrect matches, remember, are where the best match is actually not the right one. Okay? So we're just plotting the best matches, but the correct, the ones that are correct and the ones that are incorrect. And so here is the squared error of the correct matches. That's what it says here, correct matches. All right. And you'll see that it falls off nicely. So most of them tend to be small and, you know, modest amount of them tend to be large. So you might say, well, I'll just set the threshold here and that'll be fine. Well, except not so fast. If you then take a plot of the incorrect matches, you see that the incorrect matches have a whole bunch of stuff early on also. So what that means is I've taken a point and I've found it's best match and the best one is not the right one. So it's not a surprise that it's a lowish value, because this is the best match. The problem is that some of your best matches are correct and have a lowish score. But some of your best matches are not correct and they also have a low score. So it's not so clear where you, where you should set that threshold.

5 - Lowes Better Way

So in David Lowe's work in doing the sift work, he had a pretty clever idea, and this is the idea. Take your first-nearest neighbor, so you do the best match. See what its score is. Now we already said setting a threshold just on that is challenging. But here's what he's going to do. Take a look at the next-best match. So I now have the first nearest neighbor and what's called the second nearest neighbor. And, take a look at the ratio of the best to the second best. Now if you think about this, [COUGH] before we get to the plot. When you're a correct match, the best is really good and it's the best one. And the next best one, well. You know, presumably it won't be it, we know it not going to be as good, and how, however much it falls off will depend upon sort of where in the image there happens to be another point that just happens to look like this one. So, you're going to get that ratio, all right? But now consider a point that is not the right one. So it's sort of randomly, maybe its real match was occluded. It's not actually in the picture. So it randomly went and picked some point, and it happened to be its best match. Its second best match, is going to be some other random point in the picture. So those are going to be more similar in value than in the correct case. And that's exactly what this graph shows. This is the ratio of the best to the next best, okay. And what you're seeing here is for the correct matches the distribution is over here, okay. So, that ratio of error, okay? Is, you know, a lot of this stuff is over here. Right, so like this 0.1 would mean that the best match SSD is sum of square differences which you want to be low, is 10% of the next best match. So that's, that's really a big difference, here it's 20%, 30% for the incorrect method matches. Remember it's randomly selecting them you hardly get any that have such an extreme ratio. And in fact you don't start seeing them show up until you get to this much higher value. So setting a threshold here, you could set a threshold, you know, right about there. And almost all of your matches, your best matches would have been correct ones. And very few of them would have been incorrect. Now you might ask, but wait a minute, you're going to throw out some correct matches that way. Isn't that a problem? There's no way Megan's going to be able to remember that entire question. So you might just say, well, aren't you going to throw out these here, that are the correct

matches, but the ratio didn't pass your test? Yep. So you're going to be throwing away some correct matches, but our assumption is that we've got plenty of correct matches and we just have to figure out which ones are there. Okay, so that's this cool method of doing this selection by comparing the best to the nearest best. Cool? Good.

6 - Feature Matching Problem

So now we have some more another trick under our belt that allows us to do this feature matching whether we're doing the or really, in this case, throwing some features away. Match selection, whether we're doing exhaustively hashing or some other method. But remember that distinction we talked about earlier where we said, well, we've got two competing criteria, goals for feature descriptors. We want them to be invariant, that is we don't want them to change very much from one image to the next at the same scene point so that I'll be able to do the match. Which means you want them to sort of, the description to sort of stay the same. But what you don't want is if I've got two similar points in the scene. You want them to have two different descriptors because I don't want to match to the wrong one. That was this notion of being distinct. So we have this invariance versus distinctive. And that's the fundamental trade off. And because of that trade off you have this problem that when you pick the best match, because you've done this tradeoff, you are still going to get a bunch of what we'd call wrong matches or outliers. That is, we have some point matches that actually fit what we want and some of them are just wrong. So the question is, what should we do? All right. So, this sort of lent, launched into the last part of this, and then hopefully, naturally, because I'm such a spectacular teacher into the, the next lesson, all right. So, remember, why are we doing this? Well, we're trying to compute a fit, a model that aligns one picture with the other, right. We're going to compute a model that is the relationship between entities. So what we're really doing is model fitting. Okay. Now, remember this picture of model fitting of lines and circles. Remember when we did that whole Hough transform thing? Where we were saying, Well, we've got a bunch of points and we'd like to find which line is the right line, or which circle is the right circle? Right. And the idea was that some of our edge points come really from the circles, and some of them don't. So really, that's what we're doing here. Is we're doing model fitting.

7 - Typical Least Squares Line Fitting

In order to motivate, we're going to go let's do the math behind some simple fitting that you, that you already know. I'm talking about typical least squares line fitting. All right, so in least squares line fitting we're going to assume that you know, we've got some data points. X1, y1 up through x n, y n. All right. And, we're going to use that equation y equals m x plus b because it's just easier to show what we're doing here. We had done that whole polar coordinate thing for Hough transform, we don't need to, to do that here. So, least squares line fitting would say find the m and b that minimizes this error function, okay. And this error function, which says I've got some y and I want you to find an m and b that minimizes the difference the y that was actually measured and the y that would be predicted based upon the x value. Okay. And that looks like this in the picture. All right? So the idea is that the distance that we're minimizing, is the this vertical distance to the line. How do we do that mathematically? well, it's pretty straightforward. We take our original equation, and all we're going to do is we're going to essentially vectorize and matricize it. I know vectorize is a word. Matricize should be a word. It's just like a really great word. So, in today's day and age, I'm going to trademark that, so if any of you uses it on Twitter, you have to pay me money. All right, here's what we're going to do. It's pretty simple. So, this was our original equation. Okay, where what I've done is, I've turned this component into vector multiplication. All right? And then, because I'm doing this sum, okay, over this squares, what I'm going to do is, I say, okay. I take this vector minus these squares. Okay, that gives me what? That's another vector that is just the deltas between the y and the prediction. And I say I want the magnitude of the, of the square magnitude, which would just be the sum of the squares. So that just gives me this very simple expression. So I go through all this mess to get this very simple expression, y minus Xh. Where the y here is this y vector. The x here is this matrix of the x and 1 for all the rows for each of the points. And then this

h, I just use h to be the parameter for m and b. So what we have is this y minus Xh squared. All right? That's our error. And what we want to do, of course, is minimize that. All right? So I just expand that out. That's what I have here on the bottom. All right. Group the whole thing. I've now got this equation there. So E is equal to that. Just copy that error function over again. And we're going to take the derivative of that error, with respect to h, and remember h was just the little vector and b. If you don't remember how to take the derivative of a quantity with respect to a vector, go remember and look it up, all right? Transform that, get to this very simple picture, okay? And what you have is h is equal to this quantity times y and that's the pseudoinverse. Right? So I hope you remember that from your linear algebra. Basically this is just the standard over-constrained least squares solutions. So the model fitting has something to do with minimizing squared error and when you work that all out you get to the pseudoinverse least squares solution.

8 - Total Least Squares

There's actually sort of a problem with vertical least squares. And the reason I called it vertical least squares is we're actually minimizing the error in a particular direction, right? Vertically, with the y. So for example, that's not even rotation in variance. So if I get lines that are kind of horizontal, that works okay because, you know, the points. But I got lines that are very steep, then as I move off the line a little bit, I end up with this huge error, which is not really right. I just moved off the line a little bit. And that's because I'm using this particular direction. And of course, it fails totally for vertical lines. So it's really not a good model, it's not a good noise model for what we're doing. So let's develop a slightly better model. We're going to refer to this as total least squares and this you've, you've also seen. So now what we want to do is we want to minimize the distance between the point xi, y and some line ax plus by equals d, where a and b are this unit normal. Right? So, it's actually the unit normal in the direction towards the line. So when you take the dot product of the point, xi, yi and a, b, that would give you this here. Okay? That should equal d, which is the distance of the point, of the origin to the line. That's this distance here, as it's drawn there. And it, to the extent to which that point does not lie on that line, that perpendicular distance, as drawn here is just ax plus by minus d. Right? It's supposed to equal these, so if you subtract d, that's the dot product and that's the perpendicular distance. At the end of the day, what that does? Is this gives us this new error function, E. Pretty straight-forward. And what we're going to do is we're minimize that error function now instead of our previous error function. How do we do that? Pretty straightforward. Same thing, we start with out E. All right? We're minim, doing a minimization. We take the derivative with respect to the parameter d, that's the first one we're going to do. That generates this equation for us. We can use that to solve for d and d is just this beautiful ax bar, x bar is the average of the x's plus by bar. That's, that's what these terms here are. Okay. So this, now I've got an expression for d in terms of a and b. So I can substitute that back in and I get this kind of ugly expression that we'll make beautiful in a minute. So I've now plug these in, so there's no more ds in there and look at that. That now looks very similar to the expression we had before, except now we've got this these means subtracted. So if I stack up all my x's, x minus x bars here and y's minus y bar there. And I'm actually going to call that matrix U because I do. I can write this whole thing as Transpose Where h now is just this parameter a, b. Remember, we got rid of d. We solved for d in terms of a, b and now we just have to solve for a and b. Okay. So now that we have an expression for E in terms of U, which is a, which are known from the x's and y's and h which is our parameter vector. In order to solve for h, what do we do? Same thing, we take the derivative of E with respect to h. Repeat that equation, set that equal to zero and that should start to look vaguely annoyingly familiar, right? We have this matrix here, U transpose U, which is just made up of the x minus x bars stacked up, y minus y bars stacked up, transpose whatever, times my parameter equals zero. Remember, what equations equal to zero are? Those were homogenous equations. Uh-huh. All right. So the solution to U transpose h equals zero. Well, the dumb solution is set h equal to zero. We said that wasn't going to work. Remember, when we did the essential matrix and when we did calibration and all that stuff. We said no, since we have everything okay up to the scale factor what we're going to do is we're going to set the magnitude of h equal to 1. Now in this particular case, we were going to do that eventually anyway, because you remember a and b, which

was the h is here, where the components of the unit direction of that line. So a and b have to be a unit vector, so we might as well just set the magnitude of h equal to 1. And so remember, how you can solve that? You can do that whole SVD eigenvector thing again, so given you transpose U, you can find the eigenvector with the smallest eigenvalue and that's your solution to for h. I'm showing you this as a way of talking about that the right thing to do is perpendicularly squares. Here's how you do perpendicularly squares and now the question that you really should be asking is, why is perpendicularly squares the right thing to do? Well, it's because we have in mind a particular model of noise.

9 - Least Squares as Likelihood Maximization

The model of noise we have, and we're going to refer to it as a generative model because I have a model of how the noise is actually put within the system. Essentially we have a particular model that we believe in. Computer vision, you have to have faith. The model that we believe is, there's actually some real line underneath. And that each point has been perturbed off that line. Here it's written as epsilon. By some Gaussian noise that has moved you off that line. And the probability of a Gaussian goes down by a function of what? X squared. So if you wanted to find the most likely solution, you would want to find the points that make the data that you're observing most likely. So those would be the ones where the sum of the square distances are the least because, and things, just thinking about the Gaussian, won't go into too much detail. If you assume all those points were independently done, so the probability of getting all those points is just the product of those probabilities. The product of those probabilities is a product of all those Gaussians. Those Gaussians go e to the minus x squared. When you multiply all those e to the minus x squared, right? You just sum up all of those x squareds and so whatever makes that sum of squared difference the smallest, right, will be the highest probability. And so, there's a whole bunch of assumptions going on underneath here in terms of the probability distributions and being independent and all that kind of stuff. But basically this idea of minimizing the perpendicular least squares comes from this idea that closer points are more likely, right? It's more likely a point will land closer to a line then further away from that line, if the point actually came from the line. Got that? That's going to be important to remember for, for our next lesson. All right. So that's written here in this equation. Basically we're saying the point x y that we actually measure is just some real point on the line u v, okay? That has been perturbed in the direction perpendicular to that line, a b, that's the line that we were solving for. And that perturbation has been some amount of noise that came from a Gaussian with a zero mean and some standard deviation sigma or variance sigma squared. So, for the line fitting or in this case, a model fit in general, this is our underlying generative model. That it is a Gaussian perturbation.

10 - Non Robustness to Non Gaussian Noise

It turns out that that works fine when you actually have Gaussian noise. Turns out it doesn't work so great when you don't. When you have some various, shall we say, non-Gaussian noise. Okay? So, here in this picture I've got a bunch of red points. And fitting the line through those points, okay, not very exciting. Actually, I probably should have had those points spread out. That's the problem when you steal some other people's slides. You say, you know I should have spread out those lines a little bit, but even with all of those points sitting right on the line, okay, I'm going to add one more point. All right. Just one. And it's not going to be a nice point, it's going to be a. Not a, if it were Gaussian point, it would be very hard to get far away. Let's suppose my points actually are made up of a bunch of points that are perturbed by the Gaussian off the line, plus evil Megan came in and sprinkled a few just nasty points. What would happen? Well, here's just one nasty point, bang. This one point, out here, makes this line the, quote, best solution for all of those points. Now, many of you've seen this before for robust fitting. Bear with me. That point is one of our outliers. It doesn't come from the model that we have. And our model is actually made up of two parts. Our model is that there's a line and that there's a little bit of Gaussian noise added, not, there's Gaussian noise added to the points off the line. This model is the evil Megan model. And we don't

even know about that. But we would like to sort of be immune to the evil Megan model. So the question is, how can we do that? And you know, why is the evil Megan model so annoying to us? Well the real problem is that squared error, right? So the bigger the error gets, you square its effect. That heavily penalizes outliers. So outliers can have much more of an effect on our model fitting than we really want them to have. So the question is how to deal with that.

11 - Robust Estimators

So, one way of dealing with it, that we're going to just talk about today. We're going to do it differently next time, is what are called Robust estimators. And Robust estimators say, instead of minimizing the sum of squares, okay? Instead, we're going to try to minimize some, some function row. And row, is going to be a function of two things. R here is for the residual. It's the delta. It is how far off the point is from your models. So for fitting lines it's how far away the point is from the line. So that's our error that we were using before. And before we were just squaring that. We're going to do some other function and it's also going to have a. Scale parameter sigma, okay? Or robust measure parameter sigma. There are lots and lots and lots of Robust estimators. We're not going to talk a lot about them. I'm just going to show you one example because it works very simply. So here's one. This is rho as a function of u. U is just be, going to be that residual, that, that, that r. And here's my sigma parameter, and the function that I'm going to minimize, is just going to be u squared over sigma squared plus u squared. So. Let's think a little bit about how that function works. What it looks like. So this is just this function y equals x squared. Okay? So for y equals x squared, you know, you can see that it grows really quickly as x grows. And that was the problem that we were trying to fix. Let's think about what happens for small values of u versus large values of u. Okay? So if we pick some sigma squared. When u is small compared to sigma, sigma squared plus u squared is about sigma squared. So that means the entire function is approximately u squared over sigma squared, so that just grows by, u squared, the way our original function did, and so it looks kind of parabolic to start with. But, what happens when u becomes really large. Well, when u becomes really large, the Robust estimator here becomes almost u squared over u squared, or one. And that's what this u is here, out here. Okay, so the idea is as your error grows, it starts going up. As a quadratic, just like our regular mean squared error, our, or just like our regular sum of squared errors. It goes up quadratically, but then it starts to level out, and that's what you're seeing here. It comes up, and it levels out. Now, how, where it levels out, that's a function of what the sigma is, okay? So for a very small sigma, it levels out quickly. For large sigmas, it levels out much more slowly. So here's an example, remember, we're going to. Here's the same set of points. Aha. We have now foiled the even, the evil Megan point. Right. So we have this point here. And you'll notice that this line, goes right through those other points. It's as if it ignored this point. Now it didn't totally ignore the point, it gave it a constant weight, but it didn't matter whether that point was here, or here, or here, or really far away, because once that u, once that residual got too big, it got a constant weight. Points that are nearby, they have the, the, the square, error, weight that we had before. So this is what it looks like when you get the scale just right. This is like Goldilocks picking the first bowl, the first time. Never happens, okay? At least, not when I tell my kids, because then it would shorten the story and they wouldn't be asleep yet, so what good is that? If I set the scale too small, I get this weird line, right? Well, basically what's happened is every point is giving it a constant error and it's just kind of going, somewhere near all of the points. And also, if I set my scale too large, remember if sigma squared is too big then it's just like regularly squares. So I get the the wrong solution again. So, these Robust estimators are, powerful, but you do have to have an explicit, not only do you have to have an explicit choice of scale, it's pretty sensitive to your choice of scale. The method we're going to talk about next time also has some scale parameters for, for noise that we have to look at, but it's going to be much less sensitive to that.

12 - End

All right, we're going to stop here. We've taken a brief look at model fitting, as really being a robust estimation problem. So for transforms, you want to estimate the transform and you want to be able

to do that robustly, and we use robust error functions to help. But what we're going to look at next time, is a more algorithmic, approach to doing this. Famous method actually invented by a very good friend of mine, he's older than I am, so I'm not quite that old, that we'll talk about. And what's cool is, it's a little bit more computer science oriented, and this is a computer science class so that's great. And it's nice to, you, we can use a little bit of combinatorial analysis to show exactly how good it is. And the other thing is, it is used all the time, so you can say well finally is going to teach me something useful. Come back next time.

4C-L2 RANSAC

1 - Intro

All right, welcome back to computer vision. Last time, we were talking about, still estimating these transforms, and we're talking about how it was a model fitting problem. And, one way of doing it, robustly, because we'd have some bad matches, was to use these robust error functions. And I promised you, and I always deliver on my promises, though the next time we would look at a more algorithmic approach and sort of a more CS oriented approach. So, to remind you our feature-based alignment algorithm is first extract some features, compute some putative matches, some possible ones based on the descriptors. And remember last time, we, we looked at this ratio of the best match to the second best match and we said, oh, you know, that's a good way of getting rid of some of our bad matches. And then, the algorithm was supposed to do the following, it was supposed to loop until we were happy, hypothesizing a transform, and I'll say actually, from some matches. And then we were going to verify, the the transform by looking for other matches that were consistent with the transform that we had solved for. And, that's really fundamentally what we're going to do today, is talk about finding consistent matches

2 - Find Consistent Matches

Some best matches are correct. That is, the, you pull the best one, and it really is the right point. But some of them are not. The most important thing, though, back like we did in the Hough transform, the ones that are not are just sort of random. They're voting for Mickey Mouse, Santa Claus, Donald Duck. They're not all voting for some other particular wrong transformation. Not that there's anything wrong with Donald Duck, but he would make a poor governor. Although, given some of the other governors, whatever. Anyway, so what we need to do is, we want to find the right ones so that we can compute the pose, the transform, the fundamental matrix, whatever it is, the model. Okay? So this is the model that we want to compute. And today we're going to talk about this algorithm called Random Sample Consensus or RANSAC. So to talk about how this works. Go back to our, again our simplest example of fitting a line. So here we have some points. And if I were to ask you to fit a line, you'd probably say, well. That's the correct line, that line that kind of goes through most of the points and somehow ignores that bad point. You're going to say, aha! That's right. Remember evil Megan? That was the evil Megan point. The problem is if we don't know that, that's an evil Megan point and we actually include it in, say, our least squares fit, we were showing you that our best fit line was not such a good one. And really what's happening, before we talked about robust estimators. What's really happened is, you're just not going to get points like that from a Gaussian distribution. That is such a low probability. That point comes from outside our model. And so we have to figure out, somehow, that's it's one of these outliers that's outside of our model.

3 - RANSAC Main Idea

So here's the main idea of this algorithm called RANSAC. Fitting a line, a model, is easy if we know which points belong to that model and which do not. At which point I wrote, duh. Yeah. A little more interesting. Now if I had proposed a particular model, a particular line, we could

probably guess, in a whole bunch of points, which of those points come from that line, or come from that model, and those are called inliers. That is, they come from the model. The algorithm Random Sample Consensus works as follows, and we're going to go through this. Randomly picks some points to define your mo-, Line or your model. Okay? Then, repeat this enough times till you find a good one. And what does it mean by a good one? A good one is one that has many inliers. That is, a lot of points are from that line. And you might think that this is a. Hopeless idea. Just randomly picking things and, and computing them and hoping that you get. Well it turns out that actually as was shown by Fischer and Bolles way back in the dark ages of the universe 1981. That actually this copes with a very large proportion of outliers very well. I'll also tell you that Marty Fischer was my first boss after my PhD, and Bob Bolles sort of taught me the ways of the force, or of computer vision after 1981. But not that many years after.

4 - RANSAC for General Model

Any given model type, so whether it's a line or transform, they have what's known as a minimal set. The minimal set is the smallest number of samples from which the model can be computed. So for a line, how many points is that? Well, two, all right? But what about transforms, when we're transforming one image to the next? Image transforms are just models. So what is the minimal set of our different transforms? Well, first of all, what are we sampling from? Well, we're sampling from proposed matches. Corresponding pairs of points, okay? So suppose I think that one image is just translated of another image. So the transformation is a translation. How many pairs do I need? One. Because if you tell me where this point a is in this other image, I now know the translational shift between any two points. Another way of thinking about that is I need a delta x and a delta y, and you give me two, an x and a y match. So I know what delta x is, I do this complicated thing called subtract, and I do it on y and that's my offset. Okay, homography, ooh we talked about this. How many pairs of corresponding points do we need to compute a homography? Do you remember, paying attention out there? Four, Megan got it right. How about a fundamental matrix? We showed you something called the eight point algorithm that said if you had eight points, corresponding pairs, you can compute that. By the way, it's actually seven. We didn't really talk too much about that. Do you remember when with eight points we could solve for the fundamental matrix, but then you would sometimes end up with a rank three fundamental matrix so you had to reduce it down to a rank two? We did that whole getting rid of the smallest icon value. That's because there's actually fewer degrees of freedom. So in principle you can do it with seven points, but we're just going to think of it as eight points. So the idea is, for translation I need one pair, for a homography I need four pairs, for a fundamental matrix I need seven or eight pairs. This is the minimal set that it takes to compute these things. Then you can write down the general RANSAC algorithm. Randomly select, it says s points or point pairs, s elements of a sample set. So you form a sample by pulling out s things that you need. You instantiate your model. So you either draw your line or compute your homography, whatever it is you do. And then you get a consensus set, C sub i. And these are the points that are sort of within the error bounds. So if you think of homography, homography maps this image to that image, and I can see are all these points map, how many of the proposed matches now land in the same place that they said they came from? That's the number of inliers. And, in fact, you might even terminate this algorithm as soon as that consensus set is big enough. Or you might just repeat it for n times and then you return whichever model gave you the biggest set. So that's the general RANSAC algorithm. And of course, like any other algorithm, there are some parameters to be chosen in order to make this thing work. So the first parameter is this number s, the minimal set. This how big is the minimal set. That number is actually usually set by whatever model you're using, right? So if you're fitting lines that number is two. If you're doing homographies that number is four, four pairs. So that's set by the model. A little more problematic was this distance threshold t, or delta as wrote in the Greek, of saying how far away from the model that you're computing is excitable to be an inlier. So let's look at that one just a little bit.

5 - Distance Threshold

The bottom line is in order for you to pick a threshold if you wanted to do this sort of analytically, what you need is a noise model. And unfortunately, this noise model is not the same model as the thing we were going before. It's not our line, it's our dist, it's our belief. I'll use the word belief, maybe, instead of model of no, because it's model. It's a model of how that the, the noise behaves. So let's go back to our line example, you remember this? So if we were just looking at a line so let's assume that we, you know, we have this real line and that our points are perturbed by a Gaussian by Gaussian noise. This is the thing we looked at last time. So the distance in a Gaussian is sometimes a complicated thing. So you've got multi-dimensional Gaussian, you may know about Chi squared and that the, and the square root of that is the distribution of, of the distances. In one d, it's actually much easier. It's essentially what's called a half normal. Because basically, you know, the probability that you're d, o, a is, it's, you could either be d this way or d that way, but d is only a positive value. So it's just twice the regular Gaussian distribution. That's what this dis distribution here looks like. Because it says, d greater than zero. But all that matters is given a noise model, you could generate a distance model. And given a distance model, you could pick a threshold. So in fact, if you have this 1D Gaussian, let's say and let's suppose it has a distribution of, of parameter of sigma squared, then you could set the threshold t, based upon t squared. Is this number 3.84 sigma squared? Where the heck does 3.84 come from? Well, I just arbitrarily said, lets pick a threshold such that 95% of the inliers would fall within that threshold. So 95%, if you got sigma squared then I said, t squared at 3.84 sigma squared. If you wanted 99%, you'd have to make it bigger threshold. If you're willing to remove some points, you'd make it a smaller one. But the idea is that you have some threshold. And then you're going to accept the point if the distance D is less than that threshold, you'll count that point as an inlier. Okay? Cool. All right. So let's see, we've got our number of points. We have our distance threshold t, it says what we did here before. We've go one more. The number of samples we're going to compute. Now this is not s, the number of points within a sample. Right? A sample is a set of points that it takes to compute a model. This is how many times are we going to draw a sample. How many models are we going to try and check. So what we'd like to do is we'd like to choose N to be big enough. That we can't actually guarantee that we're always going to get inliers, can we? Right? Because if we're randomly picking points, et cetera and unless we were to exhaustively pick all, say, N choose two. If we're just randomly picking things, it's only with probability that we can say that, what's the likelihood that we got a good solution? So what's we're going to do is let's pick a probability p and what we want to know is how big does N have to be, such that the probability is p, let's say, 0.99, 99%. That at least one of our samples will just be made up of points that come from the correct model? Right? So, that's a way of saying if, if this line, I actually get two points that really came from the line. Or another way of saying it is, I've got this, all these putative matches that are compute and I want to compute a homography. I want at least one set of four matches that I test to actually be the correct homography. All right. So the question there is can we figure that out? And in particular, it should be pretty clear. What if all the points were right? That is all the matches were right. Well, then as soon as you pick some set of four, you would have four points that's correct and you would get the right homography. What if none of the point matches were correct? Well, then you could be there all day and you're never going to get four that match. In between those is that there's some percentage of points that are not correct. So if the ones that are correct are called inliers, what do you thing the ones that are not correct are called? Outliers. Outliers. So the question is for a given percentage of bad points, for a given outlier ratio E. So let's, E is 20, is 20%. Means that 20% of your points are bad. For a given ratio, how many samples do we have to check?

6 - Calculate N

So, what we're going to do is we're going to calculate N. And it's actually really easy. You just step right through it. So we start off with s. S is going to be the number of points in the minimal set, okay? P: the probability of success. This is, I wanted to have probability of p, that at least one of my samples will be a correct set of points. Now e, we said before, is the proportion of outliers. So,

what's the proportion of inliers? Well, if the proportion of outliers is 20%, the inliers is 80%, just 1 minus e, okay? All right, now we start with just a little bit of combinatorics, all right? So, let's suppose I pick s of these things. What's the probability that they're all inliers? Well, if the pro, probability of in, being inlier is 1 minus e, the probability that they're, all s are inliers is 1 minus e to the s. I'm assuming that there are a lot of points to choose from. So, that the fact that I pulled one out doesn't diminish things. It, it's a pretty good approximation. Okay, so that's the probability that all of the points in the sample I pulled are inliers. So, what's the probability that at least one of those is an outlier? Well, 1 minus that. Okay, so that's just 1 minus 1 minus e to the s, okay? So, if I pull out n sets, what's the probability that all n are outliers? Well, given that my sets are independent, so it's just this 1 minus, 1 minus e to the s, all raised to the n, okay? So, that's the probability that all n samples have an outlier. Okay. But, of course, what we want is the probability that all of them are outliers. So that means they were all bad. We want that probability to be really small. That is, less than 1 minus the probability of, of the success. So judging here, right. So if p here was 99%, 1 minus p would be 1%. We want the probability that they were, all of our samples had at least one outlier in them. We want that to be less than 1%. So that means that we can now just substitute in. And we get this ugly expression, 1 minus 1 minus e to the s to the N has to all be less than 1 minus p. Drum roll, please. We can solve for N and bingo. There is our equation. N's gotta be greater than the log of 1 minus p divided by the log of 1 minus 1 minus e to the s. And I'm sure you all have deep intuitions for what that looks like. No? Okay, let's look at some representative values.

7 - What does N Look Like

So here it's just a table and I've highlighted a couple of entries. Lets first take a look at, s equal 2 here. So lets suppose we were doing fitting a line or if we're doing a transformation that's a similarity transform where you take two pairs. Notice that if my percentage of outliers is 50%. So half my matches are bad. I only have to choose 17 examples, for me to have a 99% chance, that at least one of them was the correct one. So doing something 17 times in a computer like that, that's pretty fast. If I'm computing homography, s is now equal to 4. Even with 50%, I only have to try 72 samples. So even though I'm totally randomly pulling these things, and I have no idea which points are right and which points are wrong, and half the matches are wrong, with just 72 of them. I have a better than 99% likelihood that one of my sets is the correct one. And in fact, even if I'm doing a fundamental matrix, all right, looking for eight matches, I only have to check less than 1200 of them in order for it to be likely that one of those matches, at least one, will be the cor, a correct set of matches so the fundamental matrix that I would get would be the correct one. One thing to notice about this table is that N does not go up very much as the percentage of outliers goes up, but it does go up pretty steeply as s gets bigger. So as the number of parameters that you need gets larger, all the sudden the number of sample sets you have to check before you, pretty sure you'll have one that's right, goes up as well. Now some of you are not table people, some of you are graph people. So what I have here is a graph where I just did s equals 4. So like a homography. And on the axis on the bottom you should see the percentage of outliers. And you can see that the number of samples required. Which goes all the way up here to well at 0.8 it's above 2000. But it stays low for a very long time. And that's the really cool thing about RANSAC, is that it stays low even with a relatively high percentage of outliers.

8 - How Big Does N Need to Be

And now let me show you the thing that I just blew right past you that's even cooler about RANSAC. All right? So how big does N have to be? Well, we gave you this equation, right? N is greater than 1 minus, log of 1 minus p over log. [NOISE] Fine. Look at that function. What's p? P is the probability of success. What's e? The percentage of outliers. What's S? The number of points in your minimal set. Notice, what's not in there is the number of points in the image, or the number of features. So, you may have 10,000 or 1,000 or 100 or 500 punitive matches, let's say, for a transform. Remember all those red lines that were going back and forth. The number of samples you have to draw has nothing to do with the number of possible matches that you're looking at. It

only has to do with the percentage of wrong ones. And, then the you know, the size, the minimal set and how sure you want to be that you got there. So that's really great because that means that the whole algorithm is pretty scale invariant with respect, or I should say size invariant with respect to the number matches, the number of, of hypothesized points that you have. And that actually is, is part of the reason that it, it, it works so well. All right, so let's illustrate that here. Let's suppose that I got two pictures, and let's just for the sake here assume that there's just a translation. So I'm just looking for one point match to one point. And the system has proposed a whole bunch of matches. So there's one, and there's another one, and there's another, and there's another. And all of these you and I know are actually correct, right Meghan? Right. Okay. But it also propose this one and propose that one and those are not right. But of course only you and I know that and 9 million people watching. Okay, but the computer is not so smart. It doesn't know that. So showing you how RANSAC works. What we do is we select one match. Here's one. That blue one right there. And we just count the number of matches that agree. So one, two, three, four, five. Okay, great. And then we take a look at the ones that don't agree. Two. So five out of seven. That's 5 times 70%. Or whatever. Five out of seven of them agree. Great. Okay, so we gotta do it again. We pick another one. We try that one, count the ones that agree, one, two. Count the ones that don't agree, one, two, three, four, five. That's two out of seven, last time I checked, two out of seven is way lower than five out of seven. So you would basically say, a-ha. Here is the set of matches that are right. Now I've got five points matches for doing a translation. How many points does it take to do a translation? It only took one. So, should I just pick one? No. I have five translations here, all of which are a little noisy. So what do I do? I just take the average. I could even do sort of a robust average, but you probably don't even have to at this point, right? You just have to make sure you have enough points left in order to take the average. So basically, what you're doing is you're going to use ransack in order to select the points that you believe actually come from the Gaussian noise model. Get rid of the outliers. And then you do an average, which is the right thing to do when you have points from a Gaussian noise model.

9 - RANSAC for Estimating Homography

So here's if we were doing a homography, here would be our RANSAC loop. Select four feature pairs at random. Compute the homography H, that's the thing that moves it from one to the other. And then, what this means is, take a look at the square distance between, some point in the image, transformed by H, and it lands someplace, and take a look at its distance from what it claimed was the putative match in the other image. And count all the ones that are less than some threshold, just do that loop for a while, and then you keep the one with the largest set of inliers. You then compute, we don't use the old H, you compute the least squares, the best H for that set of inliers and that'll be a much more robust estimate of the transform.

10 - Adaptive Procedure

One more little piece of technical detail which is easy and then we'll just show you some examples and then we're going to go have a beer because it's Friday. Remember, n is a function of the probability you want of success. S, the size of the minimal set. And e, e is the percentage of outliers. But I don't know e, right? If I knew which ones were wrong, right? If the oracle of Delphi told me which ones were wrong I would of just thrown those away to begin with, and just computed the thing that was right. So if I don't know which ones are wrong I also probably therefore don't know what percentage of them are wrong. I could try to guess. So here's what you can do. You could say here it's just in words and then we'll do the algorithm. You can guess say, okay, I don't know, I'm going to assume that it's 50% are wrong. Maybe even 80 but whatever it is I'm going to assume that some percent is wrong. And I'm going to start computing some models. If I start finding lots of models where the inliers are much higher percentage than I guessed, then I'm going to adapt my belief about the outliers to some smaller value. So that's written here, as a, as an algorithm. basically, I start off saying, I'm going to assume that everything is an outlier and therefore, I'm going to assume that I have to take an infinite number of samples. So I'm going to be

here all day or night, and weeks, and years, whatever. But, not so bad. The algorithm says, I start how many have I done so far? Zero, I say as long as N is bigger than my current set of samp, samples that I've tried I'm going to pick a sample and I'm going to count the number of inliers. And I'm going to guess that the percentage of outliers is 1 minus the percentage of the inliers that I've seen so far. After all, this is just my first one, et cetera. If that value is less than the value that I started with, and since I started out believing that they're all outliers, I'm going to get some values smaller. So if the new e is less than the one I had before, I set that value, and I re-compute N. So, first time around, N was at infinity, and then N gets lower. And I keep doing this and what can happen is, since I only change e if it's smaller, that means N can only get smaller. So, eventually my samples, the number of samples that I've tried, will get bigger than the N that I need. And, when I do that, then I stop. And that's a way of adapting when you don't know a priori, the number of outliers that are there. Something that I, I didn't mention before. It's true if you have a real noise model in terms of how the noise distribute from measurements, you can analytically set some threshold. Dirty little secret, we almost never have a noise model. Sometimes we do, but not too often. Because we have a noise model from somebody else's censor center. So you know what we do? We end up adjusting it to see what works. Now, not always, and certainly never in our papers. But for real, sometimes you just have to make the thing work.

11 - RANSAC for Fundemental Matrix

How about some example, because RANSAC is actually used quite a bit. So you guys have already seen this one, right. So here's RANSAC for recognition. This again is from David Lowe's work. And what you can see here is there are a whole bunch of matches that were correct, but here's one, for example that was not correct. So if we were just doing, say, an affine transform, how many matches do you need to do an affine transform? Let's see. One for translation, two for similarity, four for homography, so affine is three. I think they're doing affine here. I don't think they're doing full perspective. So basically, you keep randomly picking three points. Find the affined deformation. And when you happen to pick this one, the correct one, all of these points, you can't even see that I'm drawing all of these things but I am, all of those points become inliers. If you happen to pick this point match, this point match, and this point match, you would have said that this box kind of landed here maybe. And then you would compute the inliers and it would be hardly any. And that's why RANSAC works for the recognition. We talked about the fundamental matrix. Okay. So here, I, these are actually images that I showed you, again I think comes from of Harris corners detected on two different images. All right. And then we do the putative matches and instead of drawing across, which can be really ugly, we did something else that's ugly. And again I want to thank Frank Dellaert who stole this slide from somebody else, and I stole it from him. So, whoever's annoyed out there, go yell at Frank. The punitive matches are drawn on this picture, instead of trying to draw them across I'm just sort of showing, this is the belief about how it thinks the points moved. And remember this is fundamental matrix. I've got one camera, and I've got another camera, and it's a question of how the points will move in the image. Well, they should all move smoothly, according to the epipolar constraint. Remember the epipolar constraint? How I've got two views, I could, if you knew the calibration, I could tell you the lines along which they have to move. So these would all be moving along these epipolar lines. But what you can see here is that a lot of these things don't seem lined up. So here we have 188 putative matches done, in this particular case, through cross-correlation. When you run RANSAC, well, now you find a fundamental matrix that has selected out these points. And you see how they all are aligned up nicely, and there were 99 inliers. And here are these points. And you see how they are all lined up not so nicely, like not at all, because they're just outliers. Their matches were just noise. So, this is a case where 50%, more or less, of the points were matches were wrong. And yet RANSAC can handle that. Here's an example from robotics. One of the things these days people do a lot of is use point clouds. Point clouds are recovered from some form of 3D sensing. Whether it's stereo or these days a lot of people use kinect-like sensors. And what you have here is an image from a set of depth points and you'll notice that the bunch of these objects have been colored as different things. But notice the table is not colored. The table has been removed from the objects. You might have

thought the objects were removed from the table. Not really. The table was removed from the objects. How was it done? What's the shape of a table? It's a plane. So if you know your model is a plane you say I'm going to pick some points to find a plane, see how many points lie on that plane. If a lot of points lie on the plane maybe that's a real plane. That's how they found the table, they used RANSAC. Here's another example, this is actually work done by a student of mine. He was doing some work for a robot seeing objects on a table and pushing them around in order to determine how many objects were really there. But the first thing he had to do was separate the objects from the table. How did he do that? RANSAC. Find the table. Here's a video stolen off of YouTube, of somebody doing real time object segmentation off of a plane. And, so you can see it runs easily, effectively, and again they're using RANSAC to define the plane

12 - RANSAC Conclusions

So, a couple conclusions about RANSAC. The good, it is simple and general, it is used often. It's applicable to many different problems. It is robust, even with respect to a very large number of outliers, that's really the key element. And, it is not sensitive to the number of points that are in the, in, in the system. We had you do the Hough transform, and those of you doing the OMS, you're going to do the RANSAC as well. I will tell you that in general it's easier to get RANSAC to work than Hough transforms. The parameters are little more finicky than in a Hough. And for large numbers of parameters RANSAC just means to take lots more samples. And it goes up kind of painfully, but not awfully. If you remember Hough transform the space complexity goes up cominetorically in number of parameters. And these days it's easier to deal with things that require you to be fast, but not big. And so, RANSAC you have to be fast as the numbers go up. In, in Hough the, the memory size gets larger. So, in general, making RANSAC work is easier. The not-so-good, I don't want to say the bad, because Barney and Bob are good friends of mine. But actually, there's a huge body of work on RANSAC and all its derivatives from that. It's, it's just one of the most fundamental algorithms in computer vision. The computational time grows quickly, with the number of parameters. That was the s, the size of the minimal set. Not, not awfully, awfully but you know, when you start to get more than ten you're going to be wishing you were less than ten. It's a little bit challenging if you've got like multiple plains, and you wanted to full, find out each of them. You have to be a lot, then you have to be a lot more careful with your thresholds. So it's not as good at multiple fits. It's really not so great for approximate models, right? So it, suppose it's kind of a plane, but not really a plane. That's a problem, because now you pulled out some points. And where it's not quite, it, it, you know, fit a plane, it's not quite the right plane anymore. So it can be kind of sensitive to when your model doesn't really match, the underlying structure of what's out there. I will tell you that it's used all the time for doing things like, it says homogra, any sorts of image transforms. It's also used often for doing understanding relative views such as fundamental matrices. And generally it's used for pretty much every problem in robot vision. It's amazing to me. affect, I see bubbles, on some occasion, and you know, I keep reminding him that you know, I wish I had an algorithm that I see used just, so often these days, because it, it is a fundamental algorithm for robot vision.

13 - End

All right. That ends, the lesson on RANSAC, the subunit on fitting models based on features, and in fact, the entire unit on features. This kind of work, finding features, figuring out how they match, is, pervasive in sort of geometric computer vision. The computer vision that talks all about how cameras align, and scenes, and depths, and, and, and points. And, so they, they, they use a lot, these days, because of the, just how ubiquitous, imagery and, and video is. and, because these, these methods are sort of powerful and real, and not just myth, they are used in lots of real applications. So, I expect that if you end up doing any work, that involves dealing with multiple images and trying to figure out something about the relationship between them, that you will actually see things like, PARIS detectors, things like the SIFT descriptors and certainly things like the RANSAC alignment.

5A-L1 Photometry

1 - Intro

All right. Welcome back to computer vision today and the next couple of lessons. We're going to just a little bit of time on the whole light reflection process that results in photos essentially hitting our camera or our retina. The idea this is how the physics of the imaging or the creation of the light that the imaging is going to make a picture of. And there's actually a huge amount known about, and, about this area, and it comes from both, sort of, the material science, where we study the materials, but the big, motivator has been computer graphics, right? Computer graphics, the idea is I have some model of of an object and its surface type and I have some model of the light. And I'd like to produce an image as if you were actually looking at the real object. So to do that, I have to understand how light interacts with these objects. Now in the old days and when I say, the old days. I mean, like 20 to 40 years ago. There was this view that computer vision was just inverse computer graphics, right? I mean, computer graphics, you have model of the object and a model of light and you make a picture. And computer vision was thought of as well, given a picture, can you tell me about the light and the object that's there. And so there was a lot of focus on recovering image prop surface properties and object properties and light and conditions as it's sort of an inverse problem. And to do that, we had to have a lot of understanding of the physics of how images were formed. Now more recently in computer vision a lot of what we've talked about here, there's been more of a focus on really the geometry of sort of where features get located and that relates to sort of the, the shapes of objects and the relation and you didn't have to worry quite so much about the precise physics. And then also a lot was saying, let's lump all together, the lighting, the shape. The texture, the imaging, the camera, et cetera. We just get out pictures and we use machine learning or recognition techniques for to trying to do labeling. And we talked and when we will actually be talking about that in the future. But it is it's still important to understand something about how the physics of light works with respect to making images.

2 - Photometry

So here's a sort of toy or a schematic, right? So the, the general idea is that we've got some lighting. All right? And the lighting hits objects. And so we need physical models or models of the physics. And the idea is that that causes light to impinge upon whatever your photo sensitive system is. Whether it's a, a retina or a CCD, or CMOS, or whatever your imaging technology is, and then, of course, that's the picture, that we're going to take a look at in the computer. So there are all sorts of physical processes that impact the image that you're going to see. So, you know, here's a picture, I think, actually, this might be from [INAUDIBLE] don't remember. And you know, here is a picture, two pictures and it is pretty clear to you what is going on here, right? We've got sort of this textured stuff everywhere and there is differential light falling on it. There are shadows. And so when you look at that, you probably don't think that that's a texture painted on the, the surface of the road, but rather that that's a shadow on a more uniform texture. But of course, it could of been painted. So it's your brain has to be doing something in order to understand that. And this white tape that's shown here, is just a little bit showing you sort of how things appear to you, right. To you it's clear that this is a thing that's on the ground, right, that, that is not part of the texture of the ground, but is, is something else. So you said okay, that's one thing that is reflecting one way, and I've got some other object, namely the, the, the asphalt, that's reflecting another way. So that's sort of lighting surfaces and shadows. Of course, there are reflections, right, so in both of these window pictures, you know, you're probably guessing that's there's not like, this thing here that's behind the window, and maybe it's just a, a reflection of what's behind you. Likewise, you can see in this tree, it's all distorted a little bit, because maybe this glass isn't so perfect, but the idea is that you can see this reflection, and that reflection is impacting the image that gets made. Going along with reflection is refraction. Right, so it's most easily seen in things like water, or in various kinds of deformed glass. So here, you know, these wavy lines on the bottom of the pool, well, there are no wavy lines

on the bottom of the pool, at least I hope not but instead the water itself is no longer flat there. There are some waves and that's causing the light rays to bend in a particular way. So, again, that impacts the image that you see. This beautiful picture here, has got both reflection and refraction going on inside of it. An interesting property which is really, it's just about reflection is what's referred to as interreflections and both of these pictures are drawn from computer graphics. It's not so much about the nature of the surface but what impacts the light that you see at a point, and sometimes we tend to think of well, there's just some light source and it's illuminating our objects. Well, there's been a lot of work done sort of since the 90's on what's referred to as interreflection, both in computer vision and computer graphics. So here where we have the shiny materials, you can see everything being reflected within it. But I think a more subtle example is oh this is the famous graphics teapot being rendered in a variety of ways, now of course, in a shiny surface you know that you're going to see a reflection of this purple teapot in front of it, but also in these partially reflective things. You see this purple color in there? That's reflective of this purple pot that's in front of it, so the idea is, you have all this light bouncing around. And all of that light can impact what's actually observed at a particular point. In computer graphics there's a method called ray tracing, which basically, as rays of light come in and they bounce off, they create illumination at all these, different facets of the model, and you can get a really rich, rendering. You know, one of the things that's a little different in computer vision today than before is. Traditionally when I talk computer vision I could assume that everybody had taken graphics. because there was sort of this connection between graphics and vision. These days it's much more likely people have taken machine learning because it's as if you have computer vision is a data driven inference problem. And people are less likely to have taken a graphics course and understand the physics that are here. Computer vision draws both from understanding what's going on from data, but also having built in models that have to do with the, the nature of the physics. Then there is things that happen to the light between bouncing off a surface and entering your eye. So here's an example referred to as scattering. Where this is what the picture maybe should look like if you think of it as being very clear. But if you actually just took a look at the raw image you would see this because of moisture and other particulates in the air, and the, the wavelengths that are absorbed. That the light changes as it comes through. And in fact there was a really cool paper done about depth, from what's called The Dark Channel. Which basically says that you know, the further away something is, the harder it is for something to be black in your image, because there's a little bit of fog all the way through. So they actually estimate the depth of the seam based upon, essentially, the amount of fog induced lightness that's there. And this is all from an understanding of the nature of the physics. And, there are lots of other phenomena that impact, you know, the light that comes in, you know, here's fog, and water dripping, and, and all sorts of, different types of effects

3 - Surface Appearance

So what we're going to to talk about today, is essentially the surface appearance of objects. And, what we mean by that is, you know the question is, if I've got some surface element, and it's being illuminated by a source, what is it the sensor's going to see. And the idea is that the image intensity is going to be a function of a couple of things, one is the local shape, and we can think of the shape as just being the surface normal of a little patch. Then there is the, surface reflectance, that is, what, what are the properties of the surface that affect the light. And then there is the illumination, that is both the direction and the type of light that's coming in, sort of its intensity, maybe its wavelengths, and those kinds of things. And, the amount of reflection that'll be observed is a function of both, sort of the incoming direction, and the outgoing direction and its relationship to the surface, and we're going to make that a little bit more formal going forward. So now we're going to do just a tiny bit of, nomenclature with respect to the physics, it's a little bit confusing and later we'll drop some of it, but it's, but it's important to get just a little bit of a basis.

4 - Radiometry Radiance

So, the general area of considering light is sometimes referred to as radiometry, which is the idea of measurement of light. And the first term that we talk about is the radiance. Okay? So the radiance is always some light that is being carried by a ray, or sort of, you can think of coming off of something, or out of something. In a minute we're going to talk about the irradiance, which is the amount of light falling on something. All right? So the radiance is sort of energy. As it says, it's energy carried by a ray. And, and what I want you do think about is, assume this like this little patch out here, that's of the sort of fixed angle, that's observed down here at some point, and this angle, this little patch is we'll call d omega. All right. So, radiance is measured is as power the amount of energy per unit area. But the area is measured perpendicular to the source. So this is our little area here, okay, but our light source is over here, I'll make a little sun there, okay, and this is a little patch of the sun. What we actually have to think about is the area perpendicular to that ray. And so because it's perpendicular, we measure, this is going to be a little weird, we measure the amount of radiance in this funny unit called watts per square meter, so that part's okay, right? It's watts per, you can imagine watts per square meter of that little disc, but it's per what's called steradian. What's a ste, well you know what a radian is, right? A radian is a measure of angle. A steradian is a measure of solid angle, right? So this is, this little cone, this is a cone, funneled out, right? And the steradian is the amount of angular area of that cone, okay? because if I had some big sort of extended source here, you'd have to integrate over all of these, these different area. But we're only going to use this little delta omega patch. And like I said, in a minute we'll, we'll simplify this whole thing. But just this idea is that, the energy that's coming from that thing is a function of. So I have to think about how much light is falling on my area as a function of the amount of different area that I can see up there, right? So one is the area down here, and the other is the amount of the source that I can see, and that's measured in steradians because that's solid angle.

5 - Radiometry Irradiance

The irradiance, sometimes called energy E, that's the energy actually arriving at the surface. Okay, so now we're going to talk about the energy at the surface. Okay, so now I've gone back to and I'm going to leave both this perpendicular part. And this little, this is the area on my surface, okay. Now, it's because I'm actually looking at my surface I'm just going to measure that as the power per unit area so that's just watts per meters squared, okay. So clearly, there's this relationship between the amount of energy hitting that perpendicular part, and then this skewed, the, the, the area on the surface. And, the, the easiest way to think about that is to think about this as a foreshortening effect, all right. So, here we have a, a light source, and the idea is that this is the direction of light, and this area, this card here is meant to be perpendicular, right. So there'll be a certain amount of light falling on that, on that card, right. But if the card was not perpendicular right, if the card was tilted away, then the same amount of light now gets spread over a larger area. And so it's foreshortened. So in other words, we get less light per little unit area from that light source. And we can write that down in the following way. Sort of for the surface receiving some radiance and from some direction, coming from this we'll say, fixed little patch, the omega, the corresponding irradiance, the light landing on the surface, is just the radiance modulated by this angle. Right? So is that angle gets closer and closer and closer to, to power of 290 degrees, the amount of ra, irradiance goes down to zero, which is right because you're essentially spreading the same amount of light over a much bigger area.

6 - BRDF

So that sort of talks about the energy and now what we have to do is we have to talk about what is the relationship between the light coming in and the light going out. Because the light going out is the light that's going to go into your sensor, right? And to model that, we use what's called the Bidirectional Reflectance Distribution Function or BRDF. Okay? And the idea of the BRDF is that I've got a little surface patch that has a normal. And my light comes in at some angle and I'm going

to be viewing it at some other angle, all right? And I want to talk about the amount of, sort of the proportion of this light that comes in as, as sort of what proportion of this light coming in, in this direction, goes out in some other particular direction. By the way, this angle theta here, that's the tilt angle. That is the amount that you're off of the normal. And then there's a rotation around the normal that's the Okay? And for some surfaces, we actually have to worry about both of those angles. All right. And so the irradiance is the amount, just like we said before, it's the light per unit area that's coming onto the surface. And now it's radiance, because now it's not the radiance from the light source, it's the radiance back from the reflecting surface. Okay. Remember, radiance is the stuff coming out. Irradiance is the stuff landing on it. So now we have the, before, the radiance was the stuff coming out of the light source. Now the radiance is the light coming up off the reflective surface. So we'll just write this as E of the irradiance of the surface in some particular direction. The L is, I don't know. I don't know why we, I guess we use L for radiance sometimes, because it's the lightness of what you're going to be seeing. That's the radiance from the surface in some particular direction going out. And the BRDF, the Bidirectional Reflectance Distribution Function is just the ratio of those two things. Right? So it's the percentage of the light reflected as a function of the light coming in, okay? A couple things about it is that obviously, you can't reflect more light than came in, so that ratio is going to be less than one. Now you might get a very bright spot in some direction, if it were to integrate over all these different lights. But from one particular direction, the most I can do is send back all the light, I can't, I can't manufacture light. This is a, a passive system here.

7 - Important Properties of BRDFs

There are a couple of cool little properties of BRDF. One named Helmholtz, after the guy Helmholtz, basically light doesn't know which way's up, or another way of saying it is it's a reciprocity. That is, if a light source is coming this way, and the camera is going that way, and I swap the camera and the light source,. I get the same, reflectance function, right? So, it doesn't matter whether the light comes in this way and goes out that way, or comes in this way and goes out that way. There is no light diode, for the nerds among us, all right? There probably are, you probably could build active light diodes, but for normal properties you don't get that for our purposes, there's another important property which is referred to as rotational symmetry. And the way of thinking about that is. If I rotate the surface, about the normal, nothing's going to change. All right. And what that, essentially means is is that I only have to worry about the little tangent plane right at the point that I'm looking. And so the only thing that matters, is the relative direction between these. Not just in, in terms of here I want to talk about the amount around the angle. So the absolute, angle doesn't matter just the, relative distance between them. And that's written like this. The absolute angle in terms of the tilt still matters, okay? And we'll, we'll show you that in just a minute. But the, the rotation around, is only relative. Around the normal. Bidirectional reflectance distribution functions for real objects, can be incredibly complicated here's a sort of, little panoply of different types of reflectance functions. We're not going to talk any more, about those. If you want to do, take a computer graphics course and learn how to render hair or. See this bird down here by the way. I think it's peacock feather. It's actually not dye, but it's refractive so bird feathers are actually not painted if you will. But they refract the light through the material that's in it. It's a very complicated BRDF. We're not going to talk anymore about that, because we're going to talk about. Two specific reflectance functions, that are special cases of BRDF's, and that you can combine them to sort of make most normal types of objects. Namely we're going to talk about what's called diffuse, or matte reflection, plus specular reflection.

8 - Reflection Models

Let's talk about reflection models. Here we have our surface, okay. And we have our incident light, and it's coming in. And it's coming in at some particular direction, and this'll be some theta i, which is going to be the tilt off of the surface normal. The first physical process I want to talk about is referred to as sort of body reflection. And what happens is, is the light comes in, and it sort of

scatters around at the top part of the surface. And then it comes out in lots of different directions. And therefore, it's impacted by the stuff that's in the surface, all right, and that's referred to as diffuse or body reflection, okay. And, it tends to have what's referred to as a matte appearance, and some sort of very, lets refer to it as non-homogeneous, sort of particulate matter, have that kind of reflection. So, an example here is this clay pot, if you think of this clay pot and you look at it, in some sense there's no shininess to it at all. Right? It's purely matte, it's purely diffuse. And in particular we'll talk later no matter what direction you look at it, it seems to be approximately the same brightness. It turns out it's actually pretty hard to make purely matte surfaces. Like if you take matte spray paint and you spray something even then there's sort of a certain amount of shininess to it. Which we'll get to in a minute. But the, the general model we're that looking at of body reflection is the light comes in, is scattered about in the medium of the surface and then goes out in variety of directions. So that's the body or diffuse reflection model. So what else can happen to the light? Well the other thing that can happen to the light, is referred to as a specular reflection, okay? So a specular reflection just like you think, some of the light before we saw penetrates down et cetera. But, some of the light doesn't, it just bounces straight back up. Okay? And that's referred to as surface reflection. And surface reflection is a specular reflection, and it, it's responsible for sort of the glossy highlights that you see on shiny surfaces. And in metals in particular. So here's a picture of a variety of things. I'm going to, this one, by the way, is, is a fake picture. That's just to show you the highlights. But for example, shiny metal things this metal pot. Here's an interesting one, right, here's a can, that clearly has stuff painted on it, right? So that's affecting the light coming in, it scatters around and bounces out. But you see also that there's this stripe here, right? Well that stripe is the specular component of what's there. By the way, if you want to study more about specular things, the reason it works in metals, it has to do with the way the light interacts with the electrons that are in the surface. And, that's part of what you know, makes metals metals, or these free electrons. Go take a look at your material, science stuff, but those sort of, atomic physical properties have an impact on how light behaves when it hits the the surface. All right? And that's your specular reflection. And the general overall view of the universe, is that the image intensity can be thought of as a combination of the body reflection, plus the surface reflection. Okay. And as a reasonable model for what's going on, we can sort of model most surfaces as having a certain component of body or matt reflection, plus specular reflection.

9 - Diffuse Reflection and Lambertian BRDF Part 1

Let's talk about now a specific mathematical model of body reflection. Remember before we said it comes in, bounces around a little bit, and then scatters out. Okay? So, the model we're going to talk about is called Lambertian model, or a, it's a Lambertian BRDF, but we'll just think of it as being a Lambertian, all right? And we're only going to talk about the body reflection part. And Lambert's law, Lambert being the guy who did this. The sort of folksy way of thinking about it is, that no matter how I look at it, no matter what direction of reflectance that I have, it looks to be the same brightness. That is the only thing that's going to impact this, is the incidence component of the light, not you know, the more perpendicular the surface is to the light source, remember the radiance and air radiance thing, right? It gets more of the light and therefore the brightness will only be effected by the angle between the lig, the surface and the light source. Now, most people, not most people, some people will then show you a picture that looks like this. And by the way, if the person who made this picture is listening, I'm not saying that you didn't understand what happened. I'm just going to say that most people when they see this picture think the following, which is not true. Okay? What they think is that, you know, depending on the light source coming in, you know, the more, the more perpendicular it is, the more light you have reflecting out. That is true, but what's drawn here is that the amount of light coming out is the same in all directions. That is just not true. Okay? By the way, I want to thank Berthold Torn, who by the way invented the shape from shading problems and most other computer vision problems way back in like the 70s and 80s, because I took my first computer vision course and he made a big deal. And you see that Berthold, I was paying attention.

10 - Diffuse Reflection and Lambertian BRDF Part 2

What actually happens in Lambertian shading is the following. Okay, when the light is reflected out, basically it falls off, so there's, it, it's sort of equal area here. So you see this amount? So more light is reflected perpendicularly out than is reflected sideways. Okay? And in fact, as you fall off this thing goes down by the cosine of the angle. So that actually almost no light at all goes out sideways. So a lot of light reflects back up and almost no light at all reflects sideways. So you might ask. Get ready Megan. Well why does it look equally bright from all directions? Well why does it look equally bright from all directions? I love it when stuff works. It's that same foreshortening effect, but this time it goes the other way around, right? If I've got sort of a pixel having a particular angle, so that's what this is meant to be, a pixel having an angle, when I look down through here. It has a smaller area. It's drawn sort of wrong here. Than when I look this way. Think about it this way. If I had a camera out here, the same angle, right, would now get a much bigger area coming into the same pixel. So if I understand the math right, and I think I do, what this means is, is that. The amount of area that I see on the surface grows by the cosine of that area. Or I should say one over the cosine of that angle. So as that angle gets smaller and gets closer and closer to 90 degrees, right, I start to see an infinite amount of patch in the same little pixel, but over here I'm putting out smaller and smaller and smaller light. And these cosines, in fact here they are here, this one and that one. They cancel. And that's why it looks equally bright from every direction. So, I think I got that right. I'm sure my computer vision colleagues are going to send me email, or just make fun of me on the web if I got that wrong but I'm pretty sure that, that's the way it works. What you need to know is that it appears equally bright from all directions.

11 - Diffuse Reflection and Lambertian BRDF Part 3

Here we're back to our diagram where we've got our incident light coming in, light going out in some direction, and the surface appears equally bright from all directions. That is, it is independent of your viewing direction, all right. So one way of thinking about this is that the BRDF of a Lambertian surface is simply a constant, and that constant is referred to as the albedo. And here it's just written as rho, and this is the rho sub-d for diffuse. By the way, often, when this is written you'll see a cosine theta i in the denominator, and that's for compensating for this viewing condition. But I think that just makes life confusing. So, we're just going to think of the, the BRDF as being a constant for a, a Lambertian surface. And so, what that means is that the surface radiance is just going to be that constant times the source intensity, i, times the cosine of the incident angle. Right? because the, the irradiance is a function of the radiance of your source times the cosine of that angle, and then the thing looks equally bright from all directions. So that's usually written this way, as just the magnitude of the source times the albedo. And you can write this as the dot product between the normal and the incident direction, right. It's just the dot product, the cosine, between the normal of the surface and the angle of the light coming in. And that's what's referred to as Lambertian shading. Here's an example of, again, a fake sphere Lambertian shaded, and the idea of course is that, you know, when it's pointing in the direction of the light source it's brightest, and then it falls off by the cosine of that angle. Okay, so that's the matte component, the body component.

12 - Specular Reflection and Mirror BRDF

What about the specular or the mirror component? Now of course in a real mirror, you all learned this in high school physics. Right? There's only a reflection at the perfect mirror angle. Remember, angle in equals angle out? I'm sure you learned that somewhere in your, in your physics class. Well, so using our, our diagramming again, what this says is that all the incident light is reflected only in a single direction. Okay. And I'm going to call that direction m for the mirror direction. And here, we have v for the viewing direction. Okay? And so that light is only going to be visible when v equals m. All right. So I have to be in the exact, so if I've got a point coming from right here, I'm going to see it right there at that, that point. And everywhere else, it's not reflecting. Or I might see it at a

different point, but all the light is being reflected along the particular ray. All right. And so the BRDF is just a, what's called a double-delta function. If I'm, if I'm thinking about this on my surface, right? This angle down and the angle out has to be equal, but this is drawn difficult. It's, it's in 2D. Not only do they have to be sort of at the, the same tilt, but they have to be in a plane. Right? All over to this and this have to be in the same plane, because if I were to rotate it this way, all right? Then it would now be bouncing off in another direction and so, that's why there's two angles. This is the tilt angle and this is the rotation around angle that says, basically one has to be pi opposite the other. So we take this BRDF and we stick it into our radiance equation. And what it says is that the radiance, essentially the brightness that we're going to see out is the magnitude of the source intensity. The row sub s is the specular reflection. So for example, it might be a darkened glass that filters out some light, so it still acts as a mirror. But only reflects some percentage of it, that's what the row would do. And then we have these two delta functions, okay? And these delta functions say that basically the angle in terms of the tilt has to be the same and also the rotation around have to be exactly pie different that's what those two things say. Now remember, the delta function is a function where if the argument is zero, it's a one. Otherwise, it's, it's zero and that's why it's written as a delta function. So I want to abuse the notation just a little bit and I'm going to say, well, I can also write this as saying that the radiance in this particular case, same as before. But now I'm using, I guess what I'm calling a vector delta function. Right? So what we said before is that I only see a reflection if the viewing angle is exactly the mirror angle, right? So essentially these two things would have to be the same, so that I'm lined up in exactly the right spot. Another way of writing that is that the half angle is the same as the normal, right? So another way of saying that is if the vector that's exactly halfway between the sensor and the light is the same as the normal, then you get a mirror reflection. And that would be for a sort of a, you know, a single perfect point. So that's the Mirror BRDF.

13 - Specular Reflection and Glossy BRDF

Most things are not, that are shiny are not perfect mirrors, but we can think of them as being glossy. Okay and what the glossy means is, if we have our illuminate here and we're coming out some angle. It means I don't have light just at the mirror point, but it's spread out a little bit. Okay. And that's what this lobe here, is meant to show. Okay, and the more gloss as opposed to being specular. Right? Blurrier, I make that lobe a little bit fatter. Okay. And that's written as this equation which says that basically now instead of doing that delta function. I just take co sign, of the angle between them. All right, and dot V, raise to some exponent. Okay? Cause that's the angle between them and I raise some exponent. Now, if the angle between them is zero, that means it's a perfect mirror angle. Right, then dot product is one. Raised to K it stays one and I get a perfect reflection. But, as the. The angle falls off a little bit, co sign becomes a little bit less than one, and when I raise a less than one to the K, well as K gets bigger, that falls off faster. Okay. And, in fact, you can see that illustrated here these are some artificially generated images, so here it's just, you know, moving a light source to different directions. Okay, not so exciting. Here, you can see that the specular component is getting tight and tighter and tighter, all right? And so it's getting more like a mirror, that's the raising of that K, okay? And so that's, the specular component.

14 - Phong Reflection Model

And in computer graphics and in computer vision, what we often do is we combine that diffuse and the specular. And it's referred to as the Phong model because Phong was one of the first people to do it. Then there's the Blinn Phong model. And there's lot of models that have increased beyond that. But the idea, basically, is that you say that. You can model the BRDF of lots of surfaces, as a combination of a Lambertian plus a Specular. So we have the Lambertian which is the, you know, everything looks equally bright from all directions. That's what these equal sized arrows are meant to show you. Remember, the light doesn't go out equally. It just looks equally bright. Plus the Specular Model whose sort of blur amount is defined by that fall off. And when you combine those you can get sort of the Matte plus the Specular. And here's an example of some recovered imaging

where here you have a original image, and here you've sort of broken it down into the. Matte components and the sepcular components. Likewise here you have this object that has the, the matte surface. That's the paint itself, plus the glossiness of the varnish. Or whatever it is. When you, what, what is it you put on top of ceramics to make it glossy? Gloss? Gloss, thank you very much. Okay, and the idea is that this is the matte component and again, this is the specular component. And the ability to pull those out, is a computer vision problem.

15 - End

That's all the physics we're going to do. We've talked about how we, model the light, and we talked about surface property, so the albedo of the mat reflectance, and sort of the specular component, and we've talked about how the shape, the surface normals, interacts with that. So in computer vision, there are these fundamental questions of, well, when I look at a, a scene, can I figure out, say the shape, or the albedo or both, just from an image that I'm, that I'm looking at. And so fundamentally, this is an ill-posed problem, right, because it's an interaction between both the shape and the whatever's painted, or the texture on the surface, and yet our brain does it all the time, which means that we, as computer vision scientists, we'd like to try to do it, too. So over the next couple of lessons, we'll be talking, sort of, the computer vision approach to recovering those elements.

5B-L1 Lightness

1 - Intro

All right, welcome back to Computer Vision. Last time, or whenever it is you chose to watch. The last thing I talked to you guys about was how surface appearance involve the interaction between the lighting and the surface itself and the shape. So that's what it says here that the image intensity that we'll see will be a function of the lighting or the illumination, the surface reflectance, and then when we say the shape what we actually mean is sort of the surface normals. That is, how the shape is how the surface bends depending upon its direction with respect to the illumination. And so one of the challenges that this poses is, if you wanted to recover say one of these, like you wanted to recover the shape or you wanted to recover the reflectance of what's there, you know the texture or how bright it is. In some sense you can't do one without knowing the other. And this is a classic example of what's referred to as an ill posed problem in computer vision. That in some sense you cant do it without making some sort of an assumption. And a simple example can be seen here. All right, so what do you see here? What you see is an image stolen from the web. But probably what you're probably seeing, is you see some sort of matte, or, light-textured sphere, with a light shining, probably from say this direction here. You might even see this is just a little ridged in it there, and then the light falls off and then there's a shadow-cast here. Is that what you see? Yes. But actually, what's really there, of course, is this very flat monitor with different pixels that are sort of violating the whole illumination thing. Because they actually emit light themselves, which violates the entire principle of what we were saying before. And, in fact, your stereo, right? My two eyes are looking at this thing and saying, this thing is totally flat. And yet, my brain says okay, yeah, there's this nice rounded thing there.

2 - Lightness Assumptions

In general, what your brain has to do is to make a whole variety of assumptions in order to disambiguate, between different possible solutions. That is different possible interpretations in order to come up with one that it really likes. And in fact your brain does this so well you don't even notice that it's doing it. A college of mine, Ted Adelson is sort of a master at exploring these kinds of thing, and he's put together some really compelling demonstrations. Of your brain making assumptions, in order to determine lightness. So one of them is here, right? So this is a relatively

simple scene. You can see the two squares marked a and b. And if you look at it real carefully you'll say oh, yeah. A is a dark square and B is a light square, right? And this is so mind blowing because. Even though I look at this all the time, I can't believe it, all right. You're ready? So here's this gray bar. Nothing up my sleeve, okay. And you'll notice that gray bar, is at constant intensity all the way along, right. It's the same intensity. It's not like it's lighter and darker. And now we're going to move that gray bar, and it's lined with square A and B. And yet there's no edge here, and there's no edge there. See I'll remove those lines. So A and B are actually the same intensity, okay? And when you look at it, you swear they're different intensities. In fact, here you can, I put two gray bars there, right? And now. Depending upon how far away I if I kind of squint and I ignore all the rest of it, I can see that all of these squares are the same intensity. But it's just, you know, incredibly hard to see that. So, what's going on there? Well clearly what's going on, a couple of things. One is, my brain is compensating for this shadow, right? So it's seeing the shadow cast by the cylinder. So it says, okay, wait a minute. If there's more light here than there is there, then this thing, if it's the same intensity coming off the, the image, it must be a lighter square. The other thing is that the idea of this being a lighter square, is consistent with this interpretation that this is a checkerboard. And these are nice light squares, and the repeated pattern. And generally the way I like to think of it, as your brain likes consistent interpretations. So this is sort of compensating for the shadow. You're brain is saying okay, I see that shadow, I understand that shadow. So the thing underneath it must actually be lighter, even though the, the image intensity coming up at you is the same between A and B. So here's another example, also from TED, that your lightness perception can be influenced by 3 dimensional cues. That is, your belief about the underlying shape. All right so here we have two squares there, and those probably look like they're the same intensity to you. And in fact they are, as the grade bar there shows you. But now we get to blow your mind again. You ready? Here we are. Okay, those two squares, and now you're saying no, not possible. It cannot be that those two squares are the same intensity. Well, here comes my bar. And there it goes. I love this. And son of a gun, it's the same intensity. In fact, let me back that up, right? So this square and this square, are the same brightness. And when you look at it just like that you just don't really want to believe it. And then when you put it next to the grey bar you realize that it's true. So that's, your brain again interpreting that okay, this area is in a more shaded area than that area. So, this square even though it has the same intensity as that, because it has less lighting falling on it, it must be brighter. Okay, so again its your brain making these kind of assumptions. And there's a very simple definition of this called simultaneous contrast, so if you're looking at this it may depend a little on the setting of your monitor. You probably see this square, as darker than this square, even though, of course they're the same intensity. And, the idea is, that your, your brain, sort of focuses on these, these contrasts. So, your brain is doing all these kind of things in order to figure out what lightness is.

3 - Reflectance vs Irradiance Quiz

So we see that the likeness that we perceive at location x, y is a product of R and E, where R is the surface reflectance affected by the color, texture of the object, and E is the incoming light energy or irradiance affected by the brightness of the light source and the angle between the incident ray, and the normal of the surface. Okay, now out of these two, which one do you think varies faster or more frequently with changes in location and which one varies slower or less frequently? Type in your answers in the boxes. You should say more in one and less in the other.

4 - Reflectance vs Irradiance Solution

Okay, take a look at this image as an example. See how the appearance of this wall, which we can assume to have constant color, varies gradually due to lighting. But look at the features in this image, these are created due to reflectance of color variations. So we can expect reflectance to change more frequently with respect to the location compared with irradiant light. There are of course other elements to consider. For instance, object boundaries and shadows. Sharp edges can also affect observed lightness. These factors definitely complicate things, but realizing that they

change more or less frequently or in different ways, forms the basis of identifying and ultimately separating these factors while observing an image.

5 - Ambiguity of Lighting and Reflectance

In some sense you might think that the visual system is making a mistake, right, I mean after all these two squares have the same amount of photons hitting your eye and yet your brain says, no that one is made out of darker stuff than this one. But of course what's actually happening is your brain is sort of decomposing this problem into the amount of illumination and the underlying reflectants. Let's use this example to frame the problem a bit. First of all we're going to assume that we have a Lambertian surface. And what we mean by a lambertian surface you remember is just the, the lightness that will, will come back to the camera. I'm just going to use the term lightness. I'm not going to use radiance or irradiance, etc, because I'm, I don't want to worry too much about that particular physics. Basically I just want to talk about the lightness that the camera sees as, is being a function of just three things. The I is just the power of the illuminate, the strength of the light. Rho is the albedo and then there's this theta angle and the cosine of the theta determines the brightness of it. And it's just the angle between the illuminate and the, the normal, sometimes called the incident angle because it's the incident angle. The problem is, we have this ambiguity, all right? Let's combine both the angle and the strength of the illuminate into a single term E. Just think of that as energy or something like that, right? It's the amount of energy incident on this, we've already taken into account the cosine angle. And then we also have some reflectance function maybe it's just the albedo R. So the brightness at any point of time is equal to this product of the reflectance times the energy, R times E. So L is equal to R times E. So, we can think of it this way, right? E is the amount, is the sort of the appearance of a white piece of paper with the given aluminant, right. So if you were reflecting everything everywhere, E would just be what it would look like if I had a white piece of paper. Likewise, R, the reflectance, if I just had a constant white light, all right, we're not worrying about color, but just constant light. It's sort of the plainer patch, that's, that's what the, the, the reflectance is. And L is what we see. All right. And the problem is, if we measure L and we want to recover R, the question is how is this possible because we might not know E, right. So E can vary, right? It can fall off, you know, because of the shadow, etc. So the question is, how can we do this? And the answer to do this is we have to make some additional assumptions.

6 - The Mondrian World

So some assumptions I'm going to tell you now are, the ones that we're going to use to drive a particular computational theory. It's not clear exactly what assumptions your brain makes, because your brain does some very complicated things. But, here's some simple assumptions that we can make use of computationally. So this first assumption is that, this light is slowly varying, okay? So for sort of a kind of a plainer or, or smooth world. That's not unreasonable. The light falls off. So we've got these beautiful lights here, that Megan has set up to make me look gorgeous. I look gorgeous, right? Right Megan? yes. Thank you very much. anyway, and as I move the light, change it a little bit. Now we have some other light sources, but the idea is that the light changes slowly as I move around. In fact, even on a cast shadow, when you see these shadows here. Right first of all they're not harsh because Megan's such a good videographer, she set up these lights. But also, because of the size of the light source, there's still relatively slowly varying compared to say the line between my skin and the shirt, all right. So the light is slowly varying. The next thing and this is somewhat more problematic, is we're going to assume within an object or within a patch. The reflectance is constant. That's a little bit of a stretch, because there's a question of how big a patch, et cetera. So we have these things here, you see this texture, and maybe you can say we're going to assume that this texture is constant. But the idea is that whatever reflectance function we're going to use, we're going to assume it's constant over a patch. And that between objects, the reflectance changes quickly. Okay, so that any smooth variation we see is due to lighting. Any sharp variation that we see, is due to the texture, all right? And that should give you a hint of how we're going to solve this, right? We're going to look for changes. And we're going to try to

eliminate the smooth changes and keep only the sharp ones. Now this assumption is sometimes called the Mondrian world. And if, if, I, I, I don't really know whether he's French, or must be Piet. Must be so, that would be Mondrian. My French is awful, so somebody, write in a complaint to Megan. Anyway Piet Mondrian, a very famous artist started a bunch in contemporary art. And you've probably seen his work. He made these pictures that were just these constant intensity patches. Now, his patches happened to be square, or rectangular, I should say, and that was part of the deal. But the idea is that, this is the model that we're going to make use of, right? That our world is made up of patches of constant intensity separated by sharp boundaries. Now, of course, we're not going to use anything so beautiful, he's an artist, but we're boring, so we're going to assume that we've got objects that look like this. Okay, so we're just going to talk about intensity today. I don't have these black stripes between them, I just have patches that have constant intensity.

7 - Lighting in the Mondrian World

So, here's what we're going to do, right? We said that L is equal to the reflectance times E, which is the, the total illumination, all right? So we're going to assume that the illuminance, E, is low frequency, very slowly. So, you know, here would be an example, it's brighter here, it's darker there, and it slowly changes. So, that's the illumination. Conversely, we're going to assume that the reflectance is constant over patches that are separated by these sharp edges. This is just what we said before, right? So, here we have these sharp edges. And that's our reflectance model. And when we light the Mondrian world, we make a picture that looks like this, all right? And it's just that L, which is the thing we actually see is R times E. So R is that constant part plus high frequency and the E has the smooth elements. So, if we want to recover the reflection, we want to recover the albedo, the R function, all right? How might we do that? Well, let's look at the intensity profiles of these two pictures. So the picture on the left is just R, this is that reflectance function. And here is L, right? The reflectance function times the slowly varying energy. Now, if we took a slice through these, right? And we compared what the plot would like, it would look something like the following. So on the left, you see we have these sharp changes, right? Which are these changes here, right there, there, there, those are the constant regions. And on the right, we have the same thing, but it's been modulated multiplied by this decreasing illumination, right? So you see that, you know, in some sense this structure is reflected in here but it's got this other slowly varying part of it. So the question is, frankly, given this, how do I recover that?

8 - Retinex

So a variety of proposals have been made. Most famous is by, Edwin Land, actually, and it's called now the Retinex Theory. He had a variety of, ways of going about this but overall the whole process is called the Retinex Theory. This was Edwin Land, a remarkable inventor. He was the inventor of the Polaroid Land camera. So he originally was doing, polarized filters, and Polaroid the company got started and he was part of that. And then when, they invented, he invented the instant film it wasn't enough to call it the Polaroid Camera, it was actually called the Polaroid Land Camera. He was incredibly industrious, and har, driven, and driving kind of guy. He used to give really interesting, remarkable demonstrations of humans perceiving different lightness, using. In fact, he had this really cool way, he could show an image of just one color. Kind of, this pink, with these different squares. And then a, a greyscale image, of, also, different squares, and overlay them. And humans would see color. And it was, it was remarkable. I had the opportunity to see him, when I was still a graduate student in Cambridge. And he's kind of giving this talk, and then also shows this thing, and these colors appear. And the whole audience broke out in applause, it was kind of cool. Anyway, Land, invented, or, or, or put forward a, a variety of what's re, called the retinex theory, and I'll give you just the general idea of how that works. So the goal, is to remove the slowly vary, it's the slow variations in the image, and there are a variety of ways of doing it. One approach, that I'll show you in just a minute. Is to, if L is equal to the product of R times E, then if I take the log of both sides. So, obviously, the log of multiplication is just the sum of the logs, right? And then what you're going to do is we're going to, and I'll show you this in a minute,

we're going to hi-pass filter this, only keep the high frequency. And then we're going to threshold to remove. Any of those low frequencies. And then we're going to invert it to get back the, actual reflectance. So let me show you what that looks like

9 - 1D Lightness Retinex

And this a picture taken out of from David Forsyth's text. Let's suppose, that we have a sharp change in albedo, and albedo's going to be row. So the log of that, is also going to have sharp changes, okay? By the way, why is it negative? The reason it is negative is albedo is always some percentage, of the light so it is a number less than one but greater than zero. So the log of something less than one is going to be negative, so that's why it is negative. If you take the derivative of this, all right what you see is that you'll just have this spikes where the thing goes up. Another spike, another spike, another spike, right. So the, the derivative of this staircase function is this set of spikes. So remember that L is going to be the product of some slowly, varying intensity. So this is a slowly varying intensity, and then this is the logarithm of it, okay. So it actually went up higher and that's the log. And when I take the derivative of the log, I get this low kind of things. Now, if L is the product of row times I, then the log is just the sum. So it's just, this is just this plus this equals that. And because derivative is a linear operator, it's the sum of this plus that. This, is what the derivative, of the log of the image looks like. This is in this one d case. Okay? So, remember what we want to reconstruct is this and so far what we have ts that. All right so here I've just copied over, what we had before. Right? This is the derivative, of the log of, of L. Well. That's a, image that we can essentially threshold. Imagine thresholding it by its absolute value. If you threshold by its absolute value, you would get this, okay? And then if you integrate that, I'd see you come along, you go up, go up a little, go down. All right? That's the albedo. Now of course, there's a constant of integration that we can't know, because we only had the derivative, so we can't know the absolute brightness. Which makes sense, right? Because if I crank up the e by a factor of, you know, if I add in a constant to it. I can't tell, that it's an e or that it's in the reflect. So there's this constant mitigation problem but that's okay the idea is that you've recovered the reflectance

10 - Color Retinex

And here's an example of it being run in a color image. What you should see here is that there's this color board in a dark shadow. Okay? And here's the same color board in bright sunlight. Right? And the idea is that in the image, if you actually measured the intensity coming off of this white square, and the intensity coming off of that black square, you would actually see the values of 119 in both places. So objectively, those two are the same value. But of course, you and I both say that this is white and this is black, even though they have the same value coming to our eye. Because clearly what's going on is we're saying, oh that's in shadows, so I, I recalibrate that. Well when you run that retinex algorithm that I just showed you, and by the way, running it in two dimensions is a little bit trickier than running it in one dimension. That it says that this thing has a value of some random number 126. This thing has a value of 175, so it is figured out, that the white square is brighter than the black square, even though in the darkness, the light square's going to have the same amount of light as the black square in the sunlight. Another way of thinking about this is if you take a piece of coal, black coal, outdoors in the sunlight, and you look at it, more light is going to be hitting your eye, coming from that black coal than a white piece of paper, inside your classroom. Okay? And yet the white piece of paper looks white and the black coal looks black. And that's because your brain is computing for all the stuff that's going on around it. And this, this constancy, and lightness constancy, we'll talk about it in just a second, is it's just remarkable.

11 - Sharp edges

So as I said these operations are easy in 1D and a little trickier in 2D, and of course it doesnt work at all if you have sharp shape variations. So here is a shape example I think also done by Ted Abelson. So here we have this sharp shape boundary right here, and the question is what is the

relationship between these, this square and that square and actually you can take a look at these edges. Right? Well, you realize that these two are the same. And yet this square probably looks darker to you than that square. And in fact, here we go again. You can see that there's not boundary here, and there's no boundary there. These two squares are the same intensity, which by the way are the same intensity as that. All right? So rednex would fail here because it would assume that this sharp boundary was due to change in an object and not due to change in an illumination. because when you have a sharp shape change you have a sharp illumination change and rednex doesn't deal with that at all. So what's going on here is refered to as sometimes as color constancy or lightness constancy.

12 - Human Color and Lightness Constancy

Color constancy is that the hue of something, the hue and pretty much saturation, under different color, over different spectra or color of lighting, you're still good at seeing. So, you know, you, you go outside and or you go inside where lights are different colors, and purple shirts look purple, green, green pants look green. Purple shirts and green pants together look bizarre. But you know, it's okay, you can still, you can still see it, regardless of the color of the illumination. The same thing, lightness constancy is like, what I was talking about the, the, the black piece of coal always looks black, even though more light is hitting your eye from the black piece of coal outdoors, than from the white piece of paper indoors. So humans are very good at perceiving ,. What color a surface would have looked like if the light had actually been white? So you sort of remove the effect of the illumination. We're pretty good at also being able to figure out what's actually hitting our eyes, so you can say well it's a bluish thing, but I actually see kind of a more purplish light coming in. You can sort of make that distinction. What humans are not so good at is being able to tell you what the color of the light is, because they're brain is spending all this time trying to eliminate, the color of the light and they're not so good at being able to report that. So there are lots of methods that are used to try to accomplish this kind of computationally. One method is. Let's suppose I don't know the color of a light, but I want to render something well on, say, a color CCD image. Right? Well, one thing you can do is you can say, let me assume that the average color is gray. That the average intensity is gray. And so those of you who have auto white balance on your cameras. And most of us with digital cameras do. It's essentially averaging over all that and saying. Let me take a look at all the different average color and assume that it's really grey out there, so anything that's makes the whole average not grey is due to the illuminant and I'll change that. And it works okay it can sort of fail in that. So, you know that tungsten film, right, so if you're, if you're, you know, if things aren't, aren't really great, you're ability to automatically do white balance, is really difficult. There's another really very simple method, assume that the brightest thing I see is white. So that way if the brightest thing I see is actually light blueish, I say okay, that was really supposed to be white. So I'm going to assume the light is kind of blue, and I'll undo the blue light that's there. Okay, those of you that ever done, manual white balancing in cameras, you know that you put out a white card, or white piece of paper, you take an image of it, you say this is a white balance image, and you're telling it, this thing is really white, so that any color that you see in there is because of the illuminant. Pull that out when you're doing the, when, when you're trying to do the color balancing. There are other things that say, I'm going to assume that the colors are supposed to span the entire color gamut. And so if I see the color sort of shifted, let me try to bring it back and fill the whole gamut. So, these methods work sort of okay for photography because, it turns out that even if you get it a little wrong, your brain, when looking at the photograph, will often undo the right thing to begin with. But none of these methods really work well enough for computer vision in order to analyze what's going on out there. It's still really an open problem.

13 - End

So that ends our conversation about lightness constancy. The, as I said, the, the truth is our computational methods are still not really adequate in this space. It could be it hasn't received enough attention to think of it from recovering each element. It's received, actually, more attention

in the computational photography domain. And I think if you take Professor Irfan Essa's class in computational photography, which I highly recommend, I think we spend a lot of time talking about that there are a lot of algorithms related to computational photography that are connected to computer vision. And this is an area where this lightness and color constancy is an area where we need more work. So the good news is there's plenty of research in computer vision to be done. So hurry up and go get your degree and come join us in the field.

5C-L1 Shape from shading

1 - Intro

Okay, welcome back to computer vision. Now this is going to be our third lesson having to do with essentially how intensity and shape, and light interact with one another. In the first one, we talked about the physics, in terms of how how that interaction works. In the second one, we talked about, what if you want to try to recover something about the surface properties of the object, you know, the reflectance function. Basically, you have to make some assumptions about the shape, or, or the illumination. Well, today we're going to turn around the other way and we're going to say, well, how can I recover something about the shape of the object based upon what I'm seeing.

2 - Shape from Shading or Lighting

So this general area of work, is actually started by, I guess, probably Horn and maybe some others, but I certainly learned about it from back, way back in the 70s and continued in the 80s and 90s quite a bit, it's referred to as shape from shading. And the idea is, over here you see a picture of a real sphere this time, right, and you can tell this is a sphere, right with the light source over here. Likewise, you can tell that from this person that, that probably this is the front part facing you and that the shape tapers off. Likewise, you can tell from the shading here something about the texture and the shape of the face. And the, the question is, is how do you use shading as sort of a cue for recovering the shape information? So in order to talk about shape from shading or, I'll call it shape from lighting. Because one of the things that we're going to do in a little bit called photometric stereo, it's not so much a shading variation as it is from having multiple light sources. What we have to do is we have to look at the relationship between the intensity that's seen by the camera and the actual shape of the object. Now, that means we have to look at that reflectance function. Remember what we defined I guess two lessons ago about this interaction between the surface normals and the light direction and the surface itself. In particular we're going to introduce a mathematical construct. Refered to as the reflectance map, and the reflectance map will be the basis of a lot of what we'll be doing today

3 - Surface Normals Math

So, unfortunately. Not unfortunately. Necessarily we have to introduce a little bit of notation and mathematics in order to make it easier to understand, what this reflectance map is. So we're going to start by assuming we have a surface z of x, y. Right? So here we have z in this direction. X and y in that direction. And you have some surface. As you move in say, the x direction and the y direction. Your z value changes and in fact, you can talk about how z changes as you move an x and y, that would be the partial derivatives of z with a spec to x and a spec to y. And we are going to find two variables referred to as p and q to represent these derivatives. Now there is a little bit is ambiguity in the literature. When I learned it from Burtoll, p was the partial of z with respect to x. Q is the partial of z with respect to y. As I was rummaging around the web to see how people talk about this all of sudden sometimes people use the negative of the partial of z with respect to x and the partial of z respect to y. And it turns out when you do it as the negative it makes some formulations a little bit easier. So I'm going to use it as the negative, but you can use it either way as long as you're consistent.

4 - Surface Normal More Math

So, let's assume we have some sort of a surface normal, okay? So, here we have our surface normal drawn and it is on a surface, okay? If we have a point on a surface, we can define two tangents, all right? One is, if I take a step just in the x direction, there's a question of, how does z change? So, if I step in the x direction, no change in y. Then my change in z will just be the partial z with respect to x. And so that's negative p. So tangent in the x direction. So it's on the surface, let's draw ourselves a little surface here. Put a point here. And this is one tangent vector, okay? Well the other tangent vector we can have is we just take a step headed in the y direction then there's how the z changes as a function of y. And that's negative q. And so that would be another tangent. So if I want to find the surface normal, that's easy to do. I just take the cross product of those two directions. By the way, these vectors they're not, I haven't made them unit vectors or anything. They're just taking one, a step that goes 1 in the x direction and changes however much it does in z, same thing for the y. So if I want a unit normal, what I have to do is I have take tx cross ty. So I, remember, the cross product of two, two vectors is perpendicular to that right? So I take tx cross ty. And if I want it to be a unit vector, I have to normalize it by the magnitude. Okay? And so, when you take tx cross ty, you get pq1. And the magnitude of that, of cour, course, is the square root of p squared plus q squared plus 1. By the way, so the reason I used the minus partial respect to z. Respect to x and y. Is, that allows my normal to be p,q, 1. If I do it the other way, it would be minus p, minus q, 1. So, same difference. So that's the algebraic description of p and q. And that's important, because we're going to use it. But it's also important to understand the geometric interpretation. Because when we reason about multiple constraints, it'll be easier to think about that. To do that, I have to introduce something called the Gaussian Sphere.

5 - Gaussian Sphere and Gradient Space Projection

And so here we have a sphere. And the idea of a Gaussian sphere is it represents all possible surface normals. So suppose I have a shape, and I'll just draw us some sort of shape like that. It's kind of a potato looking thing. Right, if I have some point here, it has a surface normal in some direction. Well that surface normal is right here on the sphere. And likewise, if I has a normal here, that one would be right there on the sphere. Okay? So there's a mapping, from all the surface normals to the Gaussian sphere. And in particular if I have z as a function of x y and I'm only worrying about things pointing up. Then essentially, only the top half of that sphere do I have to worry about. Right? So there's going to be a relationship between every normal, and a point on the Guassian sphere. So, the p q space that I was just talking about is actually a projection of the Guassian sphere. And the way it works is as follows. Imagine, and I'll draw this in 3D in a minute. A map by here in the 2D. Imagine I have a, an origin at the center of the sphere and I come out, at the location of that normal and I hit a plane that's located at the top pole of this sphere. So here I'm showing it in, in 2D. It just makes it a little bit easier. But actually to look at p q space, we have to look at this in 3D. So here the idea is, here's my little normal who is there, and it points up until it hits this plane. And that location would be p0, q0. All right. For this particular normal. And this z equals 1 plane up here, that's sometimes called the gradient space or the p q plane projection. And the idea is that every normal, has a p q value. So I can represent normals just by their p q value. And so now we're going to use that to define our reflectance map. Got it, cool.

6 - Gradient Space of Source and Normal

Let's suppose I have two vectors so I'm working from a point I've got two vectors. I've got a vector S, that's in the direction of my source and a vector N, that's in the direction of the normal of the surface. All right. Well those vectors are whatever they are. Their unit vectors are sort of small s, small n. So, remember I can project any vector up on the pq plain, and that's what this is. This is at the z=1 pq plain. So I've got a pq for the normal, out of the surface and I've got like p sub s, q sub s for the vector that is in the direction of the light source. So when I write that I get the following, right? Remember I said that our unit vector n is just p q one divided by the magnitude of p squared

plus q squared plus one squared square root of. And likewise, the unit vector in the direction of the source is this p s q s one divided by its magnitude, all right? So far so good. So we have an algebraic equation of both of those normals. So let's suppose for a moment we have a Lambertian surface, then the only thing that determines, in terms of the geometry here, the amount of brightness that we see is just this angle i, which is kind of lost in this mess here. But there it is, it's the angle between those two vectors. And what I need to know of course, is the cosine of that angle, right? That's the Lambertian surface equation. Well, given all of this algebra, we can represent that cosine as follows, right? The cosine of that angle is just the dot product of those two unit normals. So it's the dot product of n, dot product of s. Which is the dot product of these quantities which is written right there. Okay? So that, that is the representation in pq space of what that cosine of that angle is.

7 - Shape from Shading Input and Output

So now we're ready to talk in general about the problem of shape from shading. So the basic problem is this, given say, one, maybe more, but for now one image, that looks like either this top image or the bottom image, can you, you the computer, recover for me a function Z of x,y. So this, that's why it's going to map under this gradient space nicely. So we're imagining that we've just got a single viewpoint, you can think of Z as coming towards me, and I want to know the height of the function as I go over the image. By the way, this is just one representation of depth, it's a very viewpoint-specific representation of depth. If I had multiple cameras and I had different viewpoints, then the notion of Z as a function of x,y would not be a great representation of the geometry that's out there. And in fact, much later in the class, we'll talk about other depth sensors and things like point clouds, which is a different way of representing geometry. But for here, we're just talking about Z as a function of x,y. So, the fundamental question is, given this single image, can we recover the 3D shape. Well, as we talked about when we did lightness, fundamentally, this is an ill posed problem, right, because I don't know what the underlying albedo is, and I don't, may, not know what the illumination is. So there could be lots of different shapes and combinations of illumination, albedo, that would give the same image. So, as I said, this is ill-posed. In order to solve this, we have to make some form of assumptions. And those assumptions can be, the lighting is known, so I know the direction of the light, maybe I'm assuming that it's Lambertian. And that it's the same albedo everywhere. That would be an example. Or maybe I say that I know what the boundary conditions are. What we're going to have to do is, we're going to have to bring in some, some other kinds of assumptions to make this work.

8 - Lambertian Case

So what we're going to do is in order to figure how to go from an image to the shape we're going to talk about this reflectance map. So the reflectance map relates the brightness I of (x, y). And again I'm just going to talk about sort of the brightness in the image assuming that the image is linear with respect to the light that is hitting it. So I'm not going to talk more about radiance or irradiance etc. I'm just talking, talking about the lightness or the brightness of the image, all right. So the reflectance map relates the image brightness to the surface orientation. Remember (p,q). For a given source direction and a given surface reflectance, okay? So this assumes for the moment that you know the source direction and you know the surface reflectance. Let's first consider the Lambertian case, all right? So here we have our same math again. We're going to assume that we've got some source brightness, which we'll just call k here. We have some surface albedo row. And we're just going to use our same equation as we had before that row times k times the cosine of the incident angle, right. That's the angle between the source and the normal, is what's going to determine how bright our image is. And what we're going to do for the next little bit, is we're going to assume that just row times K equals one. Or another way of saying it is, we're just going to assume that our image intensity is the cosine of the angle. You could make it proportional to the cosine of the angle, but the idea is that it's a fixed number. In fact, later we are going to assume that it's proportional. Clearly what matters in the brightness that we see is this angle, the incident angle between the normal and the light source. Alright, so here is the mathematics of our Lambertian system, and in

order to try to keep everything straight, I've sort of color coded. The normal is pinkish, purplish. Magenta, is that fuchsia? Yeah, I don't know. I don't really know my colors. And this sickly green bluish I don't know, some color. So we got sickly purple and sickly green. But just keep them straight, all right? But the idea is the purple is the surface normal, the green is the direction towards the illuminant. So the dot product that we showed before is just p times ps plus q times qs plus 1 over the normalization. And we refer to that as R of p,q. And this is the reflectance map in this Lambertian case, okay? So this R of p,q. So what is that going to look like in pq space. Well to take, figure that out, we can actually just do it algebraically. But it's actually kind of cooler to do it geometrically. All right. So here we have our light source. And we have some vector pointing up to our light source. Now if I have, if my normal is some theta i away from it, that means that it is in some cone. Around that that light source direction, right. So for that constant cosine of I, or some constant theta of I, that's a cone around the illumination direction, alright. Well, I can project, remember, pq space allows me to map any direction, any surface orientation, to a PQ location, right? So I'm going to project that constant cone up into PQ space and I'm going to get this, this elliptical. It's not really ellipse. I'm just going to get this cone sliced by a plane. And the idea is that everywhere along that curve is the same cosine angle between the surface-normal and the source direction. So that means that it's equally bright. Those are referred to as iso brightness contours in pq space. That's a lot to say, but basically, along that curve in, in pq space, I get the same intensity. Because that curve represents the set of possible angles that are the same cosine theta away from the source direction.

9 - Iso Brightness Contours

Look at that a little more carefully in pqs space. Let's assume that we've got some source pq that's right here, all right. So if my surface normal were pointing right at that direction, so the theta i is 0, cosine i is 1, then my value would be 1, all right. But suppose cosine is 0.9, all right? Well, that would be some, some curve. Draw it around here, right. This is some Iso-brightness contour, so every one of those pqs would give a intensity of 0.9. Going out a little further 0.8, 0.7. Gets even further out, 0.6, 0.5, right? Here's all the way down to 0.3. And then finally, what happens when we get to a point where the surface normal, is perpendicular to the light source? Well if the surface normal's perpendicular to light source, the cosine of theta i is 0. So no light is being ref, is hitting that surface, okay? And so that would be this line here all right, right there. Where it says 0 value, because that's the equation of a line in pqs phase, right? P times ps plus q times qs plus 1 equals 0, that's like a x plus b y plus c equals 0 in the p, but it's in pq space so it's ap plus bq plus c equals 0, so it's a line. So the perpendicular situation is a line in pq space. So, and notice by the way, that p, q is a, the R of p, q is a maximum, right when the surface normal lines up with the light source, as you might expect. So that's our reflectance map in the Lambertian situation.

10 - Shape from a Single Image

How about shape from a single image. So given R(p,q), so that means that we know where the light source direction is and we know the surface reflectance, so we can eliminate that or assume it equals to one. Can we determine p, q everywhere, the surface orientation, uniquely for every image point. So here we have, you know, some random point and maybe its intensity compared to the brightest point is 0.7, so it's somewhere on that curve. But it's somewhere on that curve. I don't know exactly where it is on that, in that p, q space. So no, I can't sort of just simply from this analysis be able to determine the total shape or the orientation at every location. So if I want to do this, if I want to compute shape from a single image. How might we do that? Well, basically this is a, this is an example of an ill-posed problem and we need some additional constraints. We need some additional information. Like the lighting is known or boundary condition, or maybe even other information. And in fact there are sort of two standards moves to make. Basically one that doesn't really work and what that does. The one that doesn't really work which we'll talk about in just a minute is the traditional shape from shading and that is we're going to add some addition constraints. And the one that does work is you actually take more images with different light

sources illuminated, and that's called photometric stereo. That, that has some real applicability. So let's first talk about traditional shape from shading, where you add more constraints. Remember here was our, our question given a single image of known surface reflectance, so we know the albedo with a known light source, can we recover its shape. As we said, that means given that R(p,q) can we determine p,q everywhere. And the answer is well, I have to cheat a little. No not cheat, I have to make some assumptions. Assumption number one, is that for example, at this boundary, which is already drawn in red, I'll do it again. We're going to assume the shape is known, and in fact, what you can imagine is that everywhere at this boundary, because it's falling off, I know that the surface normal is 90 degrees from the camera. So I know something about the surface orientation. Then the other thing that I'm going to assume is that the surface is integrable. What that means is, remember I'm recovering orientation, orientation you're going to give us the surface derivative. Well, the whole thing has to integrate to a depth of value, okay? And then finally, I'm going to assume some sort of smoothness, and I'm going to put this whole thing in some form of an optimization method. Back a 100 years ago, I did a paper Yvonne Leclaire called direct height from shading, I think it was called. Which attempted to not do it quite so much in a, orientation based way, but directly in a height way and take the derivatives, it was a, running out of connection machine. But basically the idea was, to recover the full height from this image, under these relatively restrictive assumptions that you know the albedo everywhere. That you know a the, the, the, either the depth or the orientation at some particular contour. We actually use stereo at the contour to get that initialization, then we did shape from shading in-between.

11 - Photometric Stereo

If you look up shape from shading, you'll see a lot of pictures that look like this. And these are typically synthetic, results. So, given some depth map, you can make some sort of a shaded image and then you can show that you can recover, the shape that's really there. Unfortunately, one of the reasons, they show synthetic results so much is. That the results on real images are somewhat, shall we say, less than wonderful. In fact, there was a paper by mubarak shah and some students I think the late 90s, early 2000s, where I think the phrase was something like, well, it works poorly in the lab and not quite as well as that in real life. You know basically,. And, and why is that? Well the problem is that these assumptions are, overly restrictive. I mean a constant albedo everywhere? That means you get no, variation in the color of the underlying texture of what's there. You know, knowing about the light source, the, the orientation directions, its a very brittle type of computation. So, what can we do instead? Well what we can do instead is we can take more images, all right. And this is what's referred to as photometric stereo. So the idea is, that we take a picture from the same camera location so we can still have Z as a function of XY. Same object, but with different lighting. So here we have the picture taken from the same location, with different lighting directions and what we want to get out, is this. Right? The Z is a function of X, Y and unlike traditional shape from shading with enough lights and sort of careful calibration, etcetera, we can also recover the albedo. The reflectance function, at every point. So now we don't have to make this a really restrictive assumption, that the color or the reflectance of the object is the same everywhere, we're allowed to let it vary.

12 - Solving the Equations

Let's go through this algebraically. So here we have our camera which has some viewing direction. All right, and the viewing direction's not going to matter because we're going to assume a Lambertian system which is what we have here. We assume that it's just proportional to the cosine of the instant angle between the light source and the normal, but the idea is that the camera is fixed. And for each one of our sources, source one we take our first image. Source two, we take our second image. Source three, we take our third image. Now notice, by the way, that rho is in these equations, right? Rho is the albedo, so I'm not assuming I know the albedo. So now I have three images, right? And I can write these as a matrix equation for, this is for a single pixel, right? So at a single pixel, I would have three different values and I could just write this equation as rho. And

these are the three vectors that are the source, directions and by the way, I assume I know the source directions times the unit normal. And the unit normal is the thing that I don't know. So I need to solve for that, all right. So remember we're making this Lambertian assumption. So solving those equations is straightforward, its linear, simple system if we have three lights, I is our three measurements three intensities. S are the three directions to the light source and we know those so I and S are known. And then as we said before n is supposed to be uninormal, rho is the albedo. So together I call them n tilda. So to solve for n tilda is trivial. N tilda is just s inverse I, but of course n tilda is not unit normal. In fact it's magnitude is what? It's magnitude is the albedo, so I can solve for rho, the albedo. And I can solve for the unit normal as well, by justing taking the magnitude of that. Now of course, that's with three lights. And anything that works well with three lights, is going to work better with more lights. So let's suppose we have more lights, like M of them. Well, we get the same set of equations but now, we need a least square solutions cause the system is over constrained and basically solve it here. So M by one is the number of measurements we have. And so, we have to take the pseudo inverse, and the pseudo inverse will give us the least squares solution for n tilda, and then again you can find the albeido.

13 - pq Space

That was the algebraic approach to the solution and it works nicely. In fact that was the math that you would actually encode if you were writing a computer program to do this. But what it doesn't do is give you exactly the intuition of what's going on, in photometric stereo. So to do that, let's take a look at this in p q space. So, let's suppose we have some light source, the first light source, p s one, q s one, and so we'll say that, remember we know it's, it, where it's located, and then if we measured it's intensity, let's say it's 0.7, that some particular curve here. So the other ones are dotted to remind you of them. But the idea here is that the solution for p and q, must lie somewhere on that curve. All right, we don't know where yet, so what do we do? Well, in the old shape up shading we did some funny sort of assumptions and integration and stuff that just doesn't work. Well, here all we do is we say let me put up another light source. Here is a second light source, p q of 2 of s and now it has to lie somewhere on this curve. 'Kay? And you'll notice those curves intersect in more than one place. Well that's troubling it intersects more than one place. What do we do to make it come to one place? Well, we just put down a third light source, all right. And if everything is calibrated and working correctly, then the orientation of that point. And the albedo are recovered at being exactly that location. So, that's the geometric, sort of, description of photometric stereo.

14 - Examples

A, a simple example is shown here. Here is a sphere, you could sort of tell that it's a sphere, right? It falls of down here so we're looking straight down. With five different illuminations. When we do photometric stereo on that, remember we can recover both surface normal and the albedo, and that's what this looks like here, right? So the albedo, you can see that it recovered, the fact that it's got these four quadrants. So even though this thing falls off slowly, it knows that it's all the same albedo. And then this is the estimated of the surface normals. And so this is what's sometimes called the needle diagram, where you're looking at where the needles are pointing, that's the surface normal. Now you can actually do this somewhat for real. So, here's an example where there are four input images, so the lighting is coming from different directions, right? You can see here, you get different kind of shading, right? And it recovers both the albedo, which is sort of the paint, the color. And you can see that it actually knows that there are these stripes on there along with the surface normals. So it recovered the shape. In fact, there's a really cool app you can download from this company called trimensional.com, and this is not a commercial endorsement. It's just a Georgia Tech endorsement. This was an app put together by Grant Schindler where you basically, I don't know if he's got it on Androids now as well but certainly at the time it was on iPhones. And you take pictures from different locations and it uses the light to illuminate it. And it does a surface reconstruction. All right? And it works pretty cool and download it and I think it's it might be free. I know it might be a dollar, maybe it's four. Whatever it is, if you, if you buy it, grant, send me

15 - Real Application

Okay, so photometric stereo is used in a couple of interesting applications. It does have some challenges, like you have to know the reflectance function. You have to know that reflectance map. So typically, it's assumed for Lambertian types of stuff, maybe a little bit of specularity. But, but basically you have to understand the reflectance function. It also has a problem with shadows or interreflections, because it's assuming that the only thing that matters. Was the orientation of the surface with respect to the light. There's some smaller problems like the cameras and light sources have to be far enough away that, that it's a constant angle. That's not so bad. Calibration is important, right? Remember we're assuming that everything is linear in terms of the intensity, with respect to the light that's hitting the camera. So you need a lot of photometric calibration. Before I conclude a couple of things about some shape from shading. One is, there was a really cool use of it. There was a paper, it actually won one of the Marr prizes in 1995 in ICCV, and it was this idea, that namely shape from shading with Interreflection Under a Proximal Light Source, Distortion Free Copying of an Unfolded Book. And it appeared later in IJCV, International Journal of Computer Vision, and what they were looking at. Was this idea that, you know, when you put a book down on a scanner, or a copier, you get this surface that bends away. And you can recover something about that surface structure. And you should also be able to sort of unshade what happens, because the light falls off. Now unfortunately, the images that are in the original picture don't reproduce very well. So you're going to see something that looks really ugly here. Obviously it wasn't pure black in here, but it was just very darkened. And they were able to recover the white and the text, by essentially undoing the shape from shading. And I thought that was pretty cool. Shape from shading also, in the human world of course is much more complicated. A whole bunch of interesting effects. One that's well known is this light source direction effect. So if you see here on the right, you probably see, especially if you're looking at this sort of vertically, you probably see a sphere lit from up here. What most people see, if you show them just this thing on the right, is a dimple, something pushed in, okay? With, again, the light source coming down. Now of course. Figure b, is just figure a rotated around. So, in one you see something bent out towards you. On the other you see something pushed in. But, the reality is they've just been rotated around. There seems to be a human preference for light sources to be up above, instead of down below. And there's a whole bunch of other work on sort of human effect. There is another demonstration of human shape from shading. Which is a multi, multi, multibillion dollar industry, and that's called makeup. So as, and, and there's no way this without being gender specific, so I will say it. As many women know, properly applied foundation, can cause the shape of your cheekbones and other parts of your face and other elements. To be perceived as having either a higher shape, or a more rounded. And what you're doing is you're changing the shading just a little bit, so that the human vision system ascribes a slightly different shape. To the underlying contour. And so it's taking advantage of shape from shading. And you know, good makeup artists know exactly how that works and that's a, that's a interesting shading phenomenon.

16 - End

That really ends our conversation about shape from shading. I'll say that and as I said with lightness, it's still pretty a pretty under-solved problem. Back in the 70s, and 80s where the view that the primary job of vision, or one of the primary jobs of vision was to reconstruct geometry, there was a lot of focus on how. Geometry reflected itself in lightness and to try to undo that process. Remember I talked about vision as being sort of inverse graphics. That whole focus has diminished quite a bit, and the other thing is, the idea of being able to take derivatives and images with our current imaging systems where things are. You know very it's all sorts of noise and kinds of things like that. It just was yielding very brittle kinds of results, and like wise the types of assumptions we had to make were sort of overly restricted. So I don't think my colleagues would disagree and say we basically can't do that right now, okay. So you show us a nice, you know, my coat lying over there, you guys can't see it, the chair, etc, whatever. I can tell a lot of shading information, by integrating many multiple cues. And, we don't really have good techniques for

doing that. There is a question, of whether machine learning is actually a play. That is, maybe our system doesn't actually undo. The physics of the image formation. But instead it has learned, the relationship between certain image patch appearance and shape. And it just does that recognition or that mapping, without undoing the physics. I would say the jury is still out on that. And probably the truth is going to be somewhere. In between I mean, Ted Adelson's demonstrations of that shadow casting, clearly there's a global effect on sort of a local process. And, exactly how we do that is still unknown. So, shape from shading is a very old problem, but we could probably use some very new solutions. But at least now you have some idea of the, sort of the nature of the problem and what we have done.

6A-L1 Introduction to motion

1 - Intro

Welcome back to Computer Vision. This lesson is going to start[br]a whole unit on motion. Now, in the real world,[br]things really move, you know, wind trees blow in the wind,[br]boats move along the water. A baseball player swings his bat and[br]typically misses if he wi, if he plays for the Mets. But, you know,[br]things are moving all the time and our perception system handles them. In Computer Vision, we also want to deal[br]with motion, but what's kind of cool is, actually, nothing is moving. What you're seeing here is[br]a repeated GIF image, and I've, I've made it sort of[br]coarsely sampled actually, it's from some slides stolen[br]from some folks listed below. So that you can see that it's[br]actually a sequence of static frames. Okay?[br]And, you know, of course, the issue is that when we look[br]at them when they're not so separated in time by quite so[br]much we see this nice motion. And what we would like is, that the same[br]perception of motion that we have, we'd like to be able give the machine. We'd like a machine to be able[br]to understand the notion of motion, if you will. So, when we're going to[br]talk about motion. Here, we're going to start[br]talking about video. Now, really, we're just thinking[br]about a sequence of frames. And, in fact, a long time ago, we used[br]to just called them sequences because a camera would go and then a camera[br]would take another picture. And then, some time later, would take[br]another picture, another picture. And the question was, how to use those. These days, of course, much more[br]ubiquitous, is a notion of video where a sequence is captured over time, [br]usually, relatively quickly. So things can't change a whole lot[br]from one sequence sample to the next. And usually taken at regular intervals,[br]whether it's every 30 hertz, 60 hertz, 24 hertz, the idea is that[br]it's at regular intervals. Now, our image is no longer[br]a function of just space, x and y. But our image is a function of space x,[br]y, and time, t. So our signal is I of x, y, and t. But as I said before, sometimes when we describe them as[br]three dimensional function like this. We tend to think of the, all three[br]dimensions as being the same, but they're not. Space is quite different than time, so, if I wearing a nice checkered shirt[br]instead of this gorgeous green. As I moved in space, you would[br]see pixels changing very quickly in terms of their[br]intensity of their color. But, if I were to change[br]the pixels that quickly in time, you would essentially see a mess,[br]you would see all this flickering and things changing very rapidly. So in general, even thought we think of[br]these as a three dimensional function, remember that time and space are[br]different in terms of their statistics

2 - Motion Applications

Let's talk a little bit about the processing of video and some of the applications or not just applications, but the general computations that we would want to do. And then we could talk about how motion could be used in order to, or the motion processing be used to do that. So, a simple one is background subtraction. So you might have a situation where you've got a static camera and a the background is considered to be a static background and I've got some object moving in the foreground. And my goal is to pull out the foreground object, the moving thing from the static background. So, I may be able to track it or model its dynamics or recognize it or something else

that's, that's focused on the object. Another interesting application might be what we call shot detection and that's illustrated here. So the idea is here we have a bunch of frames that are in a video and commercially produced video is usually made up of a sequence of shots, each shot coming from a particular camera. The camera might be moving, it might be panning. It might be dallying moving, translating and then you'll cut to a different shot. And in this particular case, our shot boundary is shown right here. Right? All of these are taken from one camera shot and these are taken from another. Often, it looks to you like they're taken one right after another. But in reality, they changed everything. They moved the camera around, changed the actor. Redid his makeup, he went and got a smoke or whatever it is they're smoking on set. And then they came back and they did the, the next shot and they just cut them together. And if you think about what that would look like to a computer in one case, I might have a nice, smooth background moving with some objects moving within it and then there's this rapid change and I might have another camera, which might not be static, it might be moving as well. And the analysis of that motion might allow you to do the shot detection. So we have background subject, subtraction, we have shot boundary detection. Maybe we have a bunch of objects that are moving in different ways within a scene and we'd like to separate out each of those objects and that's referred to as motion segmentation. Motion segmentation is you segment the video into multiple, what we call coherently moving objects. All right? So they're each moving, but they're moving in an independent way. So here's an example. And what's funny is this is a static image of a snake and a mongoose taken off the web and you could imagine trying to separate these out just spatially. In fact, you can sort of do that. But in fact, the color of the mongoose and the snake is very similar to the color of the background. And if you were to track these things, you would see pixels coming and going. But if you actually were to look at the motion, you would see the motion of the snake as being a coherent thing and the motion of the mongoose as being a coherent thing and you would be able to use motion to segment out those objects.

3 - Motion and Perceptual Organization

This notion of coherently moving objects actually comes from some very early work in vision that was done by, the Gestalt psychologist. Gestalt psychology, which is really sort of the turn of, I was about to say the turn of the century, turn of the previous century, really around 1900 or so, is when they first started thinking about the types of processes of the brains, the brain does. In terms of what actually has to be constructed, if you will. That is, what sort of processing does the brain have to do. Now, it, they didn't really have computational models yet, so it was very descriptive. But it really was the precursor to a lot of the computational stuff, that we all do now. So, one of the founders pictured here is Max Wertheimer. And if you, if you read, it says that one of the things that really interested him was, at a train station, you could see a set of moving lights going. You know, one, then another, then another, then another, and of course these lights would just come on. But his, he and everybody else, saw it as moving, right? Saw these sequence of lights as motion. And his question was, how does that happen? What is the brain using? And he took a bunch of his assistants, who later went on to also become famous Gestalt psychologists, and they started looking at the way the human system groups things together. And they came up with a bunch of rules. So up here we have the not grouped, right? We just have a bunch of black dots. Well it turns out if you just move them a little bit, right? Proximity will cause you to say oh I have three sets of two there. Likewise appearance, though you see that there's black, white, black. All right? Similarity also in shapes, we've got a horizontal ellipsis and vertical ellipsis. But one of the ones that they did that was kind of cool was called common fate and you can see what common fate here means is that they're moving the same way. And then this gets studied tremendously by all sorts of people doing work in psycho-physics and trying to understand how far could a point move between two frames for you to see it moving? And then it turned out that they were actually different things. If you do it a very small amount, you actually see the point moving, right? You actually see that it went through all those little stages. And then, if I put the point further away, it'd be off and then on. You would get a sense of motion but you didn't think you actually saw the point move. You would think it disappeared here and reappeared there. So you had this idea that something move but you didn't

actually see the things moving. That was called a long range process. I think these were done by Oliver who was a famous psycho-physicist. And so anyway, so this is, this idea of how does the brain process such motion? Sometimes, common fate is the only cue you have. So for example, I'm showing you a random dot image. You probably don't see very much there. What do you see? You see random dots. Good. All right? Take it away. Yawn. [NOISE] Well don't yawn, you'll go to sleep. All right, put up another one. There it is. Is that the same random dot image? You have no idea. Okay? But let me go to the next one. This is a gif I made myself, I'm very happy. Now you're seeing two random dot images sequenced right after the other. So what you're probably seeing is a square moving. Back and forth, right and left, all right. You probably see a pretty sharp boundary of that square. You actually think there's like an edge there. Well, let me stop it, no edge, okay. That edge was manufactured in your head based upon the motion field. Okay, and our goal is to understand this computation of motion that would allow a system to say there is a boundary here based upon different motion.

4 - Impoverished Motion

Something else that was observed about moving lights. So I'm showing you this picture, and you probably don't see very much. So you probably don't know what this is. To me it's obviously a horse jumping over a fence. Except when I put it in motion, oh. It's Meghan strolling along. She has such a nice stroll. No it's not, it's just some set of random dots from some other exhibit. Notice that that center point is fixed. It's not moving at all. And, in fact, if I erase that and look at that, and you just stare at that point, the whole thing looks kind of weird. But then if you just stop looking at that point and just look at the whole thing, you see somebody moving. Except now I can't get that weird center point out of my head, all right? But the idea is with this very impoverished stimulus, you are able to perceive not just motion. Remember, nothing is moving here, right? There are no points moving on the screen. Point, point, point. You see them moving. Not only do you see points moving, you see a person moving. All there are are static pictures of points. That's all that's there. The rest of what's going on is being manufactured in your head. And our goal, because we're computational vision people, is to think a little bit about how to get the computer to manufacture some of that information.

5 - Examples of Impoverished Motion Quiz

Can you find some interesting examples of Impoverished Motion from the web? Now these don't have to be videos of dots moving, they can be anything where you have very little to go by, and yet your brain reconstructs and interprets the overall motion. Sometimes correctly, sometimes not. You can paste in links to web pages, videos, or papers.

6 - Examples of Impoverished Motion Solution

Now here's an old video from Cornell University. It shows how the same motion can be perceived differently in different scenarios. Fascinating, isn't it? There is a lot more in this video linked below. Feel free to discuss this and other examples you found on the forum.

7 - More Applications of Motion Analysis

Another example here mosaicing. Now we've talked about mosaicing before. We talked about panoramas about aligning images and bringing them together. Well instead of just a small discreet set of images, suppose you actually have a video. So you're taking your camera and you're panning it around. What you might want to do is make a single image. Okay? And this comes from Michael Irani's work. Of where you can see where all the frames are. Okay you can sort of see these, this jagged line, each one being the edge of a frame. And you realize what happened is the, the camera was swept out this way. All right, and I'm able to make this single image. All right, and If I'm

rotating my camera about its axis, I'm not actually doing any translating. The reconstruction can be perfect because I don't have any motion parallax. And if these things are planes I have to align some points. How many points does it take? It's a homography question. That's right, even Megan got four points for that. And in fact, if I am moving, but things are really on a plane, it's still, four points. So there are other applications of motion analysis. You know, segmenting objects in space or time. Sometimes you can get the 3-dimensional structure if you know how the points are moving, you know which point is which. Just like when you saw the person. You might want to learn the dynamics of how some things move. Maybe you're doing, physiology. And you're watching people moving, and you want to understand something about their gait, and the dynamics of their gait. All right? Recognizing events and activities. You'd like to be able to say, what just happened in the video? So it's not just object recognition, it's action recognition. And finally, you might just want to process the video and understand the motion to improve the quality. So you might do motion stabilization, right, so that you've got this jerky camera and you'd like to get a nice smooth video out. There are simple ways of doing that and complicated ways. In fact, a colleague of mine here Irfanisa and a former, student of mine Veroquatra and some others did some work on stabilizing video especially when you have rolling shutters in cell phones. Anyway, these are all reasons for wanting to process motion.

8 - Motion Estimation Techniques

So for the remainder of this lesson, and then the next one. Oh, and then the next one. Oh, then the next one too. We're going to be talking about motion estimation. Basically, given a sequence, tell me what moved, tell me how it moved. Generally, you can say there are two different approaches to this. The first approach, involves feature-based methods. Now feature-based methods are just what you might think. Extract some visual features like corners, textured areas. Oh yeah, we know all about that. And track them across multiple frames. So that is find them from one frame into another. Oh, we know all about that too. And that will give you what's called a sparse motion field. That is, you you don't have the motion of all the points. You have a motion of a bunch of points that were good to track. All right? But that works really well when image motion is large, so you move like tens of pixels, and you'll have a significant change between frames. The other method class of methods are what are called direct or dense methods. In dense methods, you sort of directly recovering the motion of every pixel of each pixel from the, the fancy work is spatio-temporal image brightness variation. So another way, remember that volume X, Y, and T? So you can think of gradients, you can think of flow within that volume, and what you're trying to do is you're trying to recover that flow based upon how the appearance changes over time. It gives you a very dense motion field, but it can be sensitive to appearance changes. This type of analysis is suitable for video where you're getting samples very quickly. So things don't move very much in time. And, in fact, if things are somewhat smooth in time and space, I could say, well, from one frame to a next, I can talk about how things are moving, like, say, a first derivative. If you're worried that a Taylor series expansion is in our future today, well, maybe tomorrow? Yes, okay. But the idea is that it's good for nice, dense, smoothly moving things.

9 - End

We've already talked a lot about finding features, detecting them, describing them and matching them. And so, we're not going to do much more in terms of that feature matching for motion, okay? Instead, what we're going to be doing over the next couple times is focusing on the dense methods where we look at the derivatives, we look at how things change. And just to sort of spill the beans all the way at the end, the best methods currently, use a combination of the, the feature to get the long-range matches and local gradients in order to get the dense shorter range. So you already know about the feature stuff, so now we need to go over the dense direct stuff. And that's what we're going to be doing over the next couple of lessons.

6B-L1 Dense flow: Brightness constraint

1 - Intro

All right, welcome back to Computer Vision. Last time, we introduced the notion of image motion. We said how motion is actually a, a construct of your head, because there's just a bunch of static frames, and your mind your brain induces this notion of believing that there's actual motion. So the question then becomes, how do we compute that motion? And we talked about motion estimation techniques are, are going to be the methods of doing this. The first one that we mentioned was featured-based methods, where you detect, describe and locate features in your subsequent frames. And we've done a bunch of that in looking at how we do the sift stuff and panoramas and those kinds of things. But the other methods were called direct or dense methods, and that's what we're going to be focusing on today. Direct and dense methods, what they do is they recover motion at every pixel in the image and that gives you a dense flow feel. And it is based upon what we call the spatial temporal brightness variations, how the appearances is moving and it's changing. It can be sensitive to appearance changes but as long as you're getting sort of, video like sequences where you have frames sampled regularly and frequently, it works pretty well

2 - Motion Estimation Optical Flow

The first thing we have to do is define what's called optical flow or optic flow. Because what we're going to be recovering today is the optic flow. And the optic flow is the apparent motion of objects or surfaces or pixels. And the reason I say apparent will be clear. Here what we have is a Rubik's cube looking device, although here it's black and white, so it's just a denuded Rubik's cube and it's been placed on a turntable that's been rotating around this way. If you take a look at these little. Patterns on the side, you can see what the rotation is. I'm pretty sure this was done by Rick Zalesky because he's looked at these gray codes to do the rotation. Now what's a little bit hard to see is that over here are these little arrows that are showing you the motion. And of course the motion is greater at the front of the turntable than say, on the cube, because it's closer to you and it's rotating a further distance. But you'll notice that there are no arrows up here. All right, on the white part of the turntable. Now, it's moving exactly the same way as the part of the turntable that has the, the markings on it. But you can't see it. There's no apparent motion, 'because there's no contrast, there's no texture. There's no indication that the pattern is moving. So, optic slow is recovering the apparent motion. Of objects or points or surfaces. The actual motion, by the way, is sometimes called the motion field. We're not going to talk about that today, we're we'll talk about that not next time, but time after, a little bit. But what the underlying physical motion is. Today we're talking about motion in the image.

3 - Problem Definition Optical Flow

So, let's take a look at a, sort of, a very simple definition of the problem of optical flow or optic flow. So, here, I have four points conveniently color coded, so you can tell their difference, and this is their location x, y at some time, t. But, they're all going to move, okay? So, they're all going to move in some direction, such that at. T plus one, they're at a different location. And that's drawn here. So the points move in the direction of the arrow. You don't actually get to see them doing the movement between t and t plus one because we only get samples of t and t plus one. But that's just showing you how they move. And in some sense, our goal is to actually recover those arrows. So the question we want to ask is, how do we estimate pixel motion from image I, x, y, of t, to image I of x, y, t plus one? We're trying to estimate that motion. So, what we have to do in order to estimate this pixel motion is essentially solve a different kind of correspondence problem than we had from stereo. Kind of, sort of related, but somewhat different here because the pixels can move how, however they want. Given some pixel in Ixyt, so here's one, here's one, here's one, here's one. Look for nearby pixels of this same color in xyt plus one. Solving this problem is what's referred to

as the optic flow problem or optical flow problem. And in particular. We're going to have to look at two words. One is this notion of nearby, and we have to look at what we mean by the same color. What we want to do is we want to look at these terms, nearby and same color, a little more carefully. So, we have two key assumptions. This notion of the same color and nearby. This same color is referred to here as color constancy. And basically this says that a point at xyt if we knew how it moved to some point x prime y prime at t plus one that point will be the same color. Okay? So, and if this is a monochrome image, grayscale image, then that point would be the same brightness. Okay? And that's the color constancy or brightness constancy. Allright? And then the second thing is that we're going to to assume that these points don't move very far from one time to the next time, from t to t plus one. And if you think that's because we're going to do something in the Taylor Universe, you're absolutely right

4 - Optical Flow Constraints

So let's look at these equations a little more carefully. So the brightness constancy equation. What we're going to do is assume we know that some pixel is moving in amount u v. U is the amount in x, v is the amount of y when we get to t plus 1. Okay? The brightness constant, constraint equation can be written like this, I of x,y,t that is, the image at time t, at location x,y, brightness if it's gray-scale, color if it's color, is going to be the same at t plus 1 at location x plus u, y plus v, where u and v are the amount the point has moved in the x direction and the y direction respectively. So that's called the brightness constancy constraint and in fact I can rewrite it like this, 0 is equal to the x plus u, y plus v, t plus 1 image. Minus the original i of x,y,t. All right, that's the brightness constancy constraint. The second assumption was that we get a very small amount of motion, okay? So that basically, u and v, let's assume they're, like, one pixel, or part of a pixel. Or just, things are changing smoothly. Yes you know it's coming. What that means is I can estimate using the Taylor expansion here that the value of an image displaced from x by u and displaced from, from y by v, is approximately, well it's exactly here if I order terms. It's the original value plus the gradient x times delta x, plus the gradient y times delta y. U and v are delta x and delta y, respectively, plus some high order terms. Right? Remember Taylor expansion? If you go infinite series, you get the exact solution, but we're just going to say plus some high order terms. And then when we make those high order terms go away, poof, wow. Isn't that great? I've got the power right here. All right? Then we say that it's an approximation. Okay, so we say that x plus u, y plus v is approximately the original image plus the gradient in x times delta x plus the gradient y times delta y, or v.

5 - Combining These Two Equations

Okay, so now I've got these two equations, I can combine them. The first one is obvious. This is the original brightness constancy equation. The second one after I substitute in the, the Taylor series expansion, there's a little slight of hand going on. What I'm doing is, I'm substituting for this value. Remember it says that I can get the location at x plus u, y plus v, all right? I can do the Taylor expansion, so here's the original x, y at t plus 1. And then there is this derivative part okay, I of x times u, I sub y times v. And I of x is the derivative in the x direction I of v is the derivative in the y direction. Now, students will usually ask me, wait a minute, is that a time t or a t plus 1? Is that a time t or a t plus 1? The short answer? Doesn't matter. The idea is we're going to assume that things are changing so slowly and so gently that the derivative at a particular point is going to be the same whether I look at t or t plus 1. It's a little bit shall we say, dicey? But actually, in the limit, it's true. And we, we only care in some sense about the limit. So, taking those two equations, I've now rearranged things. I just moved this part over to here. And then in the next line, what I did was, this is I sub t. I sub t is this difference there, between I of x, y, t plus 1, minus I of x, y, t. Well, what is I sub t? Well, if I of x is the derivative of I in the x direction, and I sub y is the derivative of I in the y direction, I sub t is the derivative of I in the time direction. It's what's called the temporal derivative. And remember, our imagery is now a function of x, y and t. So, I can take a derivative in x, in y, or in t. Another way to think about is, if I just look at some point x, y and I see how much the, the image changes from t to t plus 1, that's my temporal derivative. And then, finally, on this last one,

you'll see that this here is just Ix times u plus Iy times v. That's just the gradient, Ix, Iy, dotted with however the, the, the point is moving, that's u, v. So, we write that like this, that 0 is approximately It plus the gradient of the image, dotted with this vector u, v. U,v is the direction of motion. Got it? Cool, all right, keep going, now, this is approximation in the limit as u and v approach 0. Now you might, ask how can u and v approach 0? Well, the idea is that that in the limit, of course, if t was very small, the amount of motion you'd get would be very small, you would just have the direction. So we're going to assume that they're very small and we're just going to take this equation directly as is. That, 0 is equal to It, plus this, this gradient. And that, written out either above as a gradient or here in algebraic form, that's the brightness constancy constraint equation. Say that five times fast. Brightness constancy constraint equation. All right. Ixu plus Iyv plus It equals 0

6 - Gradient Component of Flow

So, here I have the equation written, and I've written it both ways. Just the original the way using the gradient imagery dot product way, or in the algebraic way. Question. How many unknowns and how many equations per pixel? The number of unknowns is easy. What was I trying to solve for? U an v, right? U and v is which way is this point moving? Great. How many equations? Well, yes, there are two equations up there, but they're the same equation. I have two unknowns, but only one equation per pixel. This should bother you, all right? It's very hard to solve things that you've got twice as many unknowns as you have equations, because remember, I have a different u and v possibly for every pixel. So I've got 2 times the number of pixels unknowns, and I've got one equation per pixel, 2 times the number of pixels. What's going on here? So intuitively we can think about what this constraint means as follows. The component of u v that's in the direction of the gradient, that's what we can measure, right? That amount will tell us how much the image will change. That's why that's the amount of It I sub t, the derivative, all right? So what this is saying is if this is the direction of our gradient as indicated here, right, here it, suppose this is my real u,v right there. Well any other u, v that has a component of the same amount in the direction, but I've got a u prime v prime, that's in an extra component that's parallel to the edge, all right? And this new vector here, which was the original u, v plus u prime v prime, it has the same amount perpendicular to the edge, in the, in the direction of the gradient, but a different amount along the edge. Remember when we were doing corner finding and we put up the thing and we said, well if you slide it up and down, on fu, if, whe, if your little patch is on the edge of, of the black square. If we slide it up and down, we can't tell at all. We can only tell how much is being moved perpendicular to the edge. That's what's going on here. Locally, we can only tell the amount of motion that is perpendicular to the edge in a little area. When you think about this little area, you can think of it like looking through a little hole. Remember, from our talk about cameras, that that's called an aperture. And this general problem, this is called the aperture problem.

7 - Aperture Problem

So here is a a very interesting line drawing. It's, it's a line with two parts and notice it has a corner. And here's the next one. Voila, it moved. Back, forth, back, forth. I could do this all day, okay? And you can see it moving, okay? Great, let me put an aperture over that. So here is a great big green aperture, all right? And now, I'm going to move that line thing the same way. And now it goes like this, whoa. Now it's moving up or at least it feels like it's moving up. Is it moving up? No. In fact, here I've made it transparent, okay? So now, this is the same motion, right? If you, fact, if you look at the corner, you can tell this thing is moving over and down a little bit. But if you were to look just inside the circle, it looks like it's moving up. Not down and over, that's the aperture problem, okay? You can only tell the motion locally in the direction perpendicular to the edge. There are some other effects of getting, sort of, strong gradient flow. So this is a, single pattern and then I'm going to use Power Point to just put this in motion using animation. So there's no video going on here, this thing is just moving back and forth. And hopefully, as you see this move, you, every time it bumps, you're going to see the inside move relative to the surround. It's not actually moving, that's in your head and that's because of various noise and various ways of computing the gradient.

Like more details and it's not, it's just not the aperture problem. But it's showing the sensitivity in your mind, in your head of the gradient. The aperture problem also is sometimes used to explain the barber pole illusion, right? So it's spinning around but you see stripes going up. And in fact, in the you shouldn't believe everything you see on the web universe. Some people will say that the aperture problem explains the barber pole illusion but actually that's not really true. It has to do with it, it doesn't explain it, because if you think about it, the gradients here are not quite vertical, right? They are off a little bit and so you can get a flow that's like that. And you have to do some integration to decide that the whole thing is moving up. And actually, Ellen Hildreth, a very long time ago and her PhD thesis looked at how you use the normal flow. And you can integrate it in order to make the, the, the, you might see the pole move up. But it is the case that there are places out there that will tell you that this is because of the, the vectors.

8 - Additional Flow Constraints Quiz

Let's say we have a bright edge that we can see at time t. The same edge at time t plus 1 has clearly moved to the right. Now when analyzing the optical flow here, you might think, what aperture? Is there a magical green donut that is forcing you to look at small parts of the image at a time? Well no. It's because with the brightness constance equation, we're trying to analyze each pixel at a time. We look at the image intensity at that specific location x, y. How that changes from time t to t plus 1. The horizontal and vertical image gradients again, at that specific location, and we try to compute the displacement u, v, for that particular pixel. Since there is no actual approacher limiting your view, how would you improve this approach to get the correct optimal flow? Think about what additional constraints you might be able to use. For instance, do you think nearby pixels move together, that is, in the same direction and roughly the same amount? And can this be used to optimize our optical flow computation? Similarly, can we say that motion over an entire image should be consistent? Or should we just look at distinct regions, like corners? Select the choices that you think are valid and useful. Anything else? Mark other and enter it in the box.

9 - Additional Flow Constraints Solution

Yes, nearby pixels tend to move together. This is because an object seen moving in a certain way occupies more than a few pixels. Well, otherwise it would be hard to see. This can be thought of as a local constraint. And we'll see more of this later. Saying that motion must be consistent over an entire image is a pretty bold claim, but it is still fairly valid and useful, especially when used as a soft constraint. This means you're not saying that motion across the image has to be identical, it's just that solutions to the optical flow problem that generate consistent motion are preferable. As you might have guessed, this is a global constraint. Remember that our goal is to compute optical flow at each pixel. That is dense flow. Now if we consider only distinct regions, then how are we going to do that? Well, at distinct regions like corners, the ambiguity of flow direction is small. If we compute optical flow at these regions, we could interpolate the flow between them. So yes, this is a useful approach as well, although it might result in approximate flow. Anything else you have in mind? Don't agree with these constraints? You know where to discuss.

10 - Smooth Optical Flow

All right, so we have a problem. The brightest constraint equation gives us more unknowns than equations. More directions to move than we have pixels. How are we going to recover the motion? Obviously, we need some extra constraints. We need some additional constraints. So, we'll talk about sort of a global approach now, and the next time we'll talk about a more local approach that, that's actually used and, in fact, some of you may even implement. Some time long, long ago in a galaxy nearby Berthold Horn and Brad Schunk did some work on what was called smooth optical flows. And basically they said, let's try to come up with an optical flow that minimizes some error. Okay? So the first part of the error is how much do we violate the optic, the brightness constancy constraint equation, right? Basically, that middle term right there, that's supposed to be zero. Right?

Ixu plus Iyv plus I sub t is equal to zero. So, violations of that means that it's not zero. Sometimes it's positive sometimes negative. I square it, and I sum it up over the whole image. That would be how bad my flow is with respect to the constraint equation. All right? But we know we can solve that exactly because we have more unknowns than constraints, so If that's our only constraint, we're, we're not done. But we're going to call this the constraint error. By the way, I kept partially originally I'd stolen, then I did on my own, I kept the integral signs instead of summations because back when this work was done we'd like to think about images as actually continual, continual functions. Of course, if you were to implement this, you'd actually have to do summations. I'll also tell you the fact that summations are not integrals matters and if you take a look at some of the work that was done in shape from shading, doing it as an integral was actually hard. Doing it in summations was, in some sense, a little bit easier. But for here, the difference doesn't matter, so I just wrote it as being summed up over the image. So here we have our optic flow constraint term. Now we need another term. Well, it says smooth optical flow, here's the idea. Remember when we did stereo? The, the state of the art solution that I showed you with that beautiful recovery? Used a bunch of terms, some that had to do with the data, but it also had to do with the smoothness of the disparity field? That was the idea that surfaces don't change a lot, all over the place. But they tend to have a particular depth or very smoothly. The same could be said of motion, that is, so if I've got something moving around, the motion vectors from one point to the next change slowly. So let me come up with a penalty term that says I really don't like velocity fields where the velocity changes quickly from one location to the neighboring location. So that's shown here. These ux, ui, that's the derivative of u. Remember, u is the x component of the motion field. The derivative in the x direction squared, because it'd be positive, negative, derivative in the y direction squared, same thing as here, which is v sub x, v sub y, the change of v as you move in x, the change in v as you move in y. Again, everything is squared. So the idea is, we would like this thing to be, like, zero, or as little as possible. By the way, what would it mean for this to be zero? Well, for this to be zero, that would mean that all the u's were constant, right? They were all the same value, so the derivatives would be zero. And all the v's would be constant. So if all my pixels were moving with the same uv, the whole image was moving the same direction, this value would be zero. Any other u's and v's will cause this term to have some value, and we want to try to minimize this thing. So now, we have these two terms. We have the constraint equation error term and we have the smoothness error term called e sub c, e sub s and what we want to do is we want to find the uv at every pixel and there's going to be a different uv at every pixel that minimizes this total error which is just the smoothness term plus some waiting factor times the constraint term. And the waiting factor is sort of, how much you believe your data. So, the bigger the waiting factor is the more important the satisfying the brightness constancy constraint is, the smaller the waiting factor, the more the smoothness term dominates and what value it should have is a function of sort of how much noise there is and how smooth you believe the entire thing is. So, this was part of a, a general approach, variational approach where you're going to solve for uv as a function of xy that is, you're going to find the optic flow u and v for every pixel that tries to minimize this error. It's somewhat painful to try to implement systems that can solve for that. Not impossible. But the idea is that you have a global constraint, the smoothness constraint, to help solve your problem.

11 - End

So that's one way you impose a constraint on the flow field in general. It has some strengths like one of the strengths are that it allows you to bias your solution with a particular prior that is like you really prefer smooth or maybe you prefer more. You, you work some constraint into a general overall thing. I will tell you that in general there are much better ways of doing it than this, in particular imposing constraints on a local basis and guess what? That's what we'll do next time.

6B-L2 Dense flow: Lucas and Kanade

1 - Intro

Welcome back to the Georgia Tech Metropolitan Opera where we will be singing about Computer Vision for the rest of the day. Because I'm, like, going nuts already recording all these lectures, and I have to do something differently. However, as my children will tell you, my singing about anything is generally a bad idea. So, let's chat instead. So last time we were talking about motion this is within this general context. And we talked about that, if you're looking locally, you can't actually tell the direction of motion of a pixel. And that was referred to as this aperture problem. Remember this little demo, where we said that even though the actual thing is moving sideways and down a little bit, you see it as moving upwards. Because the only part that you can see that edge, all you can tell is the direction of motion in the direction of the gradient. And we showed that here, that that was derived from our brightness constraint equations. Where we showed that the u, v dotted with the gradient has to be the negative of I t, that's what that equation says. But an UV, any motion who's component perpendicular to the edge along the gradient is the same, would be an acceptable u, v. So we needed some additional constraints. So, last time we talked about int introducing some global requirement. We talked about a smoothness constraint that would allow us to optimize the solution over the entire image. But today, we're going to look at more local and frankly, much more effective or certainly more commonly used methods.

2 - Solving the Aperture Problem

So how are we going to solve the aperture problem? Well, look, the basic idea is the same as before, we have to impose some sort of constraints, but this time we're going to impose some local constraints to get more equations per pixel. So for example, We talked about maybe we'll assume that locally right around this point, the motion field is very smooth. In fact how smooth do we want to assume? How bout we assume that it's actually the same value. That is over a window, we're going to assume that u v is the same value everywhere. So if we were using, let's say a five by five window, so there's 25 pixels, if we were assuming that there was one u v for all of these, the uv for, let's say, the center pixel. That would give us 25 equations per pixel. And here they are so, what we've got here is, here's our u and v that we're assuming is the u v of some center pixel. And here we have 25 rows of the derivative on x and the derivative in y at each of those 25 points. And th, this, these are the gradients, dotted with u v equals the negative, and here is the temporal derivatives at each of those points. So I can write that as a d equals b, where d is this displacement vector u v and b is just this 25 by 1 vector, that is essentially the negative of all the temporal derivatives. Okay, so we have this equation a d equals b and it's got 25 equations and two unknowns. All right, well, so that's a little bit of a problem. How do we solve a system when we have more equations than unknowns? Well We do our standard least square solution. When we have an equation like this, we say, well, we're going to minimize the square difference, okay. And they way you do that is by using the standard pseudo uh,inverse method. So we multiply a by a transpose. And since a is 25 by 2, a trans-flows is 2 by 25, so a transpose is just a 2 by 2 and d is a 2 by 1, and a transpose b is a 2 by 1, and we end up with this equation right here, okay? And these have the terms written out. And these sums are summed over whatever the window is it that you're summing over. So it's five by five would be over all those 25 pixels and over here, you just have the product of the, the sum of the product of the, image gradient and times the temporal gradient. This is our, what we have to solve, and by the way, this technique was first profos, proposed by Luke Scigotti way back in the dark ages of computer vision, 1981, which is not quite the dark, dark age. It was the dawning of it. The dark ages were in the 60s. All right, we just did that. We can solve our squares and come up with a solution, but lets understand exactly what solving this means, okay. So you remember the aperture problem, right? And so here it's drawn out. That the idea is if I have a gradient in some direction, and so this is my little gradient there. Right? So that any u v that's along this line would be an acceptable u v. That is any u v whose component in the direction of the gradient would be

acceptable. And that's what this says. That there's a single equation for that. And that's the equation of a line in uv space.

3 - Combining Local Constraints

Thinking about UV space, if we have a particular gradient, that gives us a line. That's what this red one shows us. But suppose we have another gradient, okay, but in the same little window. Well, that gradient would give us a different line. So here is our, this is gradient 1, this is gradient 2, all within the same window. And then, here's gradient 3, and it gives us a different line. Clearly, what's going on, is, if I have a bunch of these different gradients, they're all going to intersect at, at the point. That point is the correct UV. So, as long as we have different gradients within the window, we can get a solution. We've seen this before, haven't we? Here is the equation we just showed you, A transpose A. And d equals negative A transpose b, all right? When can that be solved? To just solve this, we would want A transpose A to be invertible, right? because if it's invertible, I just multiply both sides by that, and I come up with a solution for d. Okay? Well when are matrices invertible? Well they're, and from our linear algebra we know that they need to be what's called well-conditioned. If they're not well-conditioned then the inverse is unstable, and basically you can look at the eigenvalues, all right? And the ratio of the largest eigenvalue to the smog, the smallest eigenvalue shouldn't be too big. That's from your conditioning. In fact, if the second eigenvalue were 0, l1 over l, or lambda 1 over lambda 2 would be infinity, and that'd be awful, all right? So what we want are eigenvalues that are, you know, approximately the same magnitude, okay, so that will make it solvable. But wait, we also know when this should be solvable from the geometric demonstration we just did. This whole problem should be solvable, as long as there are different gradients within the window. because if they're different gradients within the window, then we get those inner sections of those lines in uv space. So does this remind you of something, gradients and eigenvalues? Of course it does, because you are like, right on the ball, all right? You remember our Harris corner detector? This matrix, A transpose A, that is the second moment matrix of the Harris detector. And if you remember, we looked at the eigenvectors and eigenvalues of that matrix to know whether or not something was a good corner.

4 - Interpreting the Eigenvalues Quiz

Memory check. If lambda 1 and lambda 2 are the two eigenvalues of gradients in a region, when do you think that region is good for tracking motion? That is, when do the optical flow equations become solvable? Is it when lambda 1 and lambda 2 are small or when lambda 1 is much greater than lambda 2? How about the opposite? That is lambda 2 is much greater than lambda 1, say an order of a magnitude larger. How about when they're of comparable magnitude and sufficiently large? Remember, select the cases where the optical flow equations become solvable. That is, you get definitive values for u and v.

5 - Interpreting the Eigenvalues Solution

We've seen this before during Harris Corner Detection. The answer is when lambda one and lambda two are both fairly large and of a comparable magnitude, it means that there are multiple gradients in the same region that can be used to locate the gradient direction correctly. Right, you remember this picture here, where we take a look and we say, well, if both of our eigenvalues are small, then this is a flat region. There's not much. If only one of the eigenvalues is big, then that's not good, and here, lambda 1 lambda 2 is in the order. If we make them lambda 1 always being the biggest one, then, then of course lambda 2 is bigger than lambda 1. But the idea was that we were only in a corner if both lambda 1 and lambda 2 were kind of large, all right? And remember, we said that we could, we didn't have any decent corners here in the blank area that was in the sky? Well, guess what? You can't really tell the motion either, right? Likewise, we said it wasn't a good corner if you were along an edge because the corner could slide along that edge, and we wouldn't where it is. Well, movement happened along the direction of the edge, and we wouldn't know how

it moved. So, where do we, where can we estimate motion really well? Well, we can mo, estimate it really well in this textured region. Basically the same constraint that we had on the Harris corners namely to have to have the gradients varying over the window and that's indicated by having two reasonably large eigenvalues of that second moment matrix. That's the same thing that applies to motion estimation.

6 - RGB Version

One quick note. If, instead of doing this in black and white, you were actually doing this in red, green, blue, in a colored version, right, you would now have, instead of 25 equations, you'd have how many? You'd have 75 equations, five times five times three. Now, if you're really shrewd, you might say wait a minute, wait, wait, suppose my window was only a single pixel. You told me professor Bobbitt that we had this problem, okay? That we had one equation and two unknowns. But with R, G and B I now have three equations, and two unknowns. Can't I solve it? Well, no Frodo, you can not, all right? The problem is is that the R, G and B images, you can just think of the image planes, are quite correlated, okay? So, you know, you have a change of intensity but you had the same changes of intensity because of the edge of my shirt, both in the green and in, and in the blue channels. Now one might get brighter and one might get darker but the gradients will be in the same direction. So the gradients are very correlated, so you can't just use the different color planes in order to solve it. Good try though, sort of.

7 - Errors in Lucas Kanade

So what I just showed you is the basic Lukas-Kanade method. kind of elegant. Basically, we've made a lot of assumptions, and these assumptions can give us errors. The biggest error we have, that is the assumption that we made that's, in some sense, the most wrong, is the idea that the motion is small. Okay? Often, motion is large. And when motion is large, a couple of things are wrong with the Taylor expansion. One is even if things are nice and smooth, they may not be linear. So we were doing a first order, Taylor expansion, not a second order. No second derivatives here. So that assumes that things are planar. Well, suppose things are curved a little bit. Well, we, we're going to handle that. In fact, we're going to talk about that in a minute. By doing something called iterative refinement. Which is, we're going to assume it's a plane, solve it. Move the picture over a little bit. Do it again, do it again, do it again, until we've converged finally to the, to the right point. And its, allows us to handle the non-planar component. But in some sense, even much more worse. Much more worse? Even worse than that, is maybe, I've got a nice bright pixel here and then the bright pixel jumps over a bunch of dark pixels. So it's not even, and then there's some other, so it's not even local anymore. That is, I can't find my, my ni, the, where I went by just sort of following my derivatives. And the way we're going to handle that is with a course define method which actually we're going to do next lesson. Today, the on, the, this lesson, the only remaining thing we're going to talk about is this iterative refinement. So, iterative refinement is designed to handle the fact that things are not exactly tangent. And so, here's how it works. I'll write down the algorithm. And then it's a lot easier to see in a picture. All right? So, the first thing is, the way you do it is, you say okay, I'm going to pretend. I'm good at pretending, that it's just going to work. All right? So, I solve for my velocities using the standard Lukas-Kanade. So I find, I assume linear Taylor approximation. I solve for u, v. Great. So, now I've got a u, v at every pixel. Now, it says I'm going to warp It to It plus 1. Okay. We talked about doing image warping when we were applying transformations. And those of you who are going to do the problem set, you'll probably get to know and love how to do warping. But basically, the u, v says how to move every pixel from one image to get to the next one. So you know what? I take this picture and I move all the pixels. And if you remember, we have to do this backward warp so we don't end up with holes. If you don't remember that, go back to the transformation lecture where we talked about warping. But the idea is that we essentially distort the, the t image to move to where I think all the pixels go. So it's almost the t plus one image. But it's not exactly the t plus one image, is it? It's close. So what do I do? I just do it again. I take the image that I created and I compare that to the real t plus one image.

And I get a small amount of more motion. I could actually add that motion to the original motion. That's the new flow field. Well, I can take that one I created. Move it a little bit. And what do I do? Do it again. Until the pixels stop moving, and that will it, that will converge to a solution. So that's really easier seeing here graphically.

8 - Optical Flow Iterative Estimation

So, here I have two curves, okay, f1 and f2, and those are meant to be as taken in two different times. And, here I'm just looking at this point, x0, and, just so you understand, this is the value at, if you will, time two. This is the value at time one, and that's true for all of these. And the question is, how much does the curve move? So, what I have to do is, I want to find this displacement at x0. All right, so do it as follows. So, here's how we do it. We initialize, since this is iterative, we have to initialize. We'll first start by assuming that our displacement was 0. That is u and v are all 0's everywhere. That's great, that makes the work really, really fast. I don't do anything, it's the original picture. Now I have to estimate the next one, d1 by figuring out what the new displacement is. That's what this new little delta d is going to be. So how do I do that? Well I have to compute these tangents and you can see the tangents are right there, okay? And what I do is, based upon that tangent, come down here. It says oh, okay, if this thing were actually a plane, this amount, that's my d hat. That's the amount I have to displace over in order to line up these points. That's assuming this thing were a plane. If we really were a plane, that would be the amount that I would have to move. So let me do that displacement and I end up there. So now I've got my d1 which was just d0 plus my new estimate. And guess what? I have to do it again so now I take a look and say I have to move this amount. So I add that to my d, and I move it that much. And guess what? I do it again. Estimate tiny little d hat, this time. And voila. It turns out that f1 of x minus d3 is almost exactly f2 at this point. Okay? That's the iterative refinement. And the only thing that it takes in order to do this iterative Lukas-Kanade beyond the regular one is this warping step. Given some UV that you've computed between one image and the next, take that first image, warp it through that, that flow field. And now you'll have almost the second image, and you just apply it over again

9 - Implementation Issues

Couple of implementation issues. In general, warping is not, warping is one of those things that in principle should be really easy. In practice you always make mistakes, I will say that MATLAB does a very fine job of, of allowing you to do warping in particular ways so if you are using MATLAB or Octave it's easy. And I think within python nonpie openCV accomplishes as well. The other thing, in order to make this work and actually we'll get to this when we do the whole multi-scale. Is its often useful to sort of take a nice bit of a low-pass filter of your images. Smooth things out, why do you want to do that? To get sort of your smoother derivatives as, as things work. And we'll talk more about that next time.

10 - End

That ends the, the, the basic discussion about. It was actually done for stereo reasons which is kind of funny because in stereo you know that points have to move along the epipolar line. Whereas in motion they can move however they happen to move. But it was originally derived in a stereo formulation. It's actually used a lot more now in the whole process of, of doing motion. To actually make work, you, you're going to have to do things hierarchically, that is, you are going to do things at a core scale and then get to a finer scale and a finer scale. And, it's a really good example of doing multiscale processing on images, and that's actually why we're going to do it next time

6B-L3 Hierarchical LK

1 - Intro

All right, welcome back to Computer Vision. And when I say back, I took a whole week off, so basically, I have no idea where we are. So hopefully you'll be more got your act together better than I do. I think last time we were talking something about the Lucas-Kanade method of solving for optical flow or for motion. And if you remember, we had this problem with optical flow that we refer to as the aperture problem. The aperture problem was if you look at a little local area and you can only tell the motion that is in the direction of the gradient perpendicular to the edge. So, basically to solve that, we had to compose some form of constraints. First we did the global constraints. Then we did the local constraints of Lucas and Kanade. And the idea was by having these local constraints, we could have more than one equation per pixel. And the basic idea was to pretend that all the pixels were moving in the same direction in a little, local neighborhood, so we would sum up over that window. We said a five by five window. You'd have 25 equations per pixel. And that basically gave us this least square solution, and we related that to the whole Harris Corners, where we said that if the matrix, here A transpose A. If that matrix was well-conditioned, which meant we had two decent sized eigenvalues it meant that the gradient occurred in various directions over that window. And we mentioned how this method of recovering how something moved was first proposed by Lukas and Kanade in 1981. But, critical to the whole derivation was the idea that we use this Taylor series expansion. That is, that we could approximate how something would move or how much the intensity would change by essentially fitting a planar ramp, that was the first-order derivative approximation. Fitting a planar ramp to a point, and then as the point would move, the intensity would change according to the slope on that ramp. But, of course, there were certain assumptions built into Lucas-Kanade that were sometimes problematic. So the main assumption was that the pixel hasn't moved, didn't change it's color, didn't change it's intensity. So if brightness constancy doesn't hold, then you have to do something in terms of feature detection. Finding a feature from one place to another, and we've already talk about doing SIFT detections and characterization, in order to do these, these matches. And we can combine those. In fact, we're not going to but I'll talk at the very end about how you can combine dense estimation with SIFT estimation. The other thing we said is that possibly a bunch of points are moving one way and another points are moving another way. So, in that area, it, within a window you don't actually have all the points moving in the same direction. And you have to solve that with a motion segmentation. We mentioned that, we'll actually go over, sort of, a mechanism of doing that, not at the end of this lecture, but at the end of the, the next one. Oh, these aren't lectures, these are videos, so that makes them sound like they're more fun

2 - Reduce the Resolution

But, the most fundamental assumption of Lucas-Kanade, or at least the one that in some sense is most problematic yet, fixable is this idea that we can use that Taylor series approximation. So, the idea of that Taylor series approximation is that we can make a small ramp estimate that is the first linear first order linear estimate. And then as things move a little bit, that'll tell us how things change. And, we've gone over one method. We talked about non-linear iterative refinements. So that if it's, if it's still the right model in terms of the point nearby is changing, but it's not linear. Maybe it's quadratic or something. We use a bunch of linear successive approximations. But actually, that's not the problem that exists mall, most often. The one that exists most often is that, actually, pixel motion is not small. And so, you can't just assume that you can tell how things have changed by just doing a very local analysis. So to solve that, we have to do something called coarse-to-fine estimation. To illustrate what I mean, here's a, a simple example. So, here you see on this one, on the left side, I have curves that have been shifted by only a small amount. So, when I find one point value, the nearest match is actually the correct displacement. So, they actually fit a little slope through here. I can do my local estimate, and I'll eventually find the, the correct match.

But over here, I've got a different situation. So, there is one point, the nearest match will now be this red point. All right? So, it basically has just estimated the change to be that much. All right? Where the actual shift of course, is much, much more. All right? And so, the estimated shift is, is too small. So, the question is how would you fix that? So, let's take a look at this on a real picture. So, here's an example on some real pictures. So, this is two frames taken out of the MPEG garden sequence. And, what you can see is that I've got one frame, and then the next frame the tree moves to the left quite a bit because the camera has moved to the right. And, you can see the tree in the front is occluding the tree in the back. Okay, so question. Do you think that this is a small enough motion to be able to run Lucas- Kanade? No, look, there's a lot more than one pixel or a small number of pixel motion with a planar assumption between. I'm cov, I'm covering entire trees in the background. All right? So, the problem is, you know, how could we solve this, right? How could we eliminate this problem of large pixel motion? And, I'm going to show you with a single slide. You're going to look at it, you're going to go, aha! I get it. Ready? Now. Aha! I get it. What do you get? Okay, here's what you got. In the top left corner is the original sequence, all right? And, you can see that the thing is moving a whole bunch of pixels. I don't know, maybe it's moving let's call it Sep. On the top, right you'll notice the picture is drawn the same size, but I've actually reduced the pixels in half. So now, it says there's, like, 90 pixels across, right. Before, there was something like 180. Well, gee, if I was moving seven pixels before, now I'd only be moving three and a half pixels. Do it again. Now, I've only got 45ish pixels. Bottom left hand corner. Now instead of moving three and a half half of three and a half is one, one and three quarters. Okay? That's still more than one, isn't it? Yes. Okay. Well, cut it in half again. So now, I have this picture on the bottom right. And if you look at that animation, you'll see, you can sort of see it shifting within the pixels. It's actually less than a one pixel motion. Okay? It's a very blurry picture, and I had to make it really, blow it up, and now you can actually see each pixel in there. But, you can see that the motion is a sub-pixel motion. So basically, what we want to do is we want to implement a motion recovery method that takes this coarse image. And then iteratively, or I should say multi-scalewise, goes to a finer and finer image.

3 - Multi Scale

This is how it works, right. So I've got my one image and my other image, which is why they're called image 1 and image 2, all right. And, the first thing we're going to do is we're going to build what are called Gaussian pyramids, all right. And that's illustrated like this. So it's Gaussian pyramid of image one. Well we'll talk more extensively in a minute about how you might go about building those pyramids. This is just mostly so you understand what is going on. So now I have these reduced pyramids at the top level. Remember I have that very blurry picture that actually only has a small number of pixels in it and the motion between them is actually much smaller. And that's shown here, right? So here, it says our u is ten. So that's saying, let's suppose the shift is ten pixels, at the lowest level. Well when I cut it in half, it becomes five. Then it becomes two and a half. Then it becomes one and a quarter. I can keep going. So this is how I reduce the amount of motion that's present. So because I've reduced the amount of motion, I can first run Lukas-Kanade between those coarsest level. And I might do the iterative thing, that's the thing we looked at last time where I set a plane, estimate it, shift it a little bit, estimate it, shift it a little bit. Keep doing til i, til I converge. So that will map the motion between the very reduced image 1 and the very reduced image 2. Right, so I can, I can compare that. I can find the motion between them. The question then is, what do I do next? Well what I have to do is, I have to go from the coarser to the next finer level, okay? And what's written here is warp and upsample, all right? We're going to go through the details of that, but basically the way you can think about this is, I compared something at a coarse level. Then I know, okay, if I were to double up this image and double up that image, I would have to double up the motion field. I would also have to double the, the, the amount I'm moving. We'll do that algorithm in a minute. But I could apply that to the next level and what that will do is, that will get me the next level very close to the second image. At that level and I could run Lucas and Kanade again, and then I can iterate and run that over again and eventually get the entire motion field.

4 - Image Sub Sampling

Couple things to show you ,first let me just show you this kind of works. Allright, here's some optic flow results taken from a computer vision class 2003 from a actually it's a tutorial. Here listed on the bottom and you can see that when you run loops at the, on that front tree without kermits, you get sort of terrible results and while when you run it with the pyramid you get a much better result. And our goal is to really get that much better result. But before we can do this we have to talk a little bit more about multiple scale. So we been having all this multiple scale stuff. Now this isn't going to surprise you that it relates to some of the aliasing stuff and frequency stuff that we talked about, oh God, sounds like, found, feels like months ago. It's probably months ago. I have no idea when it was. Anyway when we were talking about Fourier analysis in different scale. And what we're going to do now is take a little detour through multi scale and then we'll come back to Lucas and Canoti. Allright, so, we talked about aliasing, and now I want to show you aliasing in, in terms of image sub-sampling, okay? So image sub-sampling is what, we talked about it, you just throw away some number of pixels. So here we have a a nice, I sure hope it's in public domain, picture by Van Gogh. This is one where you can see his ear. So on the other side, I guess, there's no ear. You know about that, right? I'm going to assume you have some experience with Van Gogh. Anyway, okay, fine. So, the picture on the left is original picture where I've thrown away every sample. Maybe you can't tell. Picture in the middle I've thrown away. Every other sample, again, both horizontally and vertically. The picture on the right, I've done it again. Eve, I can tell, you probably can't tell, I can tell it's, kind of, looking kind of crummy. And, in fact, if I blow up that picture on the right, I see this. Okay? And what you're seeing here, the, the junk that's in here is caused by this, this aliasing, like just what we talked about last time and, when I zoom it back up, you can see that you've brought these high-frequencies in there and there's all this stuff in here that just doesn't seem like it's the blurry version of what was in the original and that's because it's not. It's an aliased example. So, how do we handle aliasing? What we talked about this last time. What we have to do is we have to filter and then sub sample. So what I've done now is I've taken a small Gaussian smoothing in the original sub sampled. Then smooth it again and sub sample. Then smooth it again and sub sample. And now when I blow it back up you'll notice that the Gaussian sample then that actually looks like a blurred version of the thing on the left. Okay, and that's because I haven't done any aliasing. So to do this multi-scale stuff the right way, I have to sort of do this blurring and subsampling.

5 - Downsample an Image Quiz

All right, let's do this in code. Write a function to down sample an image to half its size. You can do this by picking alternate rows and columns. Now write another function, this time blur and then down sample the image. Here is a test image. I wonder how long you can sit on that at a time. Ok, to relieve that guy of his misery, let's down sample the image. This should be half the original size that is half width, half height. Once more, and a third time. Let's do the same with the blur and down sample function. All right. Now we have two images one eighth the original size. Let's view the down sampled image. Magnify to the original size. And let's compare that with the other image. Implement the two functions and compare the results. You know what they look like.

6 - Downsample an Image Solution

The first one is very easy. Octave and Matlab make it really trivial to specify indices repeating at a given step. In this case, we would like to start with row 1, step 2, and til the end. Exactly the same for columns. This selects rows and columns 1, 3, 5 and so on. Okay, now for blurring. Remember how we used imfilter? Pass in an image and a filter, defined here using the fspecial function. As mentioned, we want a Gaussian of size 5 by 5. All right, so this is our filtered image. We simply used the same indexing as before. Make sense? Let's see what it looks like. All right, here's the result of only down-sampling. And this is with blurring and down-sampling. This appears to be a more natural blurred version of the original image, whereas the former has ugly aliasing effects.

Note that the result is being affected by how the resize function works. You can control how it functions, using a third parameter. Here, passing in nearest means no interpolation. As you can see, this is a much more accurate representation of what the pixels in the one eighth size image look like. By the way, this is the downsampled image. Compare this with the blurred and downsampled version. It still wins. You can also use linear or bicubic which is the default. Try these out, we will be using downsampling to build Gaussian pyramids next.

7 - Image Pyramids

Image pyramids. The phrase was popular back the late 70s, early 80s. Paper that made it really famous was the Burt and Adelson, same Ted Adelson that I showed you an example of from before, in terms of some visual illusions. And they built something called a Gaussian Pyramid is just this reduction just like I told you before. One of things we can do with Gaussian images, or these reduced images, is that we can also create something called band-pass images. And having this ability to do both Gaussian and band-pass let's you deal with scale very well, which is why I want to show you this a little bit here. So, what you have here is, it says Gaussian pyramid and it is a Gaussian pyramid, but it's blown back up again. All right, in fact, go, remember these pictures, where I'm showing you the different Van Gogh's and it says half resolution, quarter and eighth, but they're the same size? That's because I blew up the eighth-sized one, back up to the original size, so you could see it, right? So it would look the same way, all right. So here, I did the same thing. This our Lena picture that we talked of be-, before. And these are, this is the original, then blurred, and sub-sampled, and blown back up again blurred, sub-sampled, blown, again, again. So these are the lower lower resolution images. What I can do is, I can subtract one picture from another to get this difference. Right? So this is a difference image. Two things, first of all this is called the Laplacian and you should know why. If you remember when we did the detections, and we first did Harris corners, and then we talked about the SIFT detectors. David Lowe was talking about using extrema points in scale space. And we said, well, one of the things you could do is you could run a Laplacian over different scales. Or you could take different Gaussians and subtract them. Those were the difference of Gaussians. And difference of Gaussians was approximately the same as a Laplacian That's why these images are called Laplacian images, okay? They're also approximately band-pass, and we talked about this just a little bit when we talked about Fourier analysis. But the idea is that the lowest frequency, if this is frequen, i, frequency here has them fall off like that. So the next one, if it had higher frequency, might look like that. And if I subtract these two, basically I'm mostly going to be looking at that space. So this is the power of the frequency. And so when you take a Gaussian blurred image and you subtract from that another Gaussian blurred image that's close but not the same amount of blurring. You get this section of frequency that you're mostly, that you're looking at. It's not totally band-pass, cause stuff leaks out this side and stuff leaks out that side. But that's why they're called, in quotes, band-pass filtering. So these images here, these are called the Laplacian images. What's kind of cool about the Laplacian images is you can actually use them to reconstruct the original image if you have one thing. So what's that one thing you have to have? You have to have the power. No. What you have to have is the smallest of the original blurred Gaussian, the top of the Gaussian pyramid. The reason I have to have that is, if I take this and I add back in this, I'll get this one. Here let me draw in Lena here like that. Put in her hat and her eyes and her nos Yeah, there she is. All right. Well once I have that, I can add that to this and get a finer version of Lena, and then I can add this and get back the original Okay? So the Laplacian pyramid plus the coarsest of the level from the Gaussian pyramid allows me to do reconstruction. All right. Keep going on this a little bit. To show you how to do the construction of a Laplacian pyramid. because you're going to use exactly the same steps for building a Laplacian pyramid, you're going to use to run hierarchical Lukas and Kanade. That's why we're doing this, so we can get over here. You got that? Good.

8 - Computing the Laplacian Pyramid

Here is an awful picture. But, its picture comes from the paper and I'll show you what we're doing. So this is our original picture. I equal G0. And in Laplacian pyramid, the zero level is the finest level. And that way, you always have the zero level, and one would be filtered down once, level two is filtered a second time, level three. And you can filter as many times as you want to, up to n. For n being some big number. Okay, but the idea is you start at zero. Sometimes people don't like that. They typically want the, the zero to be this most compact one. But that doesn't make any sense. because really smart guy. He wouldn't do that. You start with your original image of zero and then you, you keep filtering it down. All right. So, that's our original image. The first thing we have is this reduce operator. Okay? And we're going to go over these in some detail, but the reduce operator essentially takes a Gaussian, that's the convolution signal, blurs it, and then this down arrow? Down arrow here means down sampling. So just taking every other pixel. That would give you level one, do it again, you would get level two, etc., etc., etc. That one's sort of straightforward. And like I said, we'll go through the numbers in a minute. The one that's a little less clear is the expand operator. Remember we have to expand these things up so if I expand up the coursed one and then I subtract that from G1, that gives me L1. That gives me the first Laplacian, all right? The expand operator is a little less intuitive than the reduce operator. Let me show you what they are.

9 - Reduce and Expand

The Reduce operation is pretty clear. So the idea is that, this level, we have a full resolution, or, or whatever level, and then we want to take the next level down. So here's a way of thinking about it, we'll just number these. One, two, three, four, five, six, seven, eight, nine. Nine pixels across. I actually don't have to produce the pixels that would go under two, four, six, and eight. Because I'm just going to, I only want half the number of pixels that I had to start with. So in fact what this does is, it simply says I'm going to take some location, here I use the middle, and I'm going to take what's called a 5-tap Filter. And here are the filters, see it's one-sixteenth, four-sixteenth, six-sixteenth, four-sixteenth, one-sixteenth. And that's, that's what these numbers are, one-sixteenth, one-fourth, three-eighth one-fourth, one-sixteenth. Okay? That's the filter that I use to get the odd pixel here in the middle. Now that's just in one scan line. I'm working with images. How do I do this? Okay. Go way back in your notes, because I know you all are taking co, copious notes. I almost said y'all. When I say y'all, Megan's going to throw that monitor at me. Okay? Nothing against my friends from the south, but I come from New York, went to school in Boston. If I actually said y'all, I think I'd get shot by my relatives. Anyway, You all have seen this before. We talked about Separable Filters. Right? So a separable filter was, instead of applying a square filter, to the filter, I apply one, one dimensional filter horizontally and then often it's the same one dimensional filter vertically. So go back up and remember about separable filters. The reduce operator uses a separable filter, a 5-tap separable filter first in the rows, then in the columns. All right. That one was straightforward. The Expand operator is a little bit trickier. And you can tell that by looking at this awful picture. In fact I'm going to pull that awful picture off by itself. Here's the deal. The idea is that on this picture only pixels that are, were real. The question is, how do you make finer pixels from a coarser levels? So here's the coarser level, here's the finer level. And it turns out you do it two different ways. Okay? So we're going to start with the odd pixels. All right? So the odd pixels occur underneath pixels that are actually there. So there are a couple of ways of thinking about this. One is we're just going to assume that we're going to, I'm just going to tell you what the filter is right there. It's one-eighth, three-fourth, one-eighth. Okay. That seems a little bit arbitrary until you realize that's just kind of like if these were zeroes. Okay. And I took in all the values here, and I made it the same 5-tap a four, but now doubled it. So one-eighth, one-half, three-quarters, one, these will be in the same ratio of one, four, six, four, one, as before. Okay? But I'm only pulling it off the odd pixels. All right. And so these points go away and I'm only left with one-eighth, three-quarters, one-eighth. By the way, we're going to provide the original Burton Edelson paper that talks about reducing expanse. You'll be able to, to look at that a, a little bit more carefully. All right. That's to reproduced a pixel, it's actually underneath one that's there. Slightly trickier are

the even pixels. We have a pixel here that's underneath something that didn't actually exist. So if I still do the six, four, one. All right? One, four, six, four, one above me. Okay? Well, when I'm on one of these even pixels, there's nothing directly above me, and there's nothing to my ends, there's only these two here. That's this one, and this one. So looking five above me, only two of them are really there, they're equally distant from me, what coefficients am I going to use? I'm going to use one-half. So to produce this finer value, I take one-half of that pixel and one-half of the other and that actually sort of makes sense. If you think about it, if I'm blowing up a picture, so I'm going to go from say, seven pixels to, to 14. I have to get space between two pixels that were there. What value should I put in that space? Halfway between the ones that were there before. The one that's a little trickier to understand is if I'm blowing it up, how come I just don't leave this original pixel alone. And you can read in the paper, this is for the odd pixels, you can read in the paper the motivation in terms of frequency theory. In terms of how they do that. All you need to know is that you now have a reduce operator and an expand operator.

10 - Apples and Oranges

want to see something cool? Yes. Okay. Let me show you how come they got their paper on the cover, I think It was the ACM, I think it was the journal, the journal of ACM, but I forget. They had the cover picture of a journal and here's why. Here's a picture of an apple, okay? Beautiful picture of an apple. Here's a picture of an orange and I'm going to tell you it's not just any picture of an orange. You'll notice that this apple is in front of some pencils. This orange is in front of some other pencils. kind of same colored background, and you'll notice they're the same size, and that's because they've been scaled to be the same size. Using the Laplacian pyramid stuff, I can make this picture. Okay, now if Megan could see this picture, Megan would be going, wow, how did they do that? Wow, how did they do that? Okay, first of all, I hope you are too, it's pretty cool. It's half orange, half apple, and it blends, and you can't see any seams, all right? And the way they did that is, they built a Laplacian pyramid, so this is the coarser scales, this is the original Laplace scale. The coarser, coarser, and they just seam together, each one of the, the pyramids, by just a single pixel, but at the small size, not at the, not at the expanded size, so at the very small one, they took half of the left, the left half of the apple, the right half of the orange and they just blended the middle row. Then they expand that, okay? Take the Laplacian here, Laplacian there, combine that, add in with single row, you do that all the way down, and what you've done is you've blended the, the low frequencies over a larger range of the image, and the high frequencies over a smaller range and you make this magic apple orange thing, and that's how you get yourself on the cover of the transactions on ACM Transactions on graphics. Cool, right?

11 - Hierarchical LK

Now we're going to apply these pyramids to Lucas and Kanade. So here's another really, really ugly picture and what I'm going to do is I'm going to step you through the algorithm in a minute, but let me just show you sort of in, in drawing. Okay? So first is here's our image at t minus one, here's our image at t. We reduce and we reduce on both of these in order to get small versions of them. The first thing we do is this M stands for motion model, we run Lucas and Kanade between those two. Okay? And that gives us a flow field, okay? And what we do is we make and that tells you how every pixel moves, we double up and multiply it by two that flow field and then we apply that to the next level down and then there's this funny function called warp. We take that function, we warp the level one of t minus one and now that's almost a level one at time t. It's almost there. So, you know what we can do because it's almost there? We can run Lucas and Kanade again. All right? We come up with a new motion field, add it back to the motion field that we had before. Expand that, multiply by two, apply in warp and continue. And that's how the iterative Lucas and Kanade algorithm works. I know that that's hard to see from a picture. Here is the algorithm written out as steps. So we're going to compute to level K, right? So we build all of our Gaussian up to level K. We initialize the flow field as being 0 at sort of K plus 1, then what do we have to do? Well, you know, here we're doing it from level K to zero. So from the previous level, we have the

flow field from the previous level, which of course is half the size. So we have to upsample the flow field, that's just expand it, make it twice as big. All right? And that's now the right size to map from level K to level K of t minus 1 to t. The problem is if in the previous level something moved by 1.2 pixels, how far is it going to move now? 1.2 times 2, 2.4 pixels. So when I expand the flow field, which is makes it a bigger size. I also have to multiply it by 2. That's what the next step of the algorithm is just, is multiply uip, vip by 2 to get the predicted flow. And then you warp that level of the Gaussian of image two towards image one in order to get 2 prime or t minus 1 to t. Whether you call it one or two, it doesn't matter. And you just keep iterating and then we apply Lucas and Kanade again between the warped version. And that same level at of I1 to get this slight what's called the correction flow and we have to add that correction flow back to the expanded flow that we had before that's what these, these equations here are doing. They a, they are adding that slight difference that we now found back to the original predicted flow. Here are some results, so this is an example of that optic flow and here is that tree. And you can see if we don't do it hierarchically, we just apply Lucas-Kanade, it does a nice job on the background. But up on the tree itself, where there was a large amount of motion, it gives you exactly what you'd expect, which is junk. But when we run it with the pyramids, we get a better solution and so you get these nice, you know, flow fields here in the middle, also in the back. Now you'll notice we still have a problem at the boundaries, all right? And why is that? Well, remember at the boundaries, there are pixels appearing and disappearing and there's no way that optic flow knows what to do when a pixel appears out of nowhere. Basically, because you moved and now there are pixels that were invisible before are now visible. And so the the optic flow along the occlusion boundaries is still a problem and you need further processing to handle that.

12 - Sparse LK

All right, so that's hierarchical Lucas-Kanade. And as we described it, it gives you a dense flow feel, it gives you a u,v in every location. But remember we said that we might only want to do Lucas-Kanade at areas where the eigenvalues were well behaved, that is the places that were kind of corner like. So, there's a version of Lucas-Kanade called Sparse Lucas-Kanade which is just that. It's a hierarchical version of, that's applied only to what they call good locations or good features to track. This actually comes out of some work by Shi and Tomassian, good features to track
and applying Lucas–Kanade to it. And by the way, those of you who use openCV, the Lucas–Kanade used to be dense and then they just changed it. It's now actually sparse so now it only gives it to you at a select set of points. And they sort of changed it without telling you because the function still compiles, it just, your code just breaks a little. So so you should only be using the, the sparse version.

13 - Start with Something Similar to Lucas Kanade

All right, so look, what I'm showing you here was sort of a state of the art of motion estimation 20 some odd years ago for derivatives, and for our first class where you're starting to do this. That's fine, we're not assuming you have any background in computer vision. But if you were actually to take a look at work that's going on now, you will find motion estimation that takes this as the basis and then it adds a lot of other stuff to it. So they worry about, sort of, how the, the, the gradient behaves, the constancy of the gradient. They have energy-minimization terms. Remember we did energy minimization for stereo? And that came up to be a better solution. It not only had the local matches but it also had more global structure to it. It also does keypoint matching, which is saying I've got something that has moved a long distance. So remember, the, the short distance, the, the expansion works really well. We have to do things hierarchically to get the long distance. Or, you could just use point to point, right? I know this point went over here, let me fill in the stuff behind it, or, or around it. And, when you do that, you get sort of a much better result all together. And this is an image taken from a paper that came out in 2009

14 - End

That ends the lesson. I was about to say that ends the lesson on Luke's flow. That's pretty much true. It's only a little bit of a lie because we're going to see a little bit more of this derivative motion stuff in the, in the next lesson. But basically what we did is we solved for a different flow at every point by assuming that a window over that point had a particular flow everywhere. Then we move the window a little bit, and now we pretend that in this new window, the flow might be something different, even though those windows overlap, which is a little strange but, but that's how you do it. So it's okay to have small errors along the way like that. But the idea, so there's a little bit of a contradiction here, right? In fact that contradiction is in what I just said. We assume that a window is all moving the same way around a point, and we assign that value to that point, then we move the window a little bit and we throw away that last thing, and we say we're going to assign it to the next point even though those windows overlap. You might say, I want to carve the image up into sort of coherently moving patches. And so, there's two parts of that. There's this notion of how do I do the carving up. And what do you mean by coherently moving? And what that means is you've got a model for the motion inside these regions, and the next lesson will talk about motion models and how we could find regions that behave according to some motion model.

6B-L4 Motion models

1 - Intro

All right, welcome back to Computer Vision. What we're doing in this lesson is, we're basically going to finish up motion estimation, and this is the raw estimation of motion with respect to sort of a motion field. Going forward, we're going to do things like tracking, and other, and other sorts of examples where we'll sort of use motion in order to essentially process an entire video, but here only talking about frame to frame motion. In the previous couple lessons we talked about methods that gave us dense flow, except for that sparse l k thing. Where we had a different estimate for each pixel and we didn't have any real serious constraint between them other than something like, local smoothness or something like that. That is, we, we were just imposing this on the image without any sort of real thought to what's going within the picture. But suppose you know that the motion is actually constrained, okay? So for example, maybe you know, you're just looking at something rotating and you want to recover the motion all right, so that would co, constraint on the motion field. Or suppose you know that some object's variation in depth is small compared to the depth to the camera. Okay, so delta z is small compared to total z. That gives you another constraint on how things move. In these cases, you might want to have sort of a model for the flow, that is, how do I think the points could be moving within a region?

2 - Motion Models

So here's some examples of models and these are the same transformations that we talked about earlier when we're talking about doing mappings for homographies and panoramas, and things like that. So we have very simply so we basically go from the red to the, the green. So if we have a translation, right, so this thing just move that way. And in translation, how many unknowns are there? Two. Two, right. So since we get two equations for each point, we just need to track one point here, one point there. If I know that between them, I know how x changed, I know how y changed. Two equations, two unknowns. Great, everything is good. We have what's called a similarity. A transform similarity is where things can translate rotate and scale. All right, and that's four unknowns, two points. Which if they, think about it. If I take a stick here, I can translate it, rotate it and stretch it. I only need two points to know the translation rotation stretch. Much more interesting is what's referred to as affine. I have this rectangle and when I move it, I can translate, rotate, skew, all right? And so, parallel lines remain parallel. And I have six unknowns. And, and there's, we, we actually looked at the math before. One way to think about it is, I take one triangle,

and I map it to its corresponding triangle. And that's an affine transform. So the six unknowns, and I can solve it with three points. And then finally, I know you're all waiting for this, right? Here it's written as perspective, we can also write this as homography. Now you're seeing why we don't have me writing anything. We did this before right? We said if we have perspective projection of a plane to another plane that, that transformation is a four-point transformation, eight unknowns and that's because it takes four and four points therefore to recover homography. So each of these can be thought of as transformations or as motions within a region.

3 - Focus of Expansion Example

So, here's a, an interesting example. This is a great old slide. I think this slide is actually from The Psychology of Computer Vision book by Pat Winston, which I think was 1975, I don't know. It's just such a cool because they, it was before we could do any, it's just beautiful. Anyway, so, the idea is here is the image on the left and another image on the right. And the question is taken at time one and time two. How are things moving? Well if I just look at these two pictures, it's a little hard to tell. But if I give you the flow field, it's very clear what's going on. Right? What's going on is the camera is moving forward. Right? The camera is moving forward, which is inducing this flow field. All right? And that's the flow field of a plane. That's going to be a Homography. And there is another flow field here and you realize this thing called a Demon. And that's whats so cute because back in the old days in AI when you had little list programs running in the background and, and, little things would run on their own and have their own behavior. They were called demons, so this is a lisp joke. Anyway, so you can see this demon is actually just a planar cardboard thing that is moving sort of forward and to your right it's left. You can see that from the flow field, and if you cut up this image into this area. And then that area, you would have two different flow fields, both of which would be homographies. So, they would both be modelable flow. All right? So, I'm going to show you a couple different things. Some is more math, some is, less math. But, it's basically the mathematics behind modeled flow.

4 - Full Motion Model

You may remember from physics or somewhere else, that if I've got some point that's at a vector R from the origin, let's say I'm the origin or the origin is the middle of the object, doesn't matter. If that point is rotating about the origin with some angular rotational omega, and it's also translating with some translational velocity T, okay? The overall velocity of that point, I hope you remember this from physics, was V is equal to omega cross R plus T. It's just true, you can work it out. And, here's what that looks like written out. You remember we did that crossing operator could be written as a matrix multiplication. We did that for essential and fundamental matrices, so that's all this is. We've just written out the cross operator as a matrix where these are the components of the, of rotational velocity, and these are the components of the translational velocity. In fact, that's what it says here, so VX, V sub X, V sub Y, V sub Z, are the velocity vectors. In the world, in the world, in the XYZ direction, VTX, VTY, VTZ are the translational components in the world and omega X, omega Y, and omega Z are the angular velocity components in the world, all right?

5 - General Motion

In order to figure out how things are moving in the image, okay. This is, this is how they are moving in the world. In order to figure out how they are moving in the image, we have to go from world coordinates to image coordinates. Okay. Here's our simple equation for, in a font that is to small to read, but it will get bigger in a minute. It's okay. We're just going to use straight perspective projection. We're going to assume the origin is in the middle, we're going to assume nothing is skewed, etc. So the only thing is is that we need to know the focal length. So little x is the focal length times big X over big Z, same thing with y. Right? That's our usual perspective transformation. Okay. That's X and Y, but what about how X and Y are changing, given how how the points are moving? Given the V in the world? Well, it turns out that's pretty easy. We're going to

call little u and little v here. The velocity in x and the velocity in y. So these are just the derivatives in time, so this is how things are changed. So u is the velocity in x, dx, dt, y is v is dy, dt. All right? Well, so to take the derivative of x with respect to something. In this case, time. Right? How it's moving. Well, you remember, the, the formula for the derivative of a, of a quotient, right? Meg is saying, thank God. No, I don't have to know these things. Well, it's just the derivative of the top times the bottom minus the derivative of the bottom times the top all divided by the bottom squared. Right? Right. Okay. That's what this is, right here, okay? Vx, that's the, that's the change, that's the velocity in x times Z minus the top, X times the velocity in Z divided by Z squared multiplied by f, that's just the constant. Same thing down below, okay? Then we start to expand this stuff and we get this nice little formula here, okay? So when you look at it, that's still kind of ugly, because you don't actually know what Vx, Vz and Vy, you, you know that they're from back here. Well, it turns out, you can write them pretty beautifully for some twisted definition of beauty in this very simple formula here. 1 over Z at some point x, y times A times T plus B x, y of omega and let me show you exactly what that is. Okay? So here, I've rewritten it. So this is the motion, so u is how much it's moving at some point x, y, so we know the point x, y in the image. The question is how much motion are we seeing? So we've got the u and the v, that's the motion in x and the motion in y. All right? And what it does is this formula relates the translation and rotational component of the point out there in the world. These are world T and world omega. Two, how it's moving in the image? Okay? Thing to realize is that this quantity A is just a function of things we know. We know, we're assuming we know the focal length of the lens and if we're at some point x, y, we know what that point, this is little x, little y. This is the location in the image. Cap X, cap Y, cap Z, out in the world. Little x, little y in the image. All right? So you'll notice that A is just this function. B is a little more complicated, but it is still just a set of things we know. Okay? So, if you tell me some point and you tell me the focal point, I can tell you this A matrix and this B matrix. I might not know T and omega, I might want to solve for T and omega, okay? But A and B I know everything of.

6 - General Motion Model

So there's only one really important and interesting thing about this equation and here it is. Notice the Z value of the point. So this is written in a funny way, right? This is Z of x, y. What does that mean? That means, if I'm at some point x, y, remember that's a ray in space. Right? Z in the, in accordance to my camera is the depth of that point. You'll notice that this thing is made up of two terms, the A term and the B term. The A term has the translation in it. The B term has the rotation in it. The depth is only in the A term, the translation term. Why? Think about motion that happens. If you remember, a projective camera just has rays out in space. And you can think of the rays as going out till they hit something and they stop, okay? Now, if I rotate those rays. In fact, maybe I can draw this. So here's my point and here's a whole bunch of rays. So let's suppose here's my image plane for frame one, okay? If I rotate some angle, so my image plane will now be there. So before this ray was hitting at that point on the plane and now it's hitting at this point on the plane. So, if before it was in the middle, now it's to the right-hand side. Okay, fine. Notice I didn't tell you how far the rays went, because as the image plane rotates and it cuts through the rays, it doesn't matter how far the rays go. So the depth has no impact on the amount of motion that you see as you rotate, but if I translate the depth matters a lot. Remember, the whole thing about the moon? Right? That the moon seems to follow you? And we're going to demonstrate this right now. So now get out your rulers on the screen, right? I'm going to take a step to the right, like that. Okay? So the amount of motion that you saw is whatever it is, all right? Now, I take the same size step. Megan, did I move in more of the field? Did I move much further across the field than I did when I was in the back? Megan's going yes. In other words, the amount of motion you get, this is called motion parallax. It is a function of the depth. So that really dorky demonstration is why the Z value is only in the term that has the translation component. If you don't translate, the depth doesn't matter in terms of how much things move. That's why we talked about when you build panoramas from rotation, I can just map the planes. I can map the, the rays, because it didn't matter what the depth was. This is the general motion model. I, I hope this was not too pedantic. I like tying back the stuff we're doing now with the, the stuff we did before.

7 - Motion of a Plane

Suppose I know something about the motion, the, the geometry of the points. For example, let's suppose that I'm on some plane. All my points are on a plane. Here's my equation of a plane. So, all my points, X, Y, and Z sa satisfy this. Remember, we showed this before when were mapping everything on a plane and saying how it would've moved. We ended up with a 3 by 3 matrix, but it was a homogeneous system, so we could scale it any way we wanted. So if we wanted to, we could set one of those elements to a 1, and then we only had to solve for the other eight. There were eight unknowns, so therefore we needed how many points, Megan? How many points? How many point? Four points. Here is the, the equation, using just the X Y values, and you'll see, a1, 2, 3, 4, 5, 6, 7, 8, eight point, eight unknowns, four points needed, so when it's on a plane. Now, if it, plane is very, very far away, what does that mean? Well, what it means is if the difference in the distance on the plane is small, compared to the distance from the plane to the camera. You can show that this is the simplified model, right? It's just six unknowns, otherwise know as an affine transform. This is one of the reasons that affine transforms are used so much. When things are on a plane, on a wall, on the floor, even on maybe, front of a body, for awhile. Mine used to be more planar than it is now. For small amounts of motion, you can think of this being as a plane that's relatively far away. And what that means is that this, that that plane will undergo an affine transformation.

8 - Affine Motion

So, if we have these motion models, we'd like to be able to use them within all of that motion math we did before on the brightness constraint equation and the optical flow and that kind of stuff and we can do that directly. All right. So here we have our image showing you the math of an affine transformation. Now, you'll all remember your brightness constancy constraint equation, yes? Yes? Yes? Yes? Looks like That. Remember, what this says was, that the dot product between the u and v, that's the amount of motion and the gradient, has to essentially remain constant with respect to the change in intensity. So that was the brightness constraint equation. Right? We had two unknowns, one equation. So the way we fixed that before is, we assumed that all the points had the same uv. Suppose instead of saying it's the same u v, we assume that it's the same affine model. That means that we substitute in for u and v the equations from above. Doesn't that work out nicely? All right, so instead of just saying u, we're going to say okay, over this window, we have to find the a1, a2, a3, a4, a5, a6. So how many unknowns is that? One, two, three, four, five, six. We have six unknowns over the window. So how many points are we, how many equations do we need? Well, we need at least six, but now our assumption has been relaxed. Before we required that everything be moving constantly, so you can't really let your windows get too big if you're going to require that they be constant motion because as you let your windows get bigger, they may not be moving constantly. But if you can allow for an affine deformation, so they're not all moving the same, they're all just the same flow, like maybe they're rotating around, now I can have bigger windows. So what we do is we take this equation, all right. Each pixel provides one linear equation with six unknown, so we're going to do that same least squares minimization we did for Lukas-Kanade, but now, instead of having a constant uv, those two unknowns over window, we're going to use this linear model with six unknowns. And we do exactly the same thing we did before. It's going to be a mean, minimized squared error, least squares, and that allows you to solve for that over that region and because the pixels are moving differently depending upon the xy, I can have a whole area that's rotating. It's one model, so all of those pixels use the same a1, 2, 3, 4, 5, 6 because it's the same rotation. All right. It's not the same little vector, it's the same model. So it can be a much more robust way of doing things. So here is a version of the Lukas-Kanade now, and it's taken from the original paper. It finds that, it's a paper that's cited a lot and it's almost impossible to find by Jim Bergen and Anandan and Keith Hanna way back in 1992, and the thing that's different about this from Lucas-Kanade is, instead of computing m for motion, which was just the specific motion everywhere, we're doing these little m, which are the model parameters, right? And so you have to compute the model parameters.

9 - Layered Motion

What if, over here I've got something spinning and over here, I've got something translating. No, sorry, expanding. I'm expanding. What I've got is I've got different motion regions and I need to find those different motion regions. So, some nice work done, again, a long time ago by, Ted and John Wang, I'm going to call layered representations. The idea is, what if we could break the image up into a bunch of different motion regions? Okay. And in fact that is illustrated here. So this is a, a figure taken directly from their paper, so the idea is we have a hand that's moving across, and it's against the background of checkers, squares moving this way, because we're always moving our hands over, moving checkerboards. I don't know, it's just what they did. Okay? So, here they're showing you each of the frames, and clearly, what you'd like to do is you'd like to be able to recover those different elements that the front part is moving this way, and the back part is moving that way. So, here's how we do it. Here's our affine equations, again. So we have these local flow estimates, right? So, compute some local flow some way. Maybe you use straight Lucas-Kanade, all right? Notice that this one equation. Let's just worry about the u, for, for right now. That's a linear equation in x and y. That's just sort of a plane. In fact, if we were just doing this in 1D, it would be a line, right, forget the y, just say that u is equal to a1 plus a2x. It would just be a, a, a, a line. These are linear systems. All right. So, the way this works is the following. And here's an example done from 1D, again taken from that paper. Suppose this is the velocity as a function of x, so we are only showing you the 1D version, cause its a lot easier. And lets suppose that this is the true flow, and what we mean by the true flow here is that the foreground region is actually moving in some way and its not a constant because its an affine flow, that is it's a, it's planar. And the background might be moving some other way. And what we want to do is, we want to be able to find that. So the first thing you do is you do a local flow estimate. Lucas-Kanade everywhere, let's say. That gives you some vectors. Now those vectors are not perfect. They are what they are, okay? Well, what you do then is, you take a look at those values. Where do I see big jumps? I see a jump there, I see a jump there, I see a jump here, I see a jump there. And I break that up into those segments, and then what I do is I take a look at the segments, and I say, which segments seem to be along the same plane? And it's planar because of this affine equation. And when you do that, you get to automatically separate out the foreground motion from the background motion just by fitting it says line fitting but essentially by fitting the affine plane to what's going there.

10 - Example Result

To actually make this thing work requires a lot of detail. Finding those breaks is not trivial. Clustering things is not trivial. Fitting the lines, well that's pretty trivial once you've found the clusters, but you've got to find the right clusters that belong to the right lines. There's a lot of detail, and you can take a look at the, at the paper that we mentioned. I just want to show you the results here. So here is again the flowerbed sequence from mpeg. And here what you can see is they've put down what the different motions fields they captured were. One was the, the flowerbed. And you can see that flowerbed is kind of like a plane that's moving in a particular way. So that's one motion. The house with the trees that are sort of right by the house is another motion, and the tree in the foreground is another motion yet. So it has done a motion segmentation of the sequence. So at the end of the day, it doesn't tell you that all the points are moving arbitrarily, like Lukas-Kanade does, or even smooth Lukas-Kanade. It doesn't tell you that they're all moving the same way. What it does is, it gives you this pa, this broken-up pattern, or this broken-up image sequence of saying, this region is moving this way, this region's moving another way, and then this background region, that's occluded by these regions, is moving, yet, a, a third way. And this is motion segmentation. There's a lot of work lately on video segmentation it's an active area of research. And there's always this, this question of, you want to break it up into enough pieces that you get out the different elements, but not so many pieces that you take something that really should have been kept together, and you break it into two parts.

11 - End

Like I said, that's the end of the work we're going to do on sort of direct motion. So basically we've looked at two generally different ways of doing this, of doing motion estimation between two frames. One can, that can handle sort of large range sparse, we find particular features and we come up with descriptors. Then we find them in the, second image. We think we know which matches are right. We guess to see which pairings are correct. Look for a motion model, and maps them across. And that works well. We can have lots of differences between the points, but we have to have a consistent model across them. And then we do that ransack thing, etc. The other method are much more direct or optic flow or spatial temporal methods where pixels may be deforming in a variety of ways. We've got some patch going one way, another patch doing another, maybe a patch is stretching, twisting, whatever. And we're going to use the idea that we can take spatial and temporal derivatives in order to tell how things are moving. You know, one way of thinking about is that the first method is good for movement of tens of pixels, so things move a lot and they may have changed. So I take a picture over here and take a picture over there of the same thing, and I want to figure out which one is which. Okay? The motion ones are better when things are moving a small amount, you know, less than ten p, a small amount of motion, but maybe lots of frames. So like in a movie when you see nice, smooth motion between something, that's when you would think about the direct optical flow of methods. So now you have, sort of in your tool kit, a variety of ways of computing motion between frames.

7A-L1 Introduction to tracking

1 - Intro

All right, munchkins. This is the wizard welcome you back to introduction to computer vision. As you can tell, I'm, sort of, energized and I've taken some time off, about a year or two or so. What we're going to do this time is start working on tracking. So the last unit or at least some previous one, hopefully, that depending upon the order in which you're viewing these. We were looking at motion which was sort of the change in imagery between two successive frames. And that change could be due to a change in viewpoint multiple cameras or a change in time, you're watching something move over awhile. This unit is going to focus on tracking, where objects are moving over time and the computations that we're doing are to keep track of them. And it is sort of a fundamental part of computer vision these days. Although, a lot of what we're going to be talking about will eventually get into things that are come from the domain of robotics. Because there has just been a lot of work done on sort of how you keep track of things overtime, we'll do a lot of modelling. Also, I want to say again, a bunch of slides adapted from Kristen Grauman, Deva Ramanan and Svetlana Laze, Lazebnik. And I, Svetlana, if I messed up your name, I'm, I'm sorry.

2 - Tracking Examples

Let me just show you some examples of what we mean by tracking. And here is some slightly older ones. The one on the left here is this girl jumping around. You see the little red ring around her head. That's, I think that's Andrew Blake's daughter a long time ago. I think she's now a lawyer. So that tells you something about the time difference. Andrew, I have no idea if that's true. And also from a similar work is this leaf blowing in the wind. And you'll notice, as the camera zooms in and out and things move around, it does a pretty good job of sticking to the leaf. Both of these examples were work that Andrew and Michael Isard did in the particular type of tracking that we'll actually do not this lecture, but a couple lessons from now. Here is a slightly more recent one. This is a tracking of body parts as somebody's moving. And you can see the, the reference there. And again, this is a tracking problem, but it's tracking with a model. Right? So the idea is I want to be able to track where the parts have moved. Here you see a sort of a more traditional notion of tracking. But in this case, it's lots of targets, right? So there are a lot of people say walking around in the street

here. And you'll notice that it does a really good job of keeping track of all the people. The little box that's around each one of them and there's another little box that's on their head. Here there's just a little bit of animation showing you that it's tracking them through spacetime, right? So this volume with these trajectories through there are showing you how it's doing the reasoning over spacetime. There are a couple of interesting things to notice here. Watch this guy going behind the pole. All right, so he goes behind the pole and he appears the other side. And you might ask, how does it do that? How does it do that? I'm not going to tell you yet. I'll tell you soon. And also here, it's showing you just the heads that have been tracked to show you that it's recovered that. So this is an example of sort of a state of the art tracking at least of a, of a couple of years ago.

3 - Tracking Challenges

As we mentioned so far we've only done flow estimations and pairs of images. Now one way of thinking about doing tracking is if you had more than two images you could do some form of optical flow or, or some sort motion between one image and the next and then you could just do it between the second image and the third, and the third image and the fourth. Basically you would just sort of follow the arrows, connect the dots. So there are a couple of challenges to just doing that. One as we saw, it's hard to compute optical flow everywhere. You have to have just kind of the right types of behavior for the gradient, in terms of being able to track these features. Second, there might be large displacement so between one frame and the next something may have moved a lot and so there's a lot of search that has to be done. And so one way of doing that is you might want to take the dynamics into account. How is this thing moving? Make a prediction. And in fact that's what we're going to spend a lot of time talking about over the next couple of lessons. The other problem with sort of connecting the dots is things will drift. That is, you know, I make a little mistake, and then a little more mistake, a little, and eventually, I'm no longer tracking the thing that I had started wanting to keep track of. And then finally, another problem is what are called occlusions or disocclusions, right? Occlusion is when something gets occluded. Disocclusion, maybe unocclusion's a better word, basically it appears from something. So something appears, was it something that had I lost before or is it something new and how do I do that kind of tracking?

4 - Shi Tomasi Feature Tracker

We'll start with one very simple description that you've actually already seen. Right? We talked about the Shi-Tomasi feature tracker. And basically it's this idea of only compute motion where you should. We talked about this when we talked about the Harris corners and we talked about how to analyze that second moment matrix. And we said well there are other ways of analyzing that second moment matrix including the the Eigenvectors and Eigenvalues there. And we said that there was this Shi-Tomasi thing called good features to track the thing that OpenCV made use of. And basically, the idea was that just track points that you can track reliably. Their basic algorithm was the following. Basically from frame to frame you're going to do Lucas-Kanade, pure little translational motion. Right. Remember the, the idea was that we assumed that within a given small patch we're going to find some translation. All right. And what's nice about that is that's pretty robust for sort of a small neighborhoods, you know, you can basically say that they're all translating. Then what you do, and we're not going to go into this too much here is we're going to take those points that have moved, and look for an affine consistency. So the idea is for each of the, the little points that you've tracked using the, the translation, is there an affine model that maps the, the tracked points to the original points? And then you can compare those back to the original. And that's exhibited here. So here's an initial image and you can see this little sign in there it says 25 miles an hour, and here they're getting closer and here you're zooming past. So the, the, the frame, the sign is both translating. Right? Its moving to the left in the image and its also expanding. All right. And here you can see the frames that had been recovered. Or I should say you can see the region that has been recovered through the tracking. And the idea is that all the points in the sign, remember it's on a plane at moderate distance, which means it's a perfect affine transform. All those points are moving in a particular affine transform as you move across the, the image sequence. And,

like I said, that was this, good features to track, which later got incorporated into OpenCV for, for doing, estimation of motion.

5 - Tracking with Dynamics

So we mentioned that you could use Lucas and Kanade between frames and, and use the, the, the Shi-Tomasi way of, of finding features small displacements. But what if there are large displacements, what if things have moved a lot because they're moving quickly? Well. One thing we do is we do know that we can build a descriptor. Remember the sift descriptor. So that if you found a matching point I'd be able to say yeah I'm pretty sure that's a match. But the question was how would I decide where to find it? So when we were doing the ransack method of finding those panoramas you remember we essentially com, compared all of these with all of those. Found the putative matches, selected a bunch, and then tried to find something that gave us a consistent model. So you're doing a lot of work in order to try to globally find all of these kinds of things. For tracking, what we want is, well, first of all we typically want real time methods. Because we want this thing to operate as things are moving along. And in particular, we, we want this notion of a coherent tract result, right? So we're not going to try to match everything in one frame to everything in another. And then do it again, and then do it again. And do it again, and do it. See? So, you just saw my head rotating around. I hope you're paying attention. So they idea is that you have this smooth, coherent trajectory. And that's what we want to do for tracking. And in order to do that. We have to worry about the dynamics. That is, how are things moving, how are they changing? So, when we do tracking with dynamics the key idea is we've got some sort of model of the motion. Maybe we're updating that model, like the velocity and acceleration, but using that model we want to predict where the objects will be in the next frame and then we're going to look around that area, okay? And that prediction will allow us to restrict where we're doing the search, and the other thing is if we have a noisy detection, that is our ability to knowing where this thing is not perfect, we should get a better overall estimate, because we're combining a prediction with the measurement, so that's the idea of tracking with dynamics.

6 - Detection vs Tracking

Fundamentally, the use of dynamics, is what's different between just doing detection, and actually doing tracking. And that's shown as an example here. Here we have 20 frames taken. And here's some dude right there. You can see that he's walking across all these. And we just like to keep track of him. And of course, he's changing his appearance every frame. He's not going to look identical from one frame to the next. So, detection, we independently detect the person each time. Right? So, we maybe have a little model of what this person looks like, and we run it maybe all over the image, and we say uh-huh, you know, he's here. Uh-huh, he's here. Uh-huh, he's there. And, just these there. No more uh-huh's, cause it's like not exciting anymore. Okay, so that's detection. Tracking, we do a little different. What we do is we predict the new location of the object based upon the estimated dynamics. And I say estimated dynamic, because for example, this guy, he's walking, so maybe we est, maybe he's got a constant velocity, but we don't know what that velocity is a priority, right? We have to estimate it, right? So here we have an example where we detect him here. And it says, you know, we'll estimate the velocity, but maybe we estimated zero velocity, or we have some prior reason to believe he's moving to the left, because this is a sidewalk where people move to the left often. Fine. That allows us to make a prediction in the next image of where that person will be and then we detect them. So now we can do two things. First of all, we can improve our estimate of where he is based upon both our detection and our prediction. And second, now we do have a reasonable estimate of the velocity. So we can use that velocity to predict in the third fame, detect etc., etc., and we iterate. So this is what the, the basic tracking loop is.

7 - Tracking Assumptions

As we said, the key idea is we're going to use this model of motion to make these predictions. And there are two goals, right? Goal number one is that we can do less work because of this. Because we have some idea of where the object is going to be we can restrict our search. Goal number two is we should be able to actually improve our estimates because we're using both predictions based upon dynamics and our measurements. Now, to do this, we're going to make certain assumptions. Okay? And the assumptions basically come down to the notion that this is some form of continuous motion. We're going to assume that objects don't disappear and reappear instantly somewhere else. For those of you child psychology, that's a whole Piaget thing. Kids don't come to that belief until I don't know, it's like year and a half or something like that. So like kids who are, are really little, if you take a toy truck and you've got a screen here, and you put the toy truck behind the screen, and then you lift up the screen and the truck's not there anymore, they're not surprised. Things disappear in their universe all the time, or it becomes a, a carrot. They're not surprised either. Then all of the sudden they get to a certain age where the truck comes, if you pick it up and it's a carrot, they're like whoa. All right? And that requires a certain change in the mental model. Okay I don't know why I mentioned that. It's just really cool. It's true by the way, not that Piaget is all true but that's true about children. Anyway so we're going to make our assumption in our tracking that, that things don't just appear and disappear. We're also making an assumption that the camera doesn't move instantaneously. You know, because that would be the same thing. All right, if I instantly went from here to here, right, big change, things will have violated my apparent continuous motion. So that's basically that there's this gradual change imposed between the camera and the scene and the object.

8 - End

So that ends the introduction to tracking. Basically saying that what we're going to do is use a model to make predictions. Take some measurements, and, integrate it. Like I said, it's not strictly computer vision, because it involves state estimation, which comes from all sorts of parts of engineering and robotics and things like that. But it's what's required in order to make the tracking to work. So well, we'll sort of take a little bit of a detour to talk about state dynamics and estimation. Little bit of robot localization using visual features and then we'll get back to fundamentals of tracking using these methods. Cool.

7B-L1 Tracking as inference

1 - Intro

Alright. Welcome back to computer vision. Today we're going to continue. Well actually we just introduced tracking and now we're actually going to do some. So, remember last time we talked about the difference between detecting and tracking. Detecting, we have some operator and we just find or whatever our target is in every frame. In tracking, we're going to predict a new location of an object. Or the, the expected distribution location of an object using estimated dynamics, so somehow we're estimating the velocity, accelerations, whatever. And then we're going to update based upon the measurements that we take. So that's sort of the shall we say the intuitive way of describing it. What we're going to do this lesson, the next lesson, the lesson after that, the le, I don't know. Probably for the next eight or seven, 12 years or so. What we're going to do is, we're going to make this, much more formal. And basically, tracking as a form of inference. And what we mean by that is, we're going to have some state of the objects. Some state we're trying to estimate. We're going to call that x. That's the parameters we care about. But we don't actually know that. So we refer to that as being hidden. And we have a sensor. Typically, it's going to be a camera for the stuff we're talking about. And what it's doing, it's taking some measurements. It's, it's measuring, like, our raw detections that we're doing before. And we're going to refer to those as noisy observations of whatever the underlying state actually was. The system that we're assuming is that every time

step t, the state, the underlying state changes from x of t minus 1 to x of t. So here we have x of t minus 1 to x of t. And then at time t, we get a new observation. We get a new measurement. And our goal, your mission, should you choose to accept it, is to recover the most likely state, time t. Now I was thinking about that, its actually, that's not quite right. What you want to do is you really want to estimate not the most likely state. I'm going to cross that out and say the distribution or density. So, if I have the density, that is I, I have some belief about where x might be, I can tell you the most likely place that would be the highest probability. I can tell you the average guess. That would be the expected value but the idea is that we're going to have a density. And, so your goal is to recover or estimate this distribution given first of all everything you've seen so far. Right? So you've been taking measurements all along since last Tuesday. Where is this thing now? And the other thing that was given to you was some knowledge or some model of the state dynamics, that is how this thing moves. Then you may have to estimate some of the parameters but you may know that you know, I'm, I'm going to assume that this thing has a particular velocity and the acceleration is just some noise. That would be a, a model that you're going to assume is known.

2 - State vs Measurement Quiz

So what do we exactly mean by state and measurement? Let's try to understand with an example. Say you're looking at a pendulum swinging back and forth and assume that it has a marker on it that you can detect reliably, using one of the corner finding or template based matching techniques you've learned so far. Now we can identify different quantities to describe the system. For instance, the center of the target can be located using an x and y coordinate. Similarly, the pendulum rod has a certain length. Let's call this l. The circular weight at the end has a radius r. We could even say that the pendulum makes an angle with the vertical line. Let's call this theta. When defining a model for this system, which of these quantities would you include in its state, and which quantities are directly observable from images? That is part of the measurement model. Again, the quantities we are looking at are, the x, y location of the target, the length of the pendulum l, the radius of the circular weight, r, and the angle made with the vertical theta. Mark the appropriate check boxes.

3 - State vs Measurement Solution

If you think about it, the only thing that we can directly observe is the x, y coordinate of the target. Yes, we could try to detect the pendulum rod or other components, but that may not be as reliable. So, this is the only thing that needs to be in our measurement model. The same x, y location can serve as our system state. However, since the motion of the pendulum is constrained by the fixed length rod, assuming it is fixed length, it's better to compress the state. It really only has one state variable. That is theta. Again, this is assuming that l and r are constants. When analyzing a dynamic system, it is very important to identify what are its real degrees of freedom. Sometimes, the state variables you need to keep track of may not be immediately apparent. We will see examples of such systems later on.

4 - Tracking as Inference

So to give you, again, still a little bit of intuition, but getting a little more formal, so here we have the same person walking at Time t, and then there's a question of what do we do at Time t plus 1. The idea is that from Time t, we have some prediction or some belief that's indicated here in red of where we think the person is. Okay, and that's our prior. So, before we take any measurements at Time t plus 1, this is our belief. By the way, I will go back and forth, the slides sometimes go back and forth, between whether we're talking about moving between t minus 1 and t, or t and t plus 1. It doesn't matter. It's just going from one step to the next and sometimes I screw up one way, sometimes I screw up another way. This is like the story of my life. All right, so at red, we have the idea is we have some prediction to Time t and we want to know what's happening at t plus 1. At Time t plus 1, we take a measurement. That's what's shown here in green. Okay, so we had some distribution that was our belief about what's there and now we have some measurement. Now

notice, that this measurement is also a distribution. We're going to talk in a little bit about what that means it means two different things. One, you can think of is that, given that the person is someplace, how likely would be the current measurement that I got? So you see here, let's assume the measurement was there, okay? This distribution says, not how likely is the person at a particular place, that's what I'm trying to estimate. What this distribution says is, how likely would this measurement be, how likely would I be to get this measurement if the person was at these different locations? That's a little confusing. When we do the math of likelihood, that'll become clearer. For now, you can think of a measurement that's just a little bit noisy, I don't know, you know, it gives me some bounds on where the person probably is. And then finally taking that prior be, belief or prediction and the measurement, I come with what's referred to as either a corrected prediction final estimate. Or the technical term is the posterior distribution is to where I think it is, and that's what's shown in blue. So I go from a prior, or an old belief, to a measurement, to a new belief

5 - Steps of Tracking

Writing these out as equations so the prediction can be thought of as this way. What is the next predicted state of the object here, here and back to x of t given t minus 1, given all the previous measurements? Okay? So you see here that it says given y from 0 to t minus 1 so those are all the measurements up until before t, make a prediction at time t. Correction is again I want a distribution, remember I'm coming up with a probability distribution of x t, given what's new. Well what's new is I have my new measurement, okay? So basically, this was my prediction based upon my measurements up until t. Then when I'm done, I should have a prediction including the current measurement. So basically, the tracking is this propagating what's called as I said the posterior distribution, posterior density of state, given the measurement. So this term here, this is my posterior because it's after I've taken the measurement at time t.

6 - Why Probability Quiz

Wait a second, why do we need probabilities? I guess it makes sense for prediction, because we don't what the value of x is at time t since we only have information until time t minus 1. But what about correction? We know what the measurement is at time t. Why can we not get a definitive value for Xt? Why is this a probability distribution? Is it because the state end measurement are different quantities? Like in our pendulum example, where we said the state could be just the angle theta and the observation was the location of the target. Or could it be that observation values are unreliable due to underlying sensor noise or even imperfect detection algorithms? What if observations are missing or beyond known limits? Is that a good reason to model state as a probability distribution?

7 - Why Probability Solution

Let's think about the first point. Our pendulum example is one where x and y are different quantities. However, this is not a strong argument. Here our state can be precisely computed using the observation. It is merely a more efficient representation. However, there are other cases where the difference between state and measurement is more important. You're looking at a clown juggling three identical balls that you want to track. You have an excellent detector that, given any video frame, spits out three x, y coordinates that are the centers of the three balls. This is your raw observation. x1, y1, x2, y2, and x3, y3. There is no guarantee that x1, y1 refers to the same ball across frames. Clearly, you need something more in your state model. How about velocity? Including direction and magnitude. That would help you predict better which x, y belonged to which ball in each frame. There are many other situations like this where directly observable quantities do not suffice as the system state. Therefore, the first point is valid. More importantly, you need to realize that detection techniques, especially dealing with real world data, are not always 100% accurate. Noise can be introduced into observations, due to various reasons. Sometimes just the discrete nature of digital images results in observations that are not smooth. It is best to take

observations with a grain of salt. Remember the video where that guy walked behind the pillar? That's a missing observation. If you didn't have anything more in your state, where would you say the person was? Well you could say that is highly likely that the person is nearby where they were last seen, therefore the last point is also a valid reason for modeling these as probability distributions.

8 - Simplifying Assumptions

To do this we're going to make some simplifying assumptions. The first assumption is a Markovian assumption and that's that only the immediate past matters. So what this says is, if I know all of the states up through t minus 1, and I want a prediction of this state at time t. That just depends on the state of t minus 1. Now these are Xs, not the Ys. These are if I actually knew the state. What this tells me is if I wanted to predict the state on Wednesday, and I knew the state on Tuesday, Monday, Sunday, Saturday. It's all the same as knowing the state on Tuesday. Okay? I only have to know one prior state. Now some of you might ask, but, wait a minute, if I know Monday and Tuesday, I might be able to estimate the velocity for Wednesday. If I know Monday and Tuesday, I might be able to estimate the velocity on Wednesday. So that's right, except not, because. Your state could also include the velocity, right? So, if I knew the position and the velocity on Tuesday I make a prediction about Wednesday, so this is the Markovian assumption. This probability of Xt given Xt minus 1, that's what we refer to as the dynamics model, that's how things are changing. Second simplifying assumption. We're going to assume that the measurements, the probability of the measurements, remember, these are noisy measurements. Given everything in the past, and the current state, only depends on the current state. So this is where we say that the probability of a measurement, given a particular state and everything else, only depends upon the state. By the way this is called the observation model and I will tell you, it is this particular assumption that is sometimes the most suspect. One way you can think about this, suppose you have some noise in your sensor that is related to some other processes going on. And it changes in a particular way that you might be able to estimate. This thing says, well, I don't look at my previous measurements in order to make that prediction. So for everything we're going to do, we're going to make use of this observation model, where the likelihood of the measurement depends only on the current state. Just know that for more advanced things like conditional random fields and other, other approaches. We'll actually be able to make the current measurement tied to previous measurements, but we're not going to do that here. I know some of you have taken various machine learning courses, and you're familiar with graphical models. And, if I were to draw up this set of simplifying assumptions as a graphical model, it would look like this. And what this says is that some state X, depends only on the state at time T, depends only on the state at time T minus 1. And furthermore, it says that my measurement depends only on the state. And whenever I look at that, I always think to myself, hm. Okay, that's a joke. Why? because those of you who know what this is, this is a hidden mark up model. An HMM. See, now Megan, who's helping with the slides, she didn't know that, so she didn't know how funny that was. So she's now what is it? ROTL? No, ROTFL, sorry, I, I left out the floor. It's really hard to say emoticon things. Not emoti, what do you call those thing, whatever they are. She's, now she's getting up.

9 - Tracking as Induction

Another way of thinking of this is that tracking is just an induction, a process, right? An induction, like for math is if I know how to do n then I know how to do n + 1. When you show what the sum of a geometric series is or things like that, you tend to prove these things by induction or something like that. So for induction, you have to have a starting case, a base case so you have some guess as to how things start. Maybe for our tracking, we have some distribution. It could be a very uncertain distribution. And then, with that first frame, we take a measurement, y zero and then we would correct our estimate. And then we just keep iterating through this. Right? So, At every time t, we make a prediction for time t plus 1, take a measurement, and we correct for time t plus 1. And

somebody spent a lot of time designing these beautiful arrows, so I wanted to make sure you saw them too. And if you go take a look all over the internet, you'll find these arrows.

10 - Prediction

So, now, let's look at these steps one at a time. Let's first look at the prediction step. Okay? So our prediction step is, given some belief about X of, at t minus one, and given the measurements all the way up through t minus one, make a prediction of Xt. State of time t, still given only the measurements through time t minus one. The way we do that and I'm, It's going to be written out here in, in sort of a standard way. And then we're going to make specific use of that both, especially in particle filtering which we'll do in a little bit. What, the way you would do that is what's known as, you use the law of total probability and marginalization, allright? So the law of total probability marginalization, I'm only supposed to be reminding you here, you're supposed to remember all your probability theory, right? Then if I've got a joint density, right, P of a and b. If I integrate that over all the b's what I'm left with is the density over a. Allright. That's marginalization. So what I'm going to do here is the idea is well if I had a joint density of xt minus one,index t and I were to integrate over all the xt minus ones, I would just have the density over xt, so that's the marginalization. And, the reason we do that is we can then break that integral up into the following, right? So, you might remember the conditioning, right, probability of A and B is the probability of A given B times the probability of B, it's what it says down here, so I've just broken up this integral down into this. Where here's the probability of xt minus one given everything up to t minus one. And I multiple that times the probability of xt given xt minus one and everything else. So I'm just applying the conditioning property to that interval. And then I can use that independence assumption that we said before. Remember, the independence assumption Is that if I want the probability of xt, and I'm given the previous state, and a bunch of other things, those other things don't matter. The only thing that matters is that previous state. That's our dynamics model. So, basically, we have two things here. We have our belief. About t minus onr, giving everything up through t minus one and then we have our dynamics. Those are the two components that it takes in order to make that guess going forward.

11 - Correction

So now we know how to do predicted value, right. So the p of x t given all our measurements. So given this, and then we take a measurement at time t, what do we want. We want the new corrected value, x of t, given y's through y t so we're folding in the new measurement, all right. The way we're going to do that is we're going to make use of Bayes Rule, and I'm just going to, I have written down here for those of you who, don't remember Bayes Rule and don't have it tattooed on the, inside of your right thigh. But here basically is what we're doing, right? So Bayes rule says that if I want to know the probability of A given B, I can flip it around by knowing the likelihood of B given A. And then there are these other terms in here, okay? And you'll notice what we did here is we switched this around. This is now the probability of getting the measurement given all these previous stuff, times the probability, what is this? Well, that's just our prediction, okay, that's, that's P of A. And then this, this is going to turn out just to be a normalization factor. By using our independence assumption, right. The independence assumption says that our measurement y t, given x t and all this stuff, is only a function of, x t. All right? And the reason that I refer to this bottom part as normalization is, remember this is a probability over here. And basically, they all have to sum to one. And we'll get to that later. And this, this bottom value just normalizes everything so that it sums to one. And if you really wanted to calculate that normalization, okay, what you have to do is, you have to compute all the different possible probability of getting that y t over all of the x t predictions over the integral over all the x t. But it, we are never ever going to be sort of computing that bottom part. That's just there so you can see what the sort of the total integral is. And in fact now it says really a normalization. So it's really, really a normalization.

12 - Summary

So, summarizing, we have two equations: a prediction equation, and a correction equation. Our prediction basically takes two components. The, the estimate of our previous step. That is, our belief about X after time t minus 1 was done. And then, we have a dynamics model. RIght? And the way of thinking about this is, the likelihood, the probability that X could be at some specific point. Well, it's the sum or the integral of getting to that point from any point that it might have been before. Integrated over the likelihood that you were at those points before. This is a lot easier to see when we do the particle filtering where we make things discreet. But for the probabilistically inclined for you if you look at that and you realize remember we talked about marginalizing. What we're doing is we're saying all of the possible places we might have been before, what is the likelihood of me getting to some particular spot. And we have to weight that by all of the probabilities of having been at that point before. That's the likelihood of being at that next spot. And then you go to the next spot, and the next spot. That's the probability. That's your prior belief,your, of your prediction about the places you might end up. P of Xt given the measurements up until now. We don't have the measurement yet. Once we get a measurement, we can do a correction, right? And our correction, we use, doing Bayes' rule, right? We have the predicted value from before. That's the, the quantity we just talked about producing. And then we have an observation model. This observation model says, well. if you really were at some particular point what's the likelihood of getting the measurement you just saw and by using Bayes rule we are allowed we use that to turn that into a prediction of Xt given everything we saw including that's why is in red the new measurement and then like I said on the bottom here it's just the normalization

13 - End

All right. Well that was painful. [LAUGH] That's the mathematics behind sort of the general way of thinking about prediction and correction cycle of tracking. Fortunately it gets a lot easier when we look at specific cases. Spec, specific models of how to do that. What we're going to do in the next one is look at a very very simple analytic form of doing this called the Kalmen filter. We won't, we'll show you the equations, we'll you the update, we'll show you the intuition behind it, without necessarily deriving the equations. Then what we're going to do is show you sort of a more modern approach using it's referred to as particle filtering in computer vision, but basically it's this idea of using samples to represent densities. And when we do computing, envision, robotics, et cetera. The idea of using samples to represent densities gives us a lot of power because we're not limited to particular, analytic, distributions. So that will make those integrals, either easier or even better just simply go away. So, let's do that next.

7B-L2 The Kalman filter

1 - Intro

So welcome back to Computer Vision. We're talking about doing tracking. Now, last time we introduced this idea of tracking as induction. That is, if I know what's going on at time t, or I have some belief about the state of the universe of time t, I make a prediction for time t plus 1. Then based upon that prediction, that might affect how I go get my measurement. I get my measurement, and it's noisy, and I correct for t plus 1. And that continual loop, that's what we mean by tracking. And we did a little bit of, sort of, onerous mathematics where we talked about, the prediction part leveraged two things. It leveraged our belief of where we were the last time that makes sense and all right, in terms of, we might believe the position, we believe the velocity or something. And we had to have a dynamics model, that is, something that tells us how things will change, all right. So given some belief about how some things were, and how some things change, that gives me a new belief about where things might be. And then the correction, the, the update, after we took a measurement, remember we made use of Bays Rule. And the idea is, we take the prediction, we

multiply that by a likelihood, and we'll talk more about that, not so much today, but next time. And combining those and then normalizing allows us to know the prediction, our best guess now, of the state at time t after the measurement at time t. And that was referred to as the posterior distribution. So today, we're going to do a specific case of what's called linear dynamics, all right? Where the predictions are based upon a linear function of our current belief. And we're going to assume that we've got Gaussian noise both in terms of how things might change, so we have a distribution, we believe how things might change plus there's some noise added, Gaussian noise. And furthermore we're taking measurements and our measurements are a function of the state, plus noise, but what kind of noise? Gaussian noise! Very good. And we'll show a a simple linear recursive filter called the Kalman filter that's designed to handle this nice clean case.

2 - Linear Models

So what do we mean by a linear dynamics model? Well, it's pretty straight forward. A linear dynamics model says that our state at time t will be distributed according to a normal distribution whose mean is just the linear function of the previous belief, okay? So D here is a it's either a scalar or a matrix, depending upon whether x is a scalar or a vector. And then we take that and we put some Gaussian noise around it, some covariance. And by the way, you'll notice this little t here on these Ds, okay? What that says is, it could be that our dynamics model changes over time. That is the actual multiplication that we do, might change. But we assume we know it. So I'll tell you the vast majority of Kalman filters, those Ds are constant. So that the, the transformation and this thing here, which is referred to as the process noise, is constant. All that's absolutely required is that it's known for every time t. So that's the linear dynamics model. We're going to do an example of that in just a minute. We also assume a linear measurement model. And that looks stunningly similar, right? What that says is that the measurement we'll get will be some Gaussian normal distribution that's about some linear, m, here is a measurement matrix, that linearly multiplies the state. And that the measurement noise. Is also a Gaussian, and again, it's subscripted by t, because it could vary with time, but we assume that it's not. So, what I haven't mentioned is that this x, both on this slide, and on this slide, those x's are unknown. We don't, remember, it's the hidden state. We don't actually know what this state is. What we're saying is that this is how our linear dynamics model works.

3 - Constant Velocity 1D Example

Let's look at a very simple one-dimensional example. Now that's not very exciting but we've got our our one-dimension stage so x is where is this thing along some line. I may have state that says where it is, I might also have state about how fast it's moving. So here, what I'm showing you is a state vector that's both position and velocity. So, here, our state vector x is, I'm using p for position instead of x, 'because otherwise those x's would get too confusing. So, we have p and v. What's v? Velocity. All right? Now, going way back to, physics that you took in either 12th grade or 10th grade, 8th grade, or if you're really smart, 3rd grade. You'd learn some very simple, updated equations, right? That basically your belief about where the point would be at some time, t would be wherever it was at t minus 1. Plus delta t times whatever the velocity was. All right? That's just simple integration by velocity. And then you'll notice there this little element here. That epsilon, that's noise. That says, oh, there's some noise that's been added, right? So maybe things aren't exactly perfect. Further more, I'm assuming that my velocity is constant, except that some noise gets added to that. That is, I might speed up or slow down a little bit, but it's totally random, so I don't know anything about it, and so that's also going to be noise. Now, I can write these same set of equations in a very simple matrix form, okay? And here it is, that x at time t is dx plus noise. And here is this matrix, right? This is that matrix one delta t 0 1. Which times the state vector, there's our state vector there. Plus noise, gives us what your next state vector will be. So this is a linear dynamics model. We're using position and velocity as your state. Now what are we going to measure? Well, maybe we can only measure the position. So even though my state that I'm trying to estimate is position and velocity. Maybe I only measure the position. I've got a little radar thing that

pings out and says, oh, okay, I think this thing is about 2.4 meters away. All right? So, that would be just a matrix of 1 0 that multiplies the state, cause all it does is it pulls out the, the position. And again there's noise. Because there's noise in my measurement. Here's my m matrix, so this is the same linear model that I have for measurements. So I have a linear dynamics model, and I have a linear measurement model.

4 - Constant Acceleration 1D Example

And by the way, I happen to be estimating position, velocity. I could also estimate acceleration. Right? So, here my state position, my, my state vector is position, velocity and acceleration. All right. And I'm just including that to show you that that's a choice that you make as the model maker. You're going to build a model for doing this tracking, and this, and this is what we're going to do. And, again, I can write this out as a linear system, using matrix notation. But now we have, instead of a two by two matrix, We have the 3 by 3 matrix, which includes the components having to do with acceleration. By the way, the other choice, you also have when you do this, and I just mention this for completeness, is you can assume that there's noise in here or not. You can make those be 0 and just have the noise be in the acceleration. It's not that you're assuming your model is perfect. That is, that your estimates are perfect. You're just saying, well I'm actually going to believe that the velocity that this is how the position's really moving, and this is how the velocity's really changing. And the only place that I'm putting in any change is in the acceleration, and you can, you can do that either way. If you didn't understand what that meant, don't worry about it. Our measurement can still be the same. It can just be position only. And you might ask well, if it's just position, how do you ever get velocity and acceleration? These equations that we'll set up, will, whether you sort of do the invert to solve for things, it figures out that you have to take the differences between the positions, divided by delta time velocity and the difference between velocity is divided by delta time for acceleration. Even if we're only measuring position.

5 - The Kalman Filter

So it's time to introduce the Kalman filter. The Kalman filter is a method of tracking linear dynamical systems with Gaussian noise. And what's really cool is, both the predicted states and the corrected states are Gaussians, right? So you're only maintaining the mean and the covariance of your state, in order to run your filter. And also, the calculations are very straightforward. I'm not even going to derive the equations here. We'll be able to get some intuitions. But you can show that it's a relatively straightforward manipulation of these linear matrices. So in pictures, the Kalman Filter can be, can be described as the following, all right. You start off in the blue here is top left, that's your prior belief, right? So let's oppose we have some density. Now what kind of shape is it? It's a Gaussian, why? because this is a Kalman Filter, okay? Kalman filters are built on Gaussians. You want to have a non-Gaussian? Too bad, this is a Kalman filter, things are Gaussians. We'll do some other little stuff later. For now, everything is. So the first thing that happens is, in this picture, it's referred to as a deterministic shift. Remember when we multiply it by that linear system. So basically, that's the part that you know what happens. And that's shown here in green, where the, the Gaussian bump is just slid over to the right. So if I'm a Gaussian bump, which I've been accused of on a variety of times, the first thing that happens is I slide over to my right. That would be your left. Now Megan is wondering, should I move the camera? I don't know what this guy's going to do. I'm going to come back in a minute. All right. Anyway, so the first thing we do is we slide, beca, and we know that's the deterministic part. Then comes the part that's referred to in this picture, and I left it in here because I think it's a cool way of thinking about it, stochastic diffusion. But it's basically the addition of noise, and this is referred to the process noise. So now we've done two things. We've deterministically stepped one way and we've grown a little fatter. Both problems. Anyway, the idea is, when I'm taking a step forward in time, I use the part that I know. I know, I think I was moving, you know, three meters per second and it's a second later. So I think I've moved three, but I also know that I'm not perfect in my knowledge, right? That's never my problem, that's somebody else's. But, no, anyway, I know I'm not perfect in my knowledge, so

therefore, you know over here my variance is wider. Okay. So I've now expanded my uncertainty. Then what happens is I take a measurement, and that's shown here as Z. Okay, so I take some measurement and what happens is if you take a look at the density in the blue here, you'll notice that it is shifted back a little bit based upon the Z. And it has also gotten narrower, okay. It's very important to think about this in the Kalman sense, and we're going to do more of this in a minute, right. We shift, expand, measure, shrink. Okay, actually, shift, expand, measure, correct, shrink. That shrinkage happens because whenever you take a measurement, you're getting new information. New information can only reduce your uncertainty. That's going to come back in a minute to be a little strange.

6 - The Kalman Filter 1D State

I want to introduce we're going to keep doing this Kalman Filter now on the 1D state. And I want to introduce a little bit of nomenclature. Some a way of writing things down. So what we have here is, we have this inner loop that we talked about. Where we predict, make a measurement correct and then time advances. Okay. So we time advance we predict, we make a measurement we correct. The first thing I want to show is, in our prediction, which is probability of x t given the measurements up through t minus 1, our prediction in Kalman land here is going to be model written as, mu minus sigma minus, sigma squared minus right. This is and see of timed t. So mu minus sub t, sigma squared or sigma minus sub t, that's the prediction of the mean and the co-variance before I take the measurement at time t. That's why it's referred to as minus. Then I take my measurement. All right. And my corrected version, which is now, the, the distribution of x given the new measurement y sub t is referred to as mu plus sub t sigma squared or sigma plus sub t. So I have these two things. I have mu minus. Sigma minus the time t that's the prediction before the measurement. Mu plus sigma plus at time t, that's my correction after I've taken my measurement.

7 - 1D Kalman Filter Prediction

Let's go through the prediction and correction just a little bit in the Kalman 1D case. Remember, we're going to assume that the state of time t, okay, is just a constant times whatever the state was at t minus 1, plus some noise that's added, some process noise. Right? So the first thing we would like to do is we would like to know our estimate. What's our best guess? At time t, given everything t minus 1, when we've got this Gaussian model. Well, we're going to write this the following, right. If, if something is a Gaussian, and I multiply it by a constant, what does that become? It becomes another Gaussian. Or another way of saying it, I have some variable that's distributed by a Gaussian, and I multiply that variable by a constant. I get a new value out. But it's also a Gaussian, all right? And in fact, if you remember from probability right? It's mean is just going to be that constant times the previous mean. So that's what this equation here says. This equation says that if I have my belief about the mean at time t minus 1. After I've done all my work at t minus 1. And if I believe that my dynamics are multiplying the state at every time step by d, my belief before I take a measurement about what the new mean is just going to be d times that. That's what this equation at the bottom says here, right? Mu minus a time t, the predicted mean is just d times the mean from what I had before. Okay? Because d is my multiplier. All right, so you remember that when I multiply something by a constant and it's Gaussian, the mean just multiplies. But what about the variance, okay? Now we have to update the variance. There are two things that are going on. First of all, we're taking some previous variable and we're multiplying it by constant. That variable had some sigma, sigma squared to it. And maybe you remember if you have a Gaussian, when you multiply that by a constant, its variance goes up by the square of that constant. So just In the multiplication part, all right. Whatever the variance was before is going to go up by now d squared. But wait, we're not done. We also added in some more noise when we took that step, right, of d squared. So that means the new variance before we take a measurement is made up of two components. It's d squared times whatever the variance was before so that's written as d sigma t minus 1 plus squared. Sigma t minus 1 plus is the standard deviation if its squared. It's the variance of what we had at the end of t minus 1. We multiply everything by d. So we multiply that variance

by d squared. That's what this term over here is. And then, we added in some more noise. Because of uncertainty of how the thing actually then moved, that is this d squared okay? So even if we were multiplying by a one that is even if we had no idea. There was that, you know, everything was just a one, our variance would still grow because of the added uncertainty. All right? So the, the update of the mean or the update of the variance for the Kalman is written like this. All right.

8 - 1D Kalman Filter Correction

What about the correction? Okay? Well the correction is a little complicated. So what I'm going to do, is I'm going to show you the formula for it. Then I'm going to give you the intuition behind it. Then I'll show you, actually, not the formula, I'm going to show you the simple formula for the 1D case, give you the intuition behind that. And then give you the full formula for the ND case. First remember that we're assuming our measurements are again just some constant times the actual state plus some Gaussian noise. Okay? That's our, that's our linear system plus Gaussian noise for the common filter. Remember, we have our predicted state. We just did that. We said that that's Mu sub t minus. All right? That's the predicted mean, and then sigma t minus squared that's the variance of the prediction. And what we want is we want the corrected estimated distribution. That's what this is, here. All right? So in Kalman with linear, Gaussian dynamics and Gaussian measurements, the corrected distribution is defined to be, and I'll show you that, a new Gaussian, okay? Which is now mu t plus and sigma t squared plus, all right? And what I want to do is, I want to show you the formula for that and then give you the intuition behind it, okay? This is the formula for the mean. Okay? So here is mu t plus, and it's a combination of mu t and the y, which we'll get to in just a minute. The update the variance is here, so sigma t plus squared, okay? And you'll notice that what's happening is that this number, if you just do a little bit of this math here Is actually going to end up being smaller, okay, than the predicted variance. And that makes sense why? Because remember after we take a measurement our variance goes down.

9 - 1D Kalman Filter Intuition

So let's take a look at the mean. This one will be a little more clear, all right? So, this is the equation just copied from the previous slide to this slide, all right, for mu plus. Step with me, I'm going to divide by m squared. Okay? So, when I divide by m squared, okay, so you see, I've just divided by m squared. So, there's an m squared down here and this m becomes m there, and this becomes an m squared there. Okay? Cool. So, what is that? I have no idea. Well yes you do, here it is. Look, see this yt over m. Remember, y measurement was just m times x. So, yt over m, that's the measurement's guess of what x is. All right, if you were looking just at the measurement, that's its guess. Now mu t minus, what's that? What mu t minus is, that was our prediction, right? Our, that's the mean, that's our best guess as to what x actually is at time t okay? So, we've got these two things in here, right, we've got mu t, which is our prediction, and y over m, which is our measurements guess, we have to somehow combine them. Well how do we combine them? Well, what's this value, right? This is just sigma squared divided by m squared, that is the measurements variance divided by m squared, that's the variance if you were to compute x just from the measurement. So this guy right here, that's the x computed just from the measurement. The variance of that is sigma squared m over m squared because y was just mx. All right? So that quantity right there, circled in blue, that's the variance of x computed from the measurement and, what is the last term there? Sigma t minus squared. That's the variance of the prediction. And you might say, well, so what?" So what? What do you mean so what? No. What this means is that the entire mu t plus is just the average. It's a weighted average of the prediction and the best guess based upon the measurement. And so down here in the denominator, that's just the sum of the weights. It is just a weighted average of the prediction plus what the measurement is telling you, which of course is what, in some sense, the universe has to be. Right? The whole idea is you're making a prediction, you're getting a measurement. Both of these are sources of information and you want to integrate them

10 - Prediction vs Correction

In fact, we can take a look at that a little more carefully, all right? So here are the expressions that we have before for the new mean and the new covariance. Suppose there's no uncertainty prediction. What that means is that sigma t minus equals 0. I don't know how that would've happened, but let's suppose. You just know where this thing is. So then what happens is if, if this is 0, okay, gee let's take a look over here. Then this thing is 0 here. This thing is 0 here. This thing cancels out. What it means is that your best update is the one that just uses the prediction because you knew exactly where it was going to go. And furthermore, your new co-variance is 0, that is, you have no uncertainty. Another way of saying this is, your measurement is ignored, okay? So if I'm sure of my prediction, I should ignore my measurement. Likewise, what if there's no measurement uncertainty, suppose your measurement is perfect, what should you do? Well, if you plug sigma m equal to 0 in here, you'll see that mu t plus your new best guess is just the measurement divided by n, that is, it's just the measurement's guess as to what x is. You're totally ignoring the prediction. And this is what Kalman does for you, right? It gives you this tradeoff between how much you pay attention to the prediction, versus how much you pay attention to the measurement as a function of the uncertainty of the prediction and the uncertainty of the measurement. And that's what makes it so powerful.

11 - Kalman Filter Processing

All right. Now, I'm going to make some simple changes with a and b, right? So, a is here and a is just this quantity there and b is here and b is just this quantity there. In which case, this whole expression, I replace the a and b, keep going. Again, a and b, carry it forward here, here, here what you'll notice I've done is I took a here. I added in the b there, which ends up subtracting that there. Keep going and what you'll see is this expression that says, mu t plus our new belief after we've taken our measurement is just mu t minus. What's mu t minus? Mu t minus was the prediction and then we have to change that prediction a little bit and that prediction is changed based upon two things. The first thing is based upon this difference here. What's that difference? That difference is the residual, right? Y divided by m, that's the measurement guess. Right? That's the measurement's guess of x. Mu t minus was your prediction. The difference between them is the residual and I multiply that by some factor k and that k is called the Kalman Gain. If my Kalman Gain was zero, then I ignore my measurements all together and I just go with my prediction. Okay? If my Kalman Gain was one, let's see I put mu t minus t. I ignore my predictions all together and I just get my measurement. And it's that Kalman Gain that you're estimating that goes between essentially zero to one in the scaler case and does something funkier in the multidimensional case. All right. Let me show you that in a simple e, example. And this is going to be a constant velocity model and what we mean by that is we assume the velocity is constant, except a little bit of noise. So the state is two-dimensional, it's the position and the velocity. Right? So, it's, I'm, I'm moving in one dimension, but my state is two-dimensional, because it's the position and the velocity. And now we're going to take a look at this graph, all right? So, in this graph, all right? So we have time and position as the things we're measuring and the thing we're estimating. The red line here is the state. This, we happen to know was the actual underlying state, okay? How did we know this? We booked a trip to Delphi, you know where Delphi is, it's in Greece. They have a thing there called the oracle and they also sell really good Baklava, but that's beside the point and you go to the oracle and the oracle tells you the truth. Okay? It tells you if the Mets will win the world series. They always say no and they also tell you the state, but the idea is that for, for the points here, I am just showing you the graph. That is the actual state. But we also took some measurements, okay? And the green dash lines, these are the measurements we happen to get. Now you'll notice, pretty crummy measurements, right? Life is the way life is. This is the noise that we actually happen to get. The purple line here is the corrected mean, okay? And what you're seeing here is that the mean is sort of being pulled, the estimate, all right? Is being pulled around by the measurements, but you see that it tracks the real state much better than the measurements do, right? because it's blending a prediction, where it's estimating the velocity with the ability to incorporate the measurements. If you

want to take a look at the, the error bars here a little bit, one of the things that cool. In fact, I'll look at just the last one. Right? This is the error bar of our prediction, okay? So remember, you see, it says, the predicted mean, which is right there. And after we did the common filtering this was our new guess, all right? And our error bars got smaller, okay? So, again, this is what's happening is we take a step, we get bigger. We take a measurement, we move and we get smaller.

12 - N-dimensional

What I just showed you was a formula for the one dimensional case. We almost always do this in the n dimension because even when we were having x move in just one direction, we had a two dimensional vector of position and velocity. In the two dimensional case, these are the Kalman equations, okay. We have our state vector x. We assume, our prediction is just based upon a linear transformation. Our covariance prediction, or the pre, the covariance of the prediction, this is that squaring operator, remember before, we did D squared sigma? Well, when you're doing this in matrices, it's D times the old sigma times D transpose, that's the squaring. Plus you add in the noise, the process noise. So those are your prediction measurements. And then your correction looks like the things on the right. The thing for you to notice here, is, you see the, the quantity in the green box? This is the residual, right there, and you're multiplying it by this Kalman gain matrix, all right? And the Kalman gain matrix has less weight on the residual as the prior covariance estimate approaches 0. So as this value's, it's, it's hard to say what it means for covariance to go to 0, but let's talk about its traits, but the idea is that it's getting smaller. As it gets smaller, the Kalman gain goes down, which is what you want to have happen, right? If your covariance of your prediction is very tight, I don't want to pay a lot of attention to my measurements. So I don't want to weight the residuals very much. On the other hand, if my measurement covariance goes down, okay? So now, what happens, is this value is very small, which means this value is what's getting put in the inverse, right. So now this sigma t goes away. What happens is the Kalman gain matrix goes up, its magnitude, if you will, goes up and it weights the measurements more. So, these matrix equations are doing exactly the same thing as the scalar equation did of balancing the prediction, and the measurement.

13 - Tracking with Kalman Filters

There's a really nice little a graphical illustration of tracking with Kalman filters. So let's suppose we have some current estimate of where we think things are, right. And that's what's represented by this little circle here in the x y plane. Our prediction shifts and grows, and that's why the, the size if you will. And you can think of this as being a a single, say, some number du, of standard deviations of uncertainty. So our, our uncertainty matrix got significantly bigger, and the mean may have shifted as well as, also. So that's our prediction. We then take a measurement, and you'll notice that the measurement here is offset a little bit. I mean, and it happens, you know, it's, it's mean is right in the middle. And it's actually tighter. So it actually has a tighter co-variance than the prediction, okay? But the thing that's important to realize is the correction has a tighter co-variance than either one of them. So you'll see that the mean of the final correction is somewhere between the mean of the measurement and the mean of the prediction. Remember, it's going to be a weighting between them. And also notice that the co-variance of the final corrected estimate is smaller than either of the two, all right? And this might seem a little weird when you look just at, say, the, the co-variance of the measurement and the co-variance of the prediction, because they're both pretty big and you might ask, how is it that the co-variance goes down even tighter than the measurement? And again, the answer is, information can only reduce your uncertainty. So whether you think of as measurement as reducing your uncertainty about the prediction or the prediction is reducing the uncertainty about the measurement. What that means is your final co-variance will be less. You'd be amazed at the number of PhD qualifying students who when asked that question standing in front of the board kind of flunk through it. So if you're thinking about doing a PhD and I'm going to be on your qualifying committee and you're doing something in robotics. Get to know and lerve, love, your Kalman filters.

14 - A Fake Quiz

This can be seen, sort of simply in this one d k. So we start off with a prior and here it is as a Gaussian. We have some measurement, that's the evidence, and here it happens to be tighter. And then we get a posterior, all right? And the posterior is narrowing because it's narrower, it's also going to be taller and then the prediction step, we just take that posterior and we do this shift and we grow the uncertainty again. So, so far, everything looks like it's behaving really well. So, let me show, give you a quiz. Well, it's not a real quiz. It's like a fake quiz. All right. Suppose this is your prior. Okay? Okay so you see it is this nice white Gaussian lump over here. And then suppose this is your measurement. Okay. This nice tight green. Where does the posterior go? Well, first of all, it's gotta go somewhere in between, because in Kalman filtering, your mean is a weighted average between the priors mean and the measurement. Okay? So the fact that it's going to go in the middle is okay. But what might bug you is it's got, again, that tight little uncertainty. I've got this Gaussian bump here, all right? Now Gaussians go to zero, really, really fast so by the time we get out here, it's like incredibly, un, unlikely. But then all of a sudden there is a measurement bump over here. Okay. So you know, what's poor Kalman to do? Well, Kalman has to put them somewhere in the middle, because that's all it does. And furthermore, it has to, it, it has to reduce that uncertainty because it's a slave to its model. It really honest to God believes that things are distributed by these Gaussians and if they really were, then that's the correct value to put in. And the question is, does this agree with your intuition? And the short answer should be, yeah, I don't think so because Megan, over here. Over here, okay. If I've really got a Gaussian lump over here, then the idea that something will now be over here, the probability is so infinitesimally small that what's probably the problem is that our model is wrong. We really should be trying to abandon the model at that point as opposed to doing the integration. And in some sense, that's the, the, the, the, the part of the message of Kalman filter. You're actually a slave to the model you really believe in.

15 - Kalman Pros and Cons

So reviewing it, sort of the pros and cons, what's the biggest pro of account filter. It's all listed here as one but look, it's just very simple. You know those equations. You just implement them. You write them. It's compact. It's efficient. It works pretty well. You have to surf, sometimes tune some things like you never want to let your process noise get too small. It's much better to overestimate process noise than under. Why? Because if you, for example, suppose you didn't think there was any process noise at all? You thought this point is just there, okay? So you're taking measurements, well, if there's no process noise, remember, every time you take a measurement, your uncertainty goes down, eventually you go to an uncertainty of zero and you'll stop listening to your measurements. So it's always good to have a little bit of process noise. So there's some tuning to make it work, but in general, it's pretty clean. The biggest problems with it are that it's first of all, fundamentally a unimodal distribution, right. I've basically got a Gaussian, which means I basically have a single hypotheses. Allright? And it's some uncertainty within that hypothesis. There are some other issues here. Remember we have this linear model. There are ways of fixing that. There's something called the Extended Kalman Filter. Which basically just takes, for those of you who care. You linearize about your current point. That's like taking a Jacobean around a nonlinear system. you currently estimate it linearly, update, then recompute your equations in the nonlinear. Prediction, so the linearity part is not that big of a problem, but this problem of [INAUDIBLE] having a single hypothesis okay, that's a Gaussian, that doesn't work so well sometimes, so the question is what you should do if the world is not Gaussian or if it's not even unimodal and the answer is, you should come back next time, because we'll tell you then.

7C-L1 Bayes filters

1 - Intro

Welcome back to Computer Vision. Today, we're going to continue on our long and, and winding excursion through tracking. And in particular, what we're going to do, today, is we're going to revisit the whole notion of base, based fi filter tracking or, or, or Bayesian tracking. And sort of develop a method that allows us to do that well. So you remember the tracking with dynamics, right? So the whole idea was, because we're tracking something that's moving with respect to the imagery, we're going to learn something about the dynamics and use the dynamics in order to make predictions. And we're doing this so that we can limit the area of search that we're looking in, and also so that we can get a better total estimate when we're done under the assumption that the things we were trying to track it's, you can't perfectly detect them and it's noisy to begin with. And we're basically making an assumption that there's some sort of smooth motion going on and that essentially our dynamics don't surprise us is, is the underlying element. So last time we developed what was called the Kalman filter. And the Kalman filter was for linear dynamical systems, dynamic systems where you had Gaussian noise contexts. So what we meant by that was the motion could be predicted by a linear factor applied to the motion that you've estimated and then you add in some Gaussian noise, if you were thinking about what the uncertainties are. Same thing with our measurements. They were a linear function of the state, and then there was noise, and what was nice about it is everything, prediction, correction your lunch break, it's all Gaussian, all right? And so the only thing you have to do is maintain the mean and the covariances of these Gaussians and it just works in one d or n d, etc. It's really easy. And the calculations are pretty straightforward also. I mean the, the equations looked a little ugly, but it's just a bunch of matrix multiplies and a couple of inverses

2 - Propagation of Densities

Pictorially, the Kalman filter looks like this. Nice Gaussian. I don't know why I'm drawing this. It's already here. A Gaussian, I'll circle it. We start off with that Gaussian. We then have the linear prediction part, which does the shift. We then say, but wait a minute, there may be some noise, so things get widened. We take our measurement, and we sort of blend. Remember, it was that the Kalman filter is a, a blending between your prediction, which was what our, our prediction here was. And your measurement, and it creates a new estimate that's a weighted average between those, depending upon the relative uncertainties, and the uncertainty gets smaller after we take the measurements. So uncertainty gets bigger when we do our prediction, because we don't know exactly what happened, and it gets smaller again when we take the measurement. We said nice things about it was that it's simple, it's compact, it's efficient and it's very easy to spell. The cons of it are in fact, maybe I should have listed these in the other order. Here it says, we'll say there's sort of a restricted class of motions defined by the linear model. Linear thing we can get around by using what's called an extended Kalman filter. A much bigger problem is, it's nice, it's got this nice unimodel distribution assumption. You're assuming the world really is a nice Gaussian of some uncertainty. And the question arises, what would you do, if it, things are not a Gaussian? So what's an example of when you might need a non-Gaussian situation? Well you remember these videos? Okay, and again this comes from the Isard and Blake condensation tracking paper. There's all sorts of dynamics going on here with things jumping around. Let's see if I do that again, yeah there she goes. And things move quickly, they they may be able to be nice and smooth sometimes, then something rapid will happen. So like when the leaves are blowing in the wind for a while they blow nicely. But basically, the problem here is that things are moving smoothly and then all of a sudden you'll get a perturbation, and you'd like to be able to keep track of that. So graphically we're going to show that like this, okay. So now we have the same model that we had before, okay? We have a, an estimate from the last time. We have a deterministic shift. We apply some smoothing because we grow the uncertainty. And then we take our measurement and we update our estimate.

But now these are not Gaussian lumps all right. These are sort of arbitrary distributions, and in fact this particular figure is taken from the paper that introduced the particle filtering that we'll talk about in a minute, to computer vision, that, that condensation paper, I just mentioned.

3 - Particle Filters Basic Idea

Before we go any further, just something you should know. In particle filtering, the measurements are typically written as the z's, z sub t and not as y of t. It was just the way it was done. It has to do with whether you're connected to vision or robotics in order it, so that it aligns with various papers and other formulations you'll see. We're going to start s, showing z's in the slides to come. It also made it much easier to borrow slides from other people, because they were using the z's. All right. So the key to sort of particle filtering is the following. If we have a, a distribution and here we have a nice little distribution, which you see is, I don't know what that distribution is, but I know one thing, it's not Gaussian. Now somebody who's really smart out there might say, well, it looks to me like a mixture of Gaussians. Well, it sort to looks to me like a mixture of Gaussians. Yeah, in this particular case, maybe it is. That's just for convenience of drawing. And in fact, there is a relationship between mixtures of Gaussians and particle filters, which we won't even talk about. But basically, it is you have a bunch of things with together represent your density. In particle filters or sample based methods, what you have is you have a set of particles and here the particles are drawn out where you can see these particles. It says, Xt. That means these happen to be our particles at time t and there's n of them and, you know, the bigger N is the better. And you'll notice it says, weighted here in parentheses, because sometimes they're weighted or sometimes they all have the same weight. It, it, density is represented by sort of where the particles are. So you see we have a nice high density here, so we have lots and lots of particles there. Okay? Not so many particles here, not so many here and lots of them there. All right? So that's where they are. And if they were weighted, it would be as if you, you can think of that as having two particles in the same spot. If I double the weight, it'd be like having two particles at the same spot. And now you can say that the probability density that x is some value is essentially the probability, if you were to draw out a particle that you would get a particle whose value is really, really close to that spot. Okay? And that's what it means to use samples or particles to represent a particular density. And if you're doing that, then your overall goal is you want sort of as n gets bigger, the idea is that the probability of getting a point from this distribution, that's x is an element of xt is a approximately what the underlying probability was. And for tracking, the idea is given everything we've observed so far. So what are our observations? Remember, they're z's. So given everything that we've observed so far, what's our belief about where x is and we want to end up with a set of particles that gives us that density.

4 - Perturbation

We still have to do the same things we had before, so you remember, in, in tracking,when we were doing tracking we had this notion of a dynamics model for and it was a linear model. So the idea is that at each time step, we have some prediction that, allow, us say, well, if I have some density about where I think I am, what's my new density? And in that particular case, it had to be linear in order to make the math work out. That was, you know, the underlying part of the, the Kalman. All right but in general, we can have a more general state transition matrix, right? So the idea is that what is the expected change from time T to T plus one. Okay so if I'm at some point or if I'm guessing that I'm at some point I can say what's my expected change. So what's important about this and the reason I'm going on and on about this is besides just having a sort of dynamics model that says how things are likely to evolve I can also intru, include a notion of perturbation or control. All right. So, the perturbation means if I was here, maybe I know that I added, I took two steps to the right. Now Megan's freaking out again. All right, I'm back. All right. So what I would do is I would say that okay, the the expected state transition is that I take where I am, I take two steps to the right, and then I add some noise. So it's a little bit like, remember when we talked about that in a linear model I could actually have a different linear transition at every time t, but usually we keep

them constant, here, we're going to have the notion of an input, and the input is represented as u, u for input. Don't ask me. You, if you, if you want to know why, ask some control theory professor, because they always use u as the input, so it's their desire. Anyway, so we're going to see a u sub t. That's the control, or the input, or the perturbation that got added to the system at time T and we assume that we know that, alright? So what we've done is we've changed our filtering just a little bit so that besides just taking observations, we're going to assume we have observations plus assume, that we have, we know what the inputs are, all right? Lots of words but I hope that you follow that.

5 - Known Perturbations Quiz

Now why would you want to add perturbations to a model? Isn't that what noise is for? Let's consider a real life scenario. Let's say I'm a robot ambling away on the moon. My goal is to reach this flag. I can use my camera to detect and track the flag. This is what my view of the world looks like. Now, where the flag shows up in my view and how it moves, depends on a bunch of different things. For instance, my movement across the surface, both back and forth and turns. Now the surface is really bumpy. This can cause my view to shake quite a bit. I can also move my camera independently, pan, and tilt. The most annoying is probably the wind blowing the flag all over. It really messes up the appearance. So which of these are known perturbations that we can model for?

6 - Known Perturbations Solution

Okay, this was fairly easy. When we're moving, we know how we are moving. That is we know how our view of the world should change with the movement. Therefore, we can take it into account in our model. Same goes with turning the camera. Now, perturbations caused by the terrain or wind are things that we cannot explicitly model for. We still know that they can affect what we see, so it's probably a good idea to include it in our noise model. Known perturbations, or control inputs or actions, allow us to build in phenomena that we can explicitly model for.

7 - Bayes Filters Framework

So we're going to go over and build Bayes filter framework and so given four things. First is I'm going to need some prior belief about where I'm starting okay, so that's p of x. Where did that come from again? Remember that drive up to Delphi and asking the Oracle and all that stuff. So you forget your luggage so you went back to Delphi. You got another prior and it tells you how things are starting, but again, maybe with a large uncertainty. So here we have our dynamical system model. And so it's the probability of some new x given my belief about the previous x plus this input. By the way, you see here it says u sub t minus 1? The, the, the world is sometimes schizophrenic about whether you think of the input as happening just after the last measurement, so it was u at t minus 1 or u happen now and I want to estimate what the state is now. So, whether it's ut or ut minus 1, it's whatever u you're going to add onto xt minus 1 in order to make your prediction about xt. All right? So, we're given these two things. We're given a prior, an action model. We're also given a sensor model. And this is going to be really important, a little bit important this lesson, very important next lesson. And the sensor model is our likelihood, okay? And it is not, I'll tell you what it isn't, first. It is not, given some sensor reading, reading, where do I think the object is? You might think that's what it is, but then you would be thinking incorrectly. No, a likelihood model is if the object were really someplace, what's the likelihood of my measurement? So normally what you'll see is and we're going to see some later like a little Gaussian blob drawn over the place where the measurement was. That does not mean, okay, that given this measurement, this is my distribution of where things are. No, no, no. What it means is, this is my likelihood, so it's highest if x was actually at this point, and then it falls off slowly. All right. That's the likelihood. And then finally, we're given the stream of observations. The z, so z1 through zt minus 1, or through zt. And also, we know the actions, okay? So we know the action data u, u1 through u t minus 1, okay? So those are the four things that we have. All right? What do we want?

Well what we want is the estimate of x at time t. Okay? Same thing we've always wanted since we were little children. And in the Bayesian world, this is called the belief for the posterior of this state. Right? And so sometimes it's even written like this. Bel for belief of xt. What's my belief about xt? Given everything that I've observed so far, knowing all their prohibitions and given my current measurement.

8 - How Likely Quiz

I'm still a little fuzzy on this likelihood thing. Let's jump back to the moon. Did I tell you about my star sensor? I use it to navigate around. There are hardly any other landmarks. When I point it to a region in the sky, it says, bright star. That is, if there is a bright star in that region. If I told you that I was looking at this particular region in the sky, how likely is it that my star sensor would go off and say bright star? Choose the appropriate radio button underneath on a three point scale. From least to most likely. Now do the same for these other regions in the sky. Remember that I'm asking you to estimate the likelihood P of bright star given the sky region.

9 - How Likely Solution

Okay, the first one is very likely. Second and third? Eh, not so much. Last one? Eh, maybe. Okay, so the distribution of P bright star, given sky region, looks something like this. Remember that this is not enough to give me a definitive answer of which region I am looking at. I need to combine this with my prior probability of where I think I'm looking. This would come from my previous estimation step, or just be a random distribution in the beginning.

10 - Graphical Model Representation

A couple of more things about the model that are similar to what we did before for the Kalman, okay? Since I know some of you have a machine learning background a little bit drawn here is a graphical model. The idea is that x is only impacted by what the x was before, and what the new protobation is. And the z's, the measurements, are only a function of the current state, all right. And that's written in the following way, right. The first thing says the probability of the measurement, given all the states, given all my histories, only a function of the current state and that's sometimes called sensor independence. Okay, the sensing is only dependent on the current state, all right. The other element is this one that says the probability of the next state given all of these things that came in and including the states before is only a function of where we were before plus the current input. And by the way, here, this is written as ut, not the ut minus 1, that's what I was telling you about before. The shame on me, because I did both of those and I should have made them consistent. But the idea, this is the Markovian property, right? We ha, we had the Markovian property last time also. And Markovian property says, if I know the state at the last time, none of the earlier state information, none of the earlier measurements, none of the earlier anything matter except the state then. And then the one thing we've added new this time is action at the current time, okay? That's the Markovian property.

11 - Bayes Rule Reminder

So before we go further, I'll just remind you again of Bayes Rule. We looked at this last time. The straight Bayes Rule is that we can say that p of x, given z, the p of the state given the measurement, can be written this way. This is the likelihood. All right? This is the total probability of z. This, that's our prior before the measurement. That's our belief, if you will, of, of, of what the value was. Okay? So, what's important here to realize is, once I get the measurement. Somebody gives you the measurement. You drove to, to to, to Measurements-R-Us, and you picked up your measurement. You didn't know which measurement they were going to give you, and there's a probability that they'll give you all these different mea. Once they give you a measurement, the probability of

having got that measurement is done. It, it, it is whatever it was. It's not going to change. So what that allows us to do is to sort of replace that, by this, normalization factor. Right? It's some value. Or another way of saying this, is that p of x, given z, is proportional to p of z, given x. All right, times the prior. So if I had a whole bunch of different possible x's, and I only had this value, okay, for p of x, given z, they have to sum to one. So what I could do is I could just take the sum of all of those, whatever they were, and then divide this value by that, divide by that sum, and I would be back to a real, renormalized probability density. All right? That's why we write it as this Eta and we talk about this proportion. And, in fact, next time when we do particle filtering, you'll see that what we do is, we if we sample all these x's, we multiply them by their likelihoods and we get all these values, but they don't sum to one. So what we have to do is take this, take all the sum, sum them up, then divide each of those by that sum and it'll sum up to one again.

12 - Bayes Filters

What I want to do is work through the, the slides here and the base filter, getting us, to how we come up with our new belief okay? So this is just our definition before, that the belief is the probability about the current state, given everything including, the current measurement, so we're going to use the Bayes rule that we just showed that says we can swap that around by putting in the likelihood and the prior. And the prior is just Xt without the Zt over here, right? That was our prediction about what Xt was and this is this likelihood model, right? This is just our sensor likelihood model. What's the likelihood of getting that value? All right. Well our sensor independence that we talked about before says, if you have this state, xt, then none of this stuff matters and it's just p of zt, given xt. That's sensor independence. All right, so here we are. So the next thing we can do is remember we talked about this before. This notion of total probability. The way we do a prediction. Okay, for any given xt what we have to do is say, well, what's our prediction of all the possible xt minus 1. What, why are there many of them? Well All the possible x's at t minus 1. If we take all the probability okay, that we would, we sum up over all those possibilities, and here, you notice what we said was, what's the probability of going to xt from xt minus 1, times this probability of xt minus 1, summed up over all of those xt minus ones. That will be the, the prediction, that will be the prior. But this value, actually, can get made much smaller. Because, notice. That has this term and that term. Because remember the Markovian property says that if we have XT minus one and we want to predict XT all this other stuff doesn't matter except the action. Right. So, basically you get me where I thought I was before and what action you did, I can tell you about where I think I am now, and that's what this Markovian step is right there. One more time, flies up, wow, beautiful. Okay, keep going. All right. Well, now, actually, we're pretty much done, and now everything is beautiful and color coded, all right? We start here, right. This value right here, that's just our belief about all the xs at the last time step. So that's written here. So now we're already looking at a recursive algorithm, right, because on the left-hand side, we have the belief of xt. On the right-hand side, we have the belief of xt minus one, so. This is a induction, remember we talked about tracking induction. This is induction formula that goes from your belief that T minus one to your belief at T. When you integrate or sum overall the possible xt minus ones, these are your predictions before taking the measurement. So this is just like we did with the Kalman before. To make your prediction, what you do is you take your belief about where things were, you pump it through your dynamics and you add it onto your uncertainty. And then what this is showing is, to get our new belief, okay, we just multiply that by the likelihood of the measurement. Remember it's not the It's not your probability of where you are given the measurement. It's how likely would this measurement have been if that had been the particular x. And then we can worry about this normalization later. That's how we get the belief at new time t.

13 - End

We're going to stop here for a little bit, let you catch your breath. Go back, review, have a beverage of your choice, make sure you understand this or just complain that gee, this seems redundant with what we did right before the Kalman filter, because yes, the math is quite similar. In fact,

underneath it's identical. But what we're doing is we're moving to get to this notion of a normalized Bayes' rule application as opposed to the straight Kalman. And then what we're going to do is we're going to use that particle representation that I showed you before to, to do this and it'll all become much clearer I hope.

7C-L2 Particle filters

1 - Intro

All right, welcome back to computer vision. We're going to continue introducing particle filters for tracking. I also want to say thank you I think it's Sebastian Thrun and, and/or Dieter Fox or somebody else who's done a bunch of slides that have actually lost their, their authorship is lost to all of humanity. Because if you take a look at particle filtering, you'll see these same pictures everywhere. So whoever did them, thanks. All right, so you remember I hope, that the reason we're doing all this is because unlike Kalman filter where we were going to represent the densities to where things are, are the nice Gaussian. The idea is we might want to have an arbitrary density. And graphically, this, that's what's shown here, the idea of propagating a density through time. Now, the basic idea that we're going to make use of is instead of having some sort of analytic representation of our densities, we're going to have these set of particles. We've talked before about how the distribution in terms of where the particles are and perhaps the weight of the particles. That's what we're going to substitute for the density. Right? And our goal is, so that as we're doing this tracking, eventually we have a set of particles such that the probability that you get a particle at some particular place is essentially the probability that the state is actually at that place having observed all the observations. And the way we do this is we're going to do this as a Bayes filter. We introduced this last time, where we said that basically you take your prediction from before. And the way you get to your prediction is, you take your previous belief of all the different places you might be. And talk about the probability that you could end up at some given xt, you some that all up, and that's your prior belief that you could be at xt. You take your measurement, you multiply it by your likelihood and because of Bayes rules which does the, the swapping of the variables, that gives us our new belief about xt. So we're going to take particles and we're going to define a Bayes filter using those particles.

2 - A Simple Example

We're going to illustrate this with a very simple example and here it is, so imagine you've got a robot that just has two things, okay? It is a map of a hallway, okay? So, here you're seeing a hallway and by the way this little blue dot here, this is the robot, the robot doesn't actually know where it is. We know where it is because we're really really smart, okay? Or we work for the NSA, I hope that didn't cost me my grant. Anyway, so the, the robot knows this, this map and it knows that it, essentially where the walls are and where the doors are and we'll assume that the doors are open, 'kay?. The other thing that the robot has, is it just has a detector, a sensor really dumb that just looks out to the side. You can even think of it as just gotta stick and it pokes it. And it says, do I hit something or don't I? And by the way, it's, it's not very accurate, so he sticks it out there and sometimes he sticks it this way and sometimes. So it's, it's approximately that direction he just says, did I hit something or not. And you might thing that might be an incredibly dumb robot and you might be right except here's a picture of a robot. I don't want to insult my SRI friends because this robot actually got pretty smart but this is the robot originally, flaky here. And flaky had down below on the bottom, these little ultrasonic sensors that would send out a chirp and it would respond back. Did I hit anything? Or how far away was the things that I hit? So, humor me that this is just a robot that basically detects, just a hole or just a not a hole. That's the only thing that it knows and because it's not perfect in setting out its, its signal, right? Just because it said, the sensor says hole, doesn't mean there actually was a hole, right? And just because, it says wall doesn't mean it actually was a wall, okay?

3 - Sensor Information

So this example, this robot that's going down the hall, it's got this noise sensor, this is perfect for sort of a probabilistic Bayesian filter based tracker, and the reason is, first of all, it's going to have some continuous belief about where it thinks it might be. Right? So that's its belief, it's distribution, it's posterior about where it thinks, and it's measurements are noisy, and furthermore, it's really unlikely that depending upon those measurements that it will have a nice Gaussian belief about where things are. It's belief is going to be whatever it is. So we're going to make make use of a particle filter to do this. All right? What you're seeing here, is this is the prior density. So here's our little, should we name our robot? Sure, what's a good name see, I would use some like R2D2 kind of name but then I would get sued by George Lucas. So we'll call it R3D3. Okay, so R3D3 is sitting over here and this here, this is his prior belief about where it is right now. We have no idea how it got that value, this is it's random distribution, and you can see it's a pretty uniform distribution of particles. Now the sensor reaches out and it gets the measurement and let's just suppose it sticks its stick out and it sees a hole. What that means is, the stick doesn't run into anything, so it says, it's a hole. Now using the sensor model, the robot has to evaluate not where it thinks it is but first, what's the probability that I would see a hole if I was at each of these locations? So,after the robot sees the hole, what it has to evaluate is what is the likelihood that I would've seen a hole, if I was at this location? So these values here, these are not the probability that I'm at that location, given that I saw a hole, it's the probability that I would see a hole, given that I was at that location. So the reason that it's highest in the middle is because even if I've got a wiggly stick over here, the likelihood that I see a hole is higher than if I'm sort of at the edge of the hole, right, because then I might sometimes run into the wall, sometimes run into the hole, and that's why I have this nice tight little distribution like that. Okay? So remember, that's not p of x given the hole, it's the probability that I would see a hole given that I was an x. Then what do we do? Well, in a particle filter we just multiply p of z, given x, probability the hole given x, times my prior distribution for everyone of these xs, okay? So that's what this distribution is down here, that is the multiplication of the prior times the likelihood, and in math, remember, the belief is proportional to the prediction, or my belief prior, that's this here, times my likelihood, that's this value here, and then I have to normalize the whole thing in order to get a new belief. Okay, that's why I'm doing that.

4 - Robot Motion

So, what happens on the next step, all right? Well, the first thing that happens is we resample. By the way, we're going to go through the exact algorithm of this in a little bit. To resample what we do is we first randomly pick a particle from this top distribution. This is the distribution we had after the multiplication, right? And of course the, the bigger the weight the more likely we'll pull out a particle, that's what these weights mean. So we're more likely to find things here than say over here for two reasons. One, there's just more particles over here, than over here, or to the right, than on the left. And also those particles have a higher weight. They have a bigger value. So I sample the particles, and every new particle that it creates starts off with a uniform weight. So I just, let's say I need 1000 particles, 1000 times, I, I might get the same particle more than once, that's okay. Then what I do is I, I sample and I move. Now, what do you mean, move? Well, remember those actions, the use, okay? So if I, if the robot thinks it took a step two meters to the right, okay? It shifts that particle by two meters. It shifts its belief of where it is two meters. And then finally, it adds a little bit of noise. This is just like we were doing with the common, remember? You start off with a distribution. You do the deterministic shift, and then you add some noise. Well our, to do our distribution, we first have to sample our particles, then do the deterministic shift, then we, add a little bit of uncertainty to that. Okay, so here you can see the resulting thing is we have a density that has been resampled, so there's like a whole bunch of points. And shifted with a little uncertainty added in.

5 - Next Sensor Reading

Now, I take another sensor reading. By the way, I just copied over, see the low, the density on the bottom. I just copied over, that's the belief that the robot has. The robot sees another hole. So that means p of z given x is the same thing I had before, because I got the same reading I had before. I saw a hole. And again, I multiply. But this time, I multiply the previous density by this and you'll notice now that the robot has a lot of belief, okay? That it's actually where it really is. And the reason is, is it saw a hole, it took a meter, it took a step of two meters, saw another hole. Megan's really hoping, I don't go another two meters. I won't and just that amount of information is enough for it to have a pretty good belief about where it is. Okay? There's still some probability that it's over here, because if that previous reading was just wrong and the current reading was right that I might be there, same over here. If the current reading was right, but the previous reading was wrong. No, the other way around. In, anyway, the, the idea's that it's, it's weighting these different probabilities. Okay? And you could, you know, do it again. So you resample and you move and you see here, we have the resample density where most of its belief is in the right place and there's some densities there and that's it. And that's the way a particle filter works, okay?

6 - Particle Filter Algorithm

Algorithmically it's pretty straightforward to do this. All right. We start off at any point in time we have a set of particles at the last time step and each particle has a state, like its position and a weight. And we're also going to be given a new action, like let's take a step to the meet, to the right two meters, and we're going to take a new measurement. And then here's how the particle filter runs. We start off with a new set of particles, empty and a normalization constant of 0, makes sense. And then we just do the following n times, where n is the number of particles that we need, all right? So, the first thing we do is we sample a particle from the previous distribution that's from this distribution here, according to its weight. Okay, so we just sample a particle, okay. The next thing we do, this is actually a combined step, okay. It says control, but it really should say control and control and diffusion, right? We sample a new state using two things. Whatever the old one was, so that's the value we pulled from the particle, right? That's this xt here. We've sampled the particle. It has a particular state that goes right in there. We also have the action. And then, we have to sample from the new distribution. So, let's suppose the action is taking a step immediately to the right. Every time we do this, make it go all right? Take a step meter to the right and then, I'm going to sample from a distribution because there's actually some uncertainty about that step, right? That's our process noise, so it's a little tight Gaussian. I generated a, a little bit of noise and I add that. That's what this sampling value is here. Using the previous sample, the control, and then whatever the noise is, okay? Then what I do is I re-weight that sample. What do I weight it by? I weight it by how likely would the measurement have been, this is my likelihood, okay? How likely would that measure have been if that new sample is actually where the the object was or where the robot was? That becomes my new weight. All right. That's the multiplication step we did before. We took our prior, we had those lumps, we multiplied and we got them through. We just keep track of the sum of those weights, we add that particle to our growing set of particles, and we do this n times. Now we're all done, except one problem, our weights don't sum to what? They don't sum to 1 because they were whatever they were. We just did this multiplication. So all we have to do is because we've been keeping track of the sum of the sum of the weights, that's theta. We divide all the weights by that value. And that's normalizing the weights. When we're done, we have a new distribution, bunch of particles. Each particle is at a particular state. Each particle has a weight. And the sum of those weights is 1. And so that's a valid, sampling-based representation of the probabilistic density. And that's it. That's all of particle filtering.

7 - Graphical Steps

Just stepping through this diagram, okay? We start off with some underlying distribution, pardon my inability to trace, okay? But, of course, we don't actually have a real distribution. What we have

is a set of particles, okay? Those particles are not uniformly weighted, remember? The sum of all the weights is one, some particles have more weight than others. That is, they, that particle has a higher probability. All right? So here's what we do. The first thing we do is we draw a set of samples from this set, okay? Now, you'll notice here, in this nice picture, one of my big particles happen to gets, to get sampled three times, the medium particles got sampled twice. And the this one got sampled once. And by the way, notice, how many times that these itty-bitty particles get sampled? Not at all. That's okay, because you're doing a random sample because it corresponds to the density, and that random sampling sometimes gets all the particles, sometimes only gets the ones with significant weights. So, we do the sampling. Now, the next thing you'll see is that each one of these particles has been shifted, but you'll notice that the shifts are not always the same. So, why do you think that is? Do you know why that is? There's a deterministic part, nothing says that my deterministic part has to be the same for all my guesses of the previous state, okay? Let's suppose what I do is I take a step proportional to the number of lines that are on the floor, right? So I'm going to step three lines. But if the lines are closer here than they were over here. Relax, Megan. Then going three step, three lines over here would actually mean I'd moved a smaller amount than if I was over here. If I had gone three lines, I would have gone a further amount. All this means is that the deterministic part doesn't have to be the same everywhere. It's just deterministic. Then, we add in essentially what could be thought of as the diffusion part or the uncertainty part. Remember when we were generating the new sample, we did the deterministic part, plus the probabilistic part. And that's what's here, right? So, after we take a sample and we move it, we add some noise. And we do that every time we have drawn a sample. So since we did three samples from this particle, it ended up at three different places. Likewise for all the others. Then what we do is we take our measurement, and we've got p of z given x. That's what this value is. And we just multiply these weights, which are all 1 to start with, by that value, okay? Oh, here it is. p of z given X. We just multiply it, and we end up with this new set of particles, okay? So if I erase my, my figures there you take a look, that's a beautiful figure. It's all of particle filtering. Just sleep with it under your pillow and you'll be in, in great shape.

8 - End

So, that's all there is to the algorithm, that's particle filtering using sampling as a, as a way of representing your densities. There are some details to mention about how to do that, which we'll talk about later, when we do if for tracking. But, before we do that, I want to show you some interesting, sort of, robot vision applications be, because they they illustrate the use of particle filters really well. And then we'll revisit the pure vision tracking but in the meantime I'm thirsty. I'm going to go get a drink so we'll be back for the next lesson in a bit.

7C-L3 Particle filters for localization

1 - Intro

All right. Welcome everybody. Hope you're having a great day. Back to computer vision and were continuing with our work and particle filters. Last time we introduced particle filtering and we talked about the algorithm that actually implemented it. So were using a set of particles representing density. And we went through the base loop of essentially propagating the belief forward through about the deterministic and non-deterministic part. Then we get a measurement which gives us a likelihood. We multiply the, the prediction by that likelihood and that gives us something in Bay's rule that we can normalize to have a new belief, a new posterior estimate of the state. So, in this lesson, we're going to look at some, robot vision examples that, sort of illustrate nicely, sort of, the simplicity of using this method.

2 - Proximity Sensor Model

Let's take a very simple robot sensing problem, right? So let's assume that a robot knows the 3D map of its world, okay? And for now, we're just going to worry about a robot running around on the floor. So it's not climbing stairs, instead it's just running around. And let's assume also that it has noisy depth sensors, so it can measure depth in a lot of different directions. And they're noisy, so the, the measurement is uncertain, but we know about the uncertainty. That is we have a sensor model. Okay? And it moves between frames, and it takes a, a set of readings, and then it moves again and takes a set of readings, and it moves again. But the idea is it keeps taking all these readings, and the question is, how well can it know its state? So, what is its state? Montana?. No. Its state is three values, x, y and theta, its orientation. Okay? That's why we're just doing a, a multiple robot on the ground three dimensions. All right? So, just to remind you, on a Ba, Bayesian filter framework, we need four things. We need some sort of the prior, remember that trip to Delphi, so you unpack your next prior. We need a dynamical system model and that's the likelihood of landing in some new place, given that you're in an old place plus whatever action was taken. We need a sensor model, remember, a likelihood that's the, not the probability of being someplace given a measurement. The likelihood is the probability of getting the measurement given what your state was, and then we need this stream of observations and the action data, the Zs and the Us. For that robot that I described, the state is obvious. The map will be easy, et cetera. The thing that's tricky, not tricky, the thing we have to define, is the sensor model. So here's a picture of a sensor model, and what this is is, this is the probability of getting a particular measurement if, let's say, the actual state was here on this black line, okay? And, what was nice is when your using a real robot, you can actually measure the likelihood, so you, you, you stick the sensor, you know, three meters away, or, let's see, it's at centimeters, so 200, so 2.3 centimeters away from the wall. You turn it on, you turn it on, you turn. You collect a whole bunch of measurements and you see what you get. That distribution would be this distribution. Okay? And in fact, the blue is what was measured. All right? So that's why the blue's got the noisy stuff. And the red is how they approximated it. And you can see they basically had some sort of a Gaussian function or something like that. In fact, you can see that the laser has a less noise than the sonar, okay? So, this is, this is actually a relatively noisy laser. A modern laser would be, you know, very precise. Except that you might have some specular problem, et cetera. So all of that would be fine, except, you'll notice there's this thing here and this thing there, and the question is, what's going on when, you know, the thing is 2.3 meters away and here it says 500. But, the astute observer will notice you're just using the very last value. Well what that really is, is that's the probability that you get no return, okay? So you're some distance from the wall, you send out your sonar chirp, or you send out your laser, and it just goes away. Gets on a train, never comes back. This happens you guys know what specular reflections are, right? So when light bounces off to, turns out that sheet rock is very specular to sonar. So when you send out a sonar chirp, often it doesn't come back, it skips like it reflects off, in which case, you don't ever hear the chirp come back. Or it comes back after having gone through a long echo path. Likewise, if you're looking at something that's shiny, like a glass wall. When you point at it, you can see the laser beam. But when you point it on the side, okay? It'll be reflective. It'll be a specular. So what these values are, is the probability of getting no return given that you're at some distance. And in fact, that probability stays the same pretty much no matter what your distance is. All right? So remember, this is not the probability that your distance is actually 500. It's the probability that you would get this no return at any given point in time, okay? So we need to incorporate that noisy model.

3 - Sample Based Localization

Here we have an example. And, what the robot is actually going to do, is it's going to follow this purple line. Okay? So this is where it starts and then it goes here, and it moves that way. What we're showing you to begin with is a couple of things. First of all, you see all those red dots? That's its current set of particles. In other words, it has no idea where it actually is. The green is an example of where it thinks it might be, in fact, this is the locally most likely. Which is just some

way of the robot guessing where it is. Now, what I'm not showing on this map is the, for each particle there's also a theta, right? Particles have to represent the state. What do we say our state was? We said it was x, y, and theta. It's hard to show this in three, so I'm only showing x and y, all right? But, for any given theta that I think it is, then I would be sending out a ray of sensor measurements. And I think I know where it should hit because why? Because I have a complete map. Okay? So what happens is, first, you're going to, your eye's just going to naturally go to the green thing. What's going on there is that's the set of sensor measurements that it's getting. Though, so wherever, wherever it thinks it is, it puts down the measurements it actually got. Now, if it's in the right place, those measurements will actually, most of the time, hit the wall at the right distance. Every now and then it'll go right through the wall because it was a no return, all right? So let me just put it in motion. It's easier to, to understand. All right, so here we go, so it's a movie. So each one of these changes, is when it thinks it is when it takes a measurement. And you'll notice, there's a clump of particles here, in fact that's the most likely clump, and there's also a clump of particles there. That's a nice way of saying the robot does not know whether it's definitely here or definitely there. And the reason for that is the hallway's kind of symmetric, right? There are these doors, okay, but there's doors on both sides. So it hasn't, yet been able to disambiguate. If this is the center of my universe, whether it's at this area or that area. But let me put it back in motion. What's going to happen is the robot's going to go inside, in the room. And you'll notice in this room, in the room it's about to go into, there's a ottoman or whatever that is in the middle that's not in the other room. So when it, once it goes in, yep. There it goes. Okay? You'll notice the particles that are in the wrong place, they're going to disappear. Okay. Now some of the rays are still going in the wrong direction because, they're, they're going through the walls, but that's just the no return. And you'll notice that, by the time we get to the end of the run, you'll notice that it has localized itself very well. Some of the measurements still go through the walls because I'm assuming those were no returns. But, just by taking these simple one-dimensional measurements, several of them at a time, and knowing the map and taking its movements, it's able to localize exactly where it is. And that's sort of a hallmark of particle filtering. You start off with a big unknown set, and eventually a whole bunch of particles go away because they're just totally inconsistent with the measurements you get. So when they get multiplied by p of z given x, the likelihood of that measurement is so small that it diminishes that particles weight a lot, and the next time you sample, let me say this again, remember very low weight particles sometimes don't get sampled at all. So what happens is only the samples that are near the parts of the sensor measurements that are consistent survive and you ke, you keep iterating. Also remember we had the two lumps for awhile there because it wasn't sure whether it was this one or that one. Because it didn't have enough information to know yet. It wasn't until it went inside that room that it had that information. And this is sort of the inherent advantage of particle filtering over say, a Kalman filter that has a single hypothesis. For those of you who have maybe heard of multi-hypothesis tracking where you, where you end up having multiple hypothesis. One way of thinking a particle filter is, is, it gives me just a uniform and easy way of doing multiple hypothesis tracking.

4 - Smitsonian Museum of American History Example

Meanwhile at the Smithsonian Museum of American history, which is just a bigger map, bigger thing. We're going to do the same thing. One thing I'll show you here is that here the map is a little bit bigger. And in fact if you take a look at this map, it was actually created by a robot doing what's called Slam. Where we actually use the robot to build the map. So here we have again, the robot doesn't know where it is. It takes a bunch of measurements. These are where the actual measurements are. The robot doesn't know that yet. And this is the set of particles that survive. So you see it's clumped up a little but it doesn't know. So it moves, takes another set of measurements, and these are the particles that survive. Now with just two sets of measurements. Almost everything else here has gone away, and we're getting a bigger clump there. Just two sets of those measurements. Right? Propagate forward, take a measurement, and now you can see, as it's advanced forward, it has pretty much very little ambiguity left. So again, just by taking these simple measurements, it knows where it is.

5 - Using Ceiling Maps for Localization

How about some very simple vision. Real so we're not using a depth sensor anymore. And this is about as simple a vision as, as you can imagine. So this is work that was done by Frank Dellaert. Who's a professor now at Georgia Tech, in our robotics and vision group. But in 1997, he was like six years old it looks like. No that's not Frank, that's some kid at the museum. Frank, Frank was doing his PhD at the time. He did something very cool, they said you know, we've got this ceiling and this ceiling has lights in it. If we had a map of the lights we should eventually be able to figure out where we are by just do we see light or don't we see light? And the way that works is you had a robot with a little camera that just looked up, all right, here's that little square. And it would just see, did I see something bright, medium, or sort of dark? Right? It would, it could just get a small number of sensor values, or you could think of it as just taking the total value as saying about how bright is that spot? Now to do that what I have to be able to do is given being anywhere in here, I have to know what is the probability of getting that brightness in the image. So, that's what this shows here. That, if this was actually the brightness, this is the probability of the brightness that I would see. So the idea being if, if this is the actual brightness of the state x of the position, then this is the, the likelihood of, of what I would see. So for example, suppose I'm under a light, okay? So typically what I see is this brightish patch here, and this map is the likelihood of getting that bright measurement at these locations, and, and by the way in this map, the black are the high points. So you see each one of these lights? So if I was actually under the lights, the likelihood of seeing something bright is high. Otherwise, the likelihood of seeing something bright is low. Sort of near a light, I'm going to see the, tend to see these gray things. And when I'm far away from a light, I'm going to see mostly darkness. So those are, are p of z's given x. And then, here's the thing running, And it's a little hard to see, you see all these red dots going? And what's happening is the robot is actually following this red line. And it comes in, and it goes down. And you'll notice, after a little while, it knows exactly where it is. Now it's going to turn off to the, to the right over here somewhere. And you see. If it hangs out in a dark area for a while, it's uncertainty grows because it doesn't know exactly where it's moving in that dark, in that dark area, okay? So you're watching it and it's doing some cool stuff. What's interesting is, so we have this map of the thing. We can compare two things. The robot knows the commands that it gave to its wheels. Okay? So it knows how far it thinks it went. It also knows how much it thinks it turned. You guys have an odometer in your car? The odometer measures distance. Odometry is when you measure how you think you move based upon how your wheels turned. If we plot on this map where the robot thinks it went based upon it's wheels turning, we see something that looks like this, okay? And the reason we know that that's wrong Is that these are areas that it can't actually go through. And you see it gets, sort of, right, but then it, sort of, gets a little confused, and it's going through parts that it cant' actually go through. When we add vision, and why do I say add vision? Well, we still use the odometry as the deterministic part. Make him jump for that thing, right? So, the odometry says, well, you moved a foot to the right. Okay? So, we add that. That's our prediction part, so we still use the odometry, but then the measurement from the vision improves our result. Okay? And that gets us the much better result. And this is using essentially a camera just looking at a little area and saying am I bright or am I dark or how, how bright is it and I have a map of the ceiling. So it's pretty cool. Well done, Frank.

6 - Puzzled Ant Quiz

So you still don't believe all this particle filter mumbo jumbo? Let's try a simple example. Here's an ant scurrying along on top of this crossword puzzle. It's one of those computer-sciencey ants, so it only moves one step at a time, either horizontally or vertically. It can sense whether it's on a black square or white square. But that is it, it has no other way of knowing where on the puzzle it is. Here is a sequence of steps that the ant took. The first column is time, t. The second column is its observation, z, which is one for white, and zero for black. The final column is that action taken, or u, which is basically what direction did the ant move in. For instance, at t equals 0, the ant saw that it was on a white square, and then it moved up. At t equals 1, it observed a black square. This was

the result of moving up in the previous step. Okay, the next action was moving left, and so on. Can you figure out where the ant is at t equals 5? Mark the appropriate checkbox, or boxes, if you're still unsure.

7 - Puzzled Ant Solution

Okay. Let's see how a very simplified particle filter can help us estimate the ant's position. Initially, at t equals 0, we have no idea where the ant is. So we might as well assume that it is in one of these squares, any one of them. This is our initial prediction. Now the observation says that we are on a white square. The likelihood of us getting a white observation when we are on a black square is really low. So, in our simple case, we get rid of the particles In the black squares. You can think of this as a correction. Okay. Now we see that the action taken was move up. So we take our entire distribution of particles and we shift the whole thing up, like that. Any particles that went outside the valid area are discarded. Now this is our best guest or prediction at time t equals 1. This time, we see that the observation is black. So, all the particles on white squares are basically inconsistent with the observation. As before, we get rid of them. All right. Our next step at t equals 1 is to move left. Observation at time two is white, so we get rid of all the black particles. Next, we move up. Interestingly, all the particles on the board are consistent with the observation. At step three, we move up again. Now the observation at step four is black, so we get rid of the white particles. Now we're left with only three. Well, from here, we move right and the observation is white, so we get rid of the one black particle and that's it. We're left with two possible positions for the ant. Since we don't have any further information, that is our best guess. Are you wondering where the ant's started? Well, it could have started here at t equals 0. Move up, see black. Move to the left, see white. Move up again, white. Move up, black. Move to the right and you see white, but that's one possible path. The other starting point could have been down at the bottom here, same thing. You move up, black. Left, white. Up, white. Up, black. Right, white. So particle fillers, useful aren't they?

8 - Resampling Method Can Matter

Before ending this lesson, I want to talk about a couple of practical considerations of particle filters, and the first one involves sampling. As you can imagine, sampling is really important. Okay? Because I have this set of samples, and every step I have to generate a new set of samples. So I have to re-sample it. I go, grab a sample, where the probability of pulling out that sample, for its state x, is proportional to the weight of that sample. And typically, I'm doing this a constant amount of times, so n times, so I'm using 1,000 samples, 10,000, however many samples, I'm doing that every time. We'll talk actually not today but next time, that number can get kind of big if your state space gets large, so being able to sample effectively or efficiently, or less often, is kind of important. So I'll talk about resampling a little bit. What we have here is a, an outer ring, okay, where the idea is that the arc length of that ring is proportional to the weight. Okay, so, you know, w1 is small, w2 is bigger, w3 is huge, etc., whatever. And these are all the weights, right? So you can think of sampling as the following, I just randomly pick a direction, and if I hit w3, I'm going to take sample number three. So that'd be my first sample. All right, do it again! If I hit wn, I pull out the nth sample, and basically I have to keep doing that n times, right? So, it's kind of, it's like a roulette wheel. I spin the roulette wheel n times, and I have to figure out where I land, and that figuring out is actually a binary search, so that's log n, but I do that n times. So it's an n log n operation. So that's kind of it's not awful, but it doesn't scale linearly with the number of, of samples you use, it goes up by n log n. It turns out there's a cute little trick that you can use that is, instead of like a roulette wheel, is more like a wagon wheel, and the way it works is this. So let's suppose I have n samples. I can make a set of spokes that are 1 over n full rotation around. So here I've got 1, 2, 3, 4, 5, 6, 7, 8, right? So each one is an eighth of an arc. And what I can do is, I can randomly put it down someplace, and if I've randomly done it, then I can just read off, for each one of these spokes, which w did it hit? Okay? And in fact, the starting way, right, I can just start at some random one here, and I see what w am I at, and I go to the next one and I see, did I pass a boundary? If so, then

I'm at the next w, and likewise. So, what's nice about that is that's done in constant n time. And it's sometimes called stochastic universal sampling, systematic resampling. The nice thing about it is that it's a linear time complexity. You know if you do 10,000 points, it takes you so long. If you do 20,000 points, it takes you just twice as long, all right? And it's very easy to implement. In fact so easy to implement, let me show it to you.

9 - Systematic Resampling Algorithm

So here I'm showing you that actual wagon-wheel algorithm that we just talked about, this systematic resampling. You start off with no new samples, and you build your, the c's are going to represent the outer ring, the cumulative distribution function. And, essentially, each one represents the one before it plus the extra new weight. Right? So, you start off at the first weight, and then c2 is the sum of w1 and w2. C3 is the sum of w1, w2 and w3. That's that outer ring, okay? You can think of it, that if it goes from zero to one, because everything sums to one, then the c's are sort of how far, what percentage around the clock have you gone? The next thing you do is you have to pick, remember I told you, you, you have to sort of randomly throw down the wagon-wheel to start with. Well if the wagon wheel has eight spokes, the first spoke has to go somewhere between zero and one-eighth. That's what this is right here, right? N to the minus 1, 1, remember you could write it as 1 over n. That's the offset. And then what you do is you go around the outer ring seeing, wait, did I pass my offset yet, as soon as I've passed it, that's the cdf part that I'm in. So just drawing that on here, if this my first one, right? And I, and, and this is where it starts on the outer ring I say, did I pass it yet? I say, nope. 'kay, did I pass it yet? Yes, I did. So this first spoke landed in w2. Then I just go to the next spoke, and I say, did my last marker pass it? No, I gotta go to the next one. Did this pass it? Yes, so my next one is from w3. So that's all I do, is I, I initialize it and then I just count up for however many samples that I need, I go and I grab the next wagon-wheel chunk and I see where on the cdf that it lands. And then I just return that set. It, it looks more complicated than it is and it's a way of doing that sampling in, in linear time.

10 - Particle Filters Practical Considerations

That's sort of how to do sampling efficiently. The other thing is, how often do you have to resample? Well, in the algorithm that I showed you, we resample all the time, and that's sort of we have a whole bunch of samples that are about uniform weight, and we spread them out. If I don't have too many samples that have very large weights, maybe I don't have to resample. Maybe I could just reuse the weights the same samples over again. Right? I'd still have to apply the deterministic part, and I have to apply the expansion part, but I don't have to do that resampling. And you can do that by just saying, let me take a look at sort of the variation in the weights. I want them as uniform as possible. And as long as they have a certain amount of uniformity to them, I won't resample. So that reduces the number of times you do resampling. A second problem is what if p of z given x. What if your likelihood is incredibly peaked. Remember, you multiply the likelihood times the weights of the samples. If it's incredibly peaked, that means that p of z given x is zero almost everywhere else. That's not very helpful, because what that means is, you've wiped out all of your your, your possible predictions. In general, you want to allow for noise in your p of z given x. Don't be so systematic about it. And you also might want to be better at sort of how you do your your proposals to begin with, right? So you want to make sure that particle filter sort of spreads out to points where to begin with. And one way of doing that actually is if you think of a sample, each sample you can think of as having its own little mini Kalman filter. So you can think of it as having a Gaussian that sort of spreads itself out and goes forward. In general, you're much better off overestimating noise and let your measurements sort of narrow you down, than underestimating noise and have your measurements think that they're much more accurate than they are. Or I should say much more constraining than they are. Another issue is recovery from failure. Suppose an object really did disappear and reappear. If I have no particles there, I don't have any way of tracking it, ever. So, a standard thing to do is each iteration, you randomly put out some particles everywhere, or sort of uniformly distribute it. Because if my measurements suddenly have

moved over here, there'll be some particles over there, and eventually that will be the distribution that will be tracked. So those three issues of sort of making sure you're smart and efficient about your resampling. Not letting your system think that measurements are too precise, and making sure you have enough noise so that you don't kill off samples too quickly. And this idea of having some random samples distributed in a, in a useful way, so you can track things if things change very unexpectedly. All these things are necessary to make particle filtering work.

11 - End

That ends the robot vision part of particle filtering. Now if you remember, we introduced all of these stuff in order to be able to do tracking. Okay, we started with common filter tracking, then we did particle filter. So, in our next lesson, what we're going to do is, we're going to go back to doing tracking. And we'll use some simple particle filter technique for doing some visual tracking.

7C-L4 Particle filters for real

1 - Intro

All right, welcome back to, Computer Vision. Today what we're going to do is, we're going to, finish up, particle filter, particle filtering and we're going to do, sort of do particle filtering for real. We mostly enter, discussed it from an algorithmic perspective and sort of how it maintains density estimate. And we even looked at some robot vision examples last time. You know, localization through the Smithsonian and, and other things. And this time, we'll talk a little bit about real vision and how you might do it. So if you remember this Bayesian filter framework. We had to have sort of four different elements, right? We needed a prior, which nobody ever talks about. It's kind of the black sheep of the family. A dynamical model or an action model that says how we think things change from moment to moment, based upon where they were and the inputs that are provided. A sensor model, the likelihood of a measurement given a state. And then we have this stream of observations, and the question is what to do with them?

2 - Real Tracking

To do real tracking, we have to specify these things. In particular, we've got state and state dynamics and we've got a sensor model. So in the math, it's easy. We just write x and we say, x is the state. But the state of what? Montana, what, Missouri, you know, what? No. Is state of the object. Everybody says, oh, it's state of the object? Well, what does that mean? It's obviously some representation of the object. Its position, its velocity. Maybe something about whether it's happy or sad and that sounds like a joke. But actually, if you can think about tracking the state of the object, there's no requirement that the state be a purely physical thing. It's anything that can have an influence on a sensor measurement that you want to know. Speaking of sensor measurements or speaking of z. That's the measurement, but the measurement of what? What measurement are you going to use and how does the measurement relate to the state? So, if we're going to do real tracking we have to know these things. And finally, where do you get your dynamics from? Do you go to the, the, the, the dynamics shelf at Home Depot and say, I, l need some dynamics please? You know, where are you going to get these for computer vision? The discussion we're going to use today comes a bit from this paper and, and the idea of using particles and sample based representations of densities, that wasn't new. That was known in a variety of mathematics and, and other engineering fields. But this is this is the paper that really brought it to computer vision.

3 - Particle Filter Tracking State

We showed this picture already, right? This is a picture of the tracking of Andrew Black's daughter. And, the first question is what is the state? All right. And the state, or the object that was to be

tracked, was a hand initialized contour of the head. On the left here you see a bunch of contours. Okay? In fact, there's even one that's up here, all right? Those contours are the contours represented by each or many of the particles, the top particles, right? And so, for example, you can actually talk about the mean state as being the sort of average of the particles. But that doesn't really tell you what it is. What do mean, you know, the state of a contour? Well, in fact, what they did is they, they drew a contour, and they said, because that's not a parameter, that's a shape. And they said, let's represent the state of that contour by an affine deformation, okay? Remember affine? Affine is, it's a three point to three point mapping, takes how many parameters? It's six parameters, okay? Translation, rotation, scale, and sheer, depending upon how you count those, I mean, it's six and you can sort of divide them up any way. So the idea is that each particle represents a six dimensional vector. So, by the way, that means our particles are living in a six dimensional space, and the bigger the, the number of dimensions the more particles you need to sort of fill up that space, in fact it doesn't grow so nicely, does it, right? I mean, if I go from two dimensions to three dimensions right, if I had a 100 particles before, if I wanted to have the same density I would now need a 1,000 particles, all right? I goes ec, it grows exponentially in terms of the number of dimensions. So for affine we're going to need six. And as I said, each of the particles represents that. Here by the way is one of the videos. And one of the questions you might ask is, well where do you get the dynamics? Where do you get the dynamics? Well, there are a couple of things you can do. The easiest thing to do is to cheat. All right? So the reason we're using dynamics here is that the girl is jumping up and down in front of a cluttered background. So to keep track of where her head is, is kind of a difficult tracking problem, that's why we're doing all this work to begin with, that's why we're using particles. But when would tracking be easy? How about if she was jumping around in front of a green screen, or a white background, okay? Or in fact, even easier, suppose you put her in a totally black turtle neck sweater, and you put her in front of a black screen, kind of like that thing. By the way, Megan doesn't let me wear black shirts because I'll disappear. But the point of my disappearing is all you would see would be my floating head, which would be kind of creepy, but it would very easy tracking. So you could have somebody jump around with their floating head, track it very easily, and measure the dynamics. So one way of getting dynamics is to sort of cheat and get it from an easier system. And I think for a bunch of the stuff they do in cassation, that's what they did. They had an easier problem.

4 - More Complex State

Something else they did, was they tracked the state of the hand. So, what do you mean, the state of the hand? That's actually, you, I, I encourage you to read the paper. They did a couple of really cool things. First of all, you have Translation. Okay? Well, how many degrees of freedom in translation? Well, we're doing it in two dimensions, so there's two degrees of freedom. They had a Rotation. Okay. Which is one degree of freedom. But the other thing they added to that was the shape of the hand. And you might ask, well how did they encode the shape of the hand. Don't bother Megan. Well actually what they did was they took a whole bunch of images of contours. And then they did a form of principle components analysis. Now you may not know what that is. But the good news is you will. Very soon we're going to do principle corner analysis when we talk about, recognition. The thing that matters is that it has a low dimensional representation of, certain variation. And what they did is they extracted out essentially, instead of keeping rotation separately, they took ten degrees of freedom that represented the shape of the hand including rotation. So you had two for translation and ten for rot, for the, for the rotation and shape. So, that's a 12 degree of freedom of space. So that's even more difficult to do the, tracking with particles because you have to have a lot of particles in order to do this.

5 - Particle Filter Tracking Measurement

Okay. That's about the state. What about the measurement? Okay. What about z? All right. Here's another picture taken from that paper. And what they did was in fact, if i go back to the hand thing, you'll see that the hand, what they want to do is track this contour, all right? So, I'll draw it on the

hand part. Well, if you've got the right state, then you would expect to have an edge in the image at the edge of where the state says the fingers are. That is there some relationship between the edges of the fingers in your state and the edges that you would see in your picture. And in fact, that's what they do. Okay? What they do is and there's more derivation. Basically, they come up with a probability distribution that says that the probability of getting a particular measurement given the state. And remember, the state tells me where the fingers are. Is essentially an exponential, so like a Gaussian that is proportional to the square of the distance to the nearest strong edge.

6 - More Tracking Contours

So that was the original condensation paper. Like I said, it was in 98. There have been many versions subsequent things that have to do with importance sampling, that look at the particles and stuff. Here's another example of doing head tracking, 2002. And you can see, it's still a hard white right there, tracking the head there, kay? There's an interesting example right here. You see the head here is fully visible, but then it goes behind the cabinet. In fact, the only part of the head that's visible I just outlined, in fact, only the top half is visible there. So the question is, how did they move through occlusion? Okay? because part of the head is missing. Well, maybe they used velocity in their state or maybe they didn't. What do you think? You think they did or you think they didn't? Personally, I would have thought they did because, I mean, I know all about common filter and I know if you track velocity it's real easy, but then if I think about particle filters, I say I probably didn't need to track velocity for the following reason. It's not the case that it has to be a great example of a head contour. It has to be the case that the example of the head contour that's here is a better example than if I put the head somewhere else. And in fact, indeed, this paper, they track the head through the, the partial occlusion without velocity. Now, would it have worked if the head was totally occluded behind the cabinet? No. Because then the thing would just stay there in fact they, they call it a velocity zero model. The particles just stay where they are and they just defuse out from the uncertainty. And so if you showed up on the other side of the cabinet, you'd be out of the frame of view altogether. Yes, Megan, I'm back, okay. Where as, if you did velocity, you would follow me, follow me, there you go, follow, very, very good. So now you get the, you get the point.

7 - An Even Better Model

You can do interesting things with States. So, here we have the, the Head Contour. Another kind of model is, suppose you've got some way of detecting colored blobs that represent say, Hands and Head. Now your state would, might be the x, y location of the hands and head. And your measurement might be where you think the colored blobs are, and your sensor model might be how well do predicted blobs match the color. Your, you can use sort of any model you want as long as you can have a state and you can talk about how that state expresses itself. In fact, here's an even better model. Suppose you want to track some region of, of the distribution of colors. Right? So like maybe this region here or this region here where you can see the ellipse, that kind of gives away that the region that we're going to, we're going to be tracking here. What would be a good model or state for tracking that? Okay. Well, clearly you want the location, because after all I want to know where the thing is. All right? And if I've defined a region, I want to know, sort of, how that region is changing maybe a little bit in orientation or size. But just to tell me this, the position and the size, that's not enough. I need something about the appearance. In this particular method, and we're going to talk a lot more about this next time. What they tracked is really cool. They tracked the distribution of colors. Right? So if I look over, let's say. I'm not going to do that region, I'm going to do this region over here. All right. There are some white pixels, there are some reddish pixels, there are some brownish pixels. And as that player moves, the shape will change in the form etc. But the overall distribution of colors might be quite similar. All right? And in fact what you have to say is well, what's a reasonable sensor model and we're going to get to that next time. But, basically what you're going to do is you're going to compute the similarity of the color distribution of what you had, and the color distribution of your proposed state. All right? And you'll need

somewhere of comparing those color distributions. And just as a way of showing you how well this thing works, this, here's two examples. This is actually done using mean-shift filtering, but could also have been done, using particle filtering, and we'll talk about that, both, next time.

8 - How About a Really, Really Simple Model

How about a really, really simple model? Would you like a really simple model? Well, of course, you would because who wants a complicated model if you can have a simple model? All right. Suppose we just start off with a patch, a location at some point: x, y. All right. So, we're going to say, all right. I've got some patch at x, y, and my dynamics is going to be random noise. So what that means is, is that's the same model that they use for the head contour. We're not going to track velocity, we're just going to assume that the position changes randomly. Okay, so as you add your particles you assume zero velocity so they stay where they are and you would add random noise. And then, how about the very simplest of sensor models. Okay, you just take the mean square difference, right? So what this says is if I propose that the state of the object is here, and it, it used to be here, I'm just going to compare the patch. I'm literally going to subtract the two patches, and say what's the mean squared error. Okay? That would be, sort of the simplest sensor model possible. You could do a slightly more sophisticated one, right? You could do a normalized correlation. You've already learned about this when we did stereo. And you could say the higher the correlation the more likely or the bigger the sum of the square differences the less likely. That would be your simp, sensor model. So to this, all you would need is a patch. So there's a patch. So those of you who are from the United States or know U.S. politics, that's Mitt Romney. The reason I'm showing you this is that we provided to the students who are going to be doing problem sets in, in the original class I'm going and the OMS class a video. Okay? And this is the patch, grabbed out of a section of one of the frames. And your goal is to see with a particle filter, how long can you track Mitt Romney until he until it falls off and he loses the election.

9 - End

That ends this section specifically on Particle Filtering. You've now seen the algorithm in some details. State and dynamics and censor models. And the really good thing about particle filters is that you can make them as complicated as you want. But you can start off simple and get something running. You then have to worry about how, how many dimensions are in your state space, because if you get too many, then you might have to have too many particles. You have to worry about re-initializing. We talked about that before, of scattering some particles around in order to you know, in case something has happened quickly that you didn't anticipate there would be some particles there. But the good news is it, it's really quite accessible, there's a lot of papers to read. You can actually start with that condensation paper. It, it really lays it out. In fact, there's a conference version of it, that came out even earlier that I think is even clearer because it has less detail. So so I hope most of you will get, will get to understand that. And certainly the OMS students, you will, as you will be doing it on the problem set.

7D-L1 Tracking considerations

1 - Intro

Welcome back to Computer Vision. Today finally, finally, finally we get to finish up on tracking. We've discussed tracking a few ways from the inference perspective where we have a dynamics model, and you know the whole bit now at this point. Where we predict forward the state, we take a measurement, and we do our correction. One was a Kalman filter where essentially we have a specific representation for our densities, namely Gaussians and all the world's a Gaussian. Then we discussed, particle filters where because we're using samples you can have much more arbitrary densities. You just have to have enough particles to represent them. And also you need a

generalized censor model which works well. For this last section just talk about a couple more methods. Just so you should see that they're being done. And then at the end, I'm going to talk about some issues that show up in tracking, no matter how you're doing it. And it's sort of outside of the main tracking conversation and yet, if you don't deal with those issues, nothing works.

2 - Mean Shift in Space

Mean-shift is a method that's easiest introduced, when we do segmentation. The idea is, in Mean-shift, is to find the modes of a distribution, or of a probability density. So, you remember, the modes are sort of, the peaked areas, and here's the basic underlying assumption. The assumption is, I've got a bunch of samples, that have been drawn from some probability distribution function, and you want to find the modes. So, there's this beautiful couple of beautiful examples out there on the Web, feel free to steal whichever one you want, I stole this one. So, here we have a set of particles, and they're just drawn from some density. And probably, if I were to ask you where's the mode? You'd probably, say, that it's somewhere around here. 'Kay, and the goal is for the system to find that on it's own. So, here we have, and again, we have our sample, so let's put down a region, okay? It's called a Region of interest. Now, we're not going to talk about, the size of that region just yet, or its shape. We'll talk about that, a little bit later. It's going to be a Gaussian, but, for now, it's just going to be a circle. So, we have this region, and we calculate, okay, of all of the points in that circle, where is the mean, that's, or the center of mass? And we say, oh, it's not right in the middle, and that difference between them, that's called the Mean Shift vector, so this is called the Mean Shift algorithm. Amazing huh? And so, what do you do with the Mean Shift vector? Well, you shift the mean, and there it go. [NOISE] Perfect. Okay? What do we do? We do it again. Okay? Here's where we are. Here's the, the weighted mean of the, the new samples. There's our Mean Shift vector. What do we do? We shift it again. [NOISE] And we do it one more time. There it is, oh, we're getting close now. Small Mean Shift vector, shift, and then finally, guess what, we are in the right spot.

3 - Mean Shift Object Tracking

What's kind of cool about mean-shift, a bunch of things, is under some reasonable assumptions like the, the shape of the thing that you put down which is a kernel kind of thing, you can prove that, in most circumstances, this thing will converge. So, it'll actually find those modes. And that's really cool. So what does this have to do with tracking, you might ask. What does this have to do with tracking? So basically, mean-shift tracking is, instead of just tracking the mean of a distribution, we're going to try to track, essentially, a, something about a distribution of a region. I'll, I'll make that clear. But our methodology is pretty similar to before. We have some position that we start with, right? And here, by the way, our initial model is not just a location. But it's this region and the size of that region. And what we have to do is we search. We're going to search in a neighborhood in the next frame for that same description. Okay? And when we do the search, we're going to move around at x y. We also might do things like change the scale, all right? And what we want to do is, we want to find the lo, the new location, maybe scale, maybe some other ch, change. Maybe an orientation thing. That maximizes some similarity function. All right? And then we just, iterate. We connect the dots. And we repeat the process and we get this beautiful animation that says, okay, the new maximum becomes the start for the next frame.

4 - Representation

The thing that makes this mean shift tracking is essentially what the model is and how we compare them. So, we've got this initial area that we want to track and we have to choose what the model is going to be. And in mean shift what we do is we choose a feature space that is the quantized color space. So, what that means is that we carve up the different RGB values into a histogram. So it's written as a 1D histogram here, but it's typically a three dimensional histogram. R, G and B. You might carve R up into four bins, G up into four bins, B up into say four bins. That'd be how many

bins together? 4 times 4 times 4? 1,000,090, no. 64, okay, in which case you have a 64 bin histogram and at every point within that region you take its color and you put it in the right bin. And that would give you your probability histogram which you can think of as a PDF of the color over that region. That's the representation of the region that we're going to try to track.

5 - Similarity Function

Okay. So, here we have two frames. On the left we have the original model the, the model is sort of a time and, and we're going to call that zero. And on the right we have some candidate. Something, some place in terms of how its changed and this can be centered at some value y, some pixel y. So on the left, we have our initial distribution. Okay. So this was our histogram that we computed, in our area, and it's typically called q. And q's got a set of bins, u1 through m. And because it's going to represent a density, we enforce the fact that sum of them equals one. So how do we do that? Well we just sum them all up and we divide all the bins by that value. Okay. And you see we have the audaciousness to call it probability. Okay? Because well were going to take these samples and we're going to assume that it's representative of a probability distribution function. So that's our distribution at the initial of the target. At some new location. Okay. We can compute the distribution again. So now p of u is a function of y where y is the point that we're considering. Remember you have to consider p of z given the state. So y is going to be that, that new point. And it's called y instead of x, that's how it's done in the Mean-Shift Literature if you read this, this is, this is what you would see. And again, were going to build a distribution by assuming that these things sum up to one. So we have this two distributions and what we have to do is we need a similarity function that compares them, that compares the original q and the new p of y which is the density at this proposed place. So the question is how are we going to compare these distributions? Now there are a variety of functions that you can use to compare histograms, where the histograms, are, are normalized to one, so they represent probabilities. There's the min value. There's, there's chi-squared, a variety of them. The one that was used with respect to Mean-Shift tracking is referred to as the Bhattacharyya Coefficient. Bhattacharyya Coefficient comes from something related to the Bhattacharyya distance. It's simply a way of comparing distributions. What you do is, you take your distributions here, and you change it by taking the square root of each of the components. So, if q is a distribution, q1 through qm, then q prime is going to be the square root of q1 through the square root of qm. Same thing with the new proposed distribution, p, around y. We're going to take p prime, where we take the square roots. When we define it that way, the Bhattacharyya relationship is defined as follows. Okay? Just take the sum over the product of those square roots. Okay. Well you might ask, why would you do that. Why would you do that? Well you quickly realize that if you take the magnitude of those square rooted vectors. Right? The magnitude is the square root of the sum of the squares. Well, the sum of the squares is what? Well, since the sum of the qs is one, the sum of the square roots then squared is going to be one, so the magnitude is one. So, these magnitudes here, those are one, so that means you just have this. So basically you're computing the dot product, the cosine of the angle between the square roots of those distributions. And that sum of the square root, the thing in the box here, that's what's referred to as the Bhattacharyya Coefficient. It's just a way of comparing distributions, and so that's what they did.

6 - Gradient

So, we need one more thing to finish mean shift tracking. You remember before, when I was showing you the mean shift example we were plopping down a circle, right, and I said we'll worry later about the shape of that? Well that circle can be thought of as a uniform kernel. What I mean by uniform is that its height is the same everywhere around and it happens to have a circular shape and it would be written like that. That's not such a great kernel, what do you think a better kernel might be? Well, instead of using uniform kernel we could use something that is differentiable, we'll use that in a minute, isotropic, all that means is that it's the same in every direction, monotonically decreasing kernel and there's a certain convexity property we're not going to worry about that. So, so what's our favorite differential isotropic monotonically decreasing always positive kernel? Yes,

it's the Gaussian. Okay. So here it's written just as a constant proportional to a fall off by the square of the distance, okay? You can also have a scale factor, right? So if you wanted you could talk about like, the sigma, sort of the, the scale of the kernel. What's really cool about this is that, unlike the uniform kernel, which goes straight and then falls off on the edge, this is differentiable, okay? You can know how much it's changing, and it also means that it goes down to zero at the edges. So as you slide that kernel along a new point, it just has a very, very small weight, in fact, an infinitesimally small weight, and then it grows. Likewise, points that are right near the top, their weight doesn't change very much, right, because as you slide it along, the weight stays about the same. The points that are going to have the greatest value are the ones that are along the slope. What that means is, you can know how your function changes as you move, that is your overall function. So, your coefficient for example, and what you can do is, you can use this gradient to essentially hill climb up your similarity function, right? So you can, by computing this, you don't have to sort of blindly search, you actually follow that gradient and get to a local maximum. All right? And that was the basis of mean shift tracking. So now, going back to our, our previous example here of the ping pong, right, we start with the, the current frame. We want to search, but now our search can actually climb to the maximum of this function, using that gradient information, and that's mean shift tracking.

7 - Mean Shift Tracking Results

In the example shown here, remember we talked about a feature space that may be four by four by four for RGB? Well they did 16 by 16 by 16, and then they manually selected a target on the first frame. So you can see here that, see this ellipse that's been put down here, and this ellipse that's been put down there? They, they specified that in advance and they said, go, and that one starts going, and I guess they go here, and that one starts going. And I want you to notice, whoops lets get rid of the, how well those mean shift models stick to the players, okay? The average number of iterations was only four, that is from one frame to another, they had to move, do the mean shift thing four times to track the density. They also worried about scale, in terms of looking at how the the scale changed.

8 - An Unfair Comparison

There is however, another way of thinking about the tracking that I just showed you. Let's not worry too much about the mean shifting part of it and just this idea of using the color distribution as a, the matching of the color distribution, the similarity as a sensor model. Okay? Because when we do this mean shift and we're tracking the single peak, we're a little bit back to like our Kalman universe, right? Where the idea is there's a single hypothesis. We could just use this similarity function, the matching function of the color distributions as the sensor model for particle filtering. That is your particles spread out and they want to say, well, what's the probability of getting this sensor measurement if that's actually the mice state. Well, you can simply say, well, it's just a measure of how close the color distribution is using that body chart distance of some other way of comparing of distributions. And in fact, people have done this and now I have to show you a totally unfair slide. And the reason it's unfair is this slide I found, I was looking is in a presentation given online, talking about mean shift. I don't actually know the source of this picture, so I can't give you the, the reference to look it up, etc. And then they show this interesting thing where the mean shift is tracking this cute little kid and then falls apart, okay? Now why would it fall apart? This is a little bit sneaky. You'll notice this brown envelope cardboard box in the background. It has some color distributions similar to the face of the kid. So, if you've ever thought little kids are really, pretty much cardboard boxes, you're right. Or at least, that's what mean shift says. And then what they showed is that if you used a particle tracker, particle filter tracker, it falls off a little bit. See, it fell off, but then it catches back on. That's what particle filters do, right? They entertain multiple hypothesis. So, even if some weird hypothesis is looking good as long as the real one eventually becomes better, it'll come back to it and that's what its showing here. But basically, the idea is you can use the similarity function of the two color distributions as your sensor model for a particle

filter. And this relates to this idea of what I said before that, particle filters really are pretty flexible. You can pull out whatever sensor model you want to use as long as you can put it within this Bayesian framework of p of z, given x.

9 - Tracking People by Appearance

One more little specific tracking method and this comes from the work of tracking people, particularly their limbs and stuff by learning their appearance and the idea is they build a person model. And the person model is made up of a couple different things. It's made up of an appearance, we'll talk about what that is in a minute, plus structure. plus dynamics, allright? Now the structure and dynamics are generic, that is, they are the lengths of my body, the dynamics are how they can move so, you know, what are the speeds people tend to move and some of the process noise, but the appearance is person specific, okay, so the idea is I'm going to learn the appearance, and then I'm going to do the tracking. And that's represented in this picture here, allright? So the idea is given some input examples, we build the model, and the model is, like I said, it's generic structure, but we might learn the colors and, you know, the textures of these different parts. And then, using that appearance and the structure, we can do the detection running, on the, on the new frames. Here's the reference, Ramanan, Forsyth and Zisserman, and this is showing you them tracking those people parts. And the idea is they use a appearance. That's all I want to talk about that specific method. A bit later actually, towards the end of this course we're going to describe some recognition methods that use a lower dimensional space for doing recognition. We've mentioned piece, principal component analysis, et cetera. Once we do that, we can talk about a couple more one or two more tracking examples that use a PCA-base description as their appearance model but until we have that it, it would be premature to talk about it.

10 - Tracking Issues

So, finally, I just want to talk about a couple of issues that you'll always face when you're doing tracking. So the first one is initialization. How do you start the thing? And there's no good answer. The examples I showed you before were done manually. Somebody says, oh, track that. Okay? So you draw a little ser, you say, keep track of that thing. You might have a background subtraction method. Okay, so every now and then, you may be able to subtract that background, find the objects. And say okay, keep track of that object because it's isolated. And I'm going to track it as it moves along even when it becomes like partially occluded or, or with other moving things in the way. Or, you might have a detector that can detect things when they're clear and fully visible, right? So I've got a a bank robber detector. No. I, I might have a particular car detector and it'll say, oh, a jeep has just shown up. Follow that jeep. So the detector is running, and it'll only fire sometimes, when it gets a good, clear view of something. But then you want to track and follow it through. Next question is where do you get your sensor and your dynamics models from? Well we've talked a little bit about this, your dynamics models you maybe can learn by essentially from real data. That's generally pretty hard. Because if you could track the real data without the dynamics model, you wouldn't be using the dynamics model. So, you might use it from clean data. Remember we talked about the girl jumping in front of the black background? That would be some clean data. Or it says here you might specify domain knowledge. That means you're, like, the smart person. You just write down what it is. Or you take another trip to Delphi, and you get another thing from an oracle. Now, for the observation model, typically, it says generative observation model. What that means is, remember we did the, the model for the robot sensing distance. So if I knew the, the wall was three meters away. And I take some measurements. And, I collect them and I get a distribution. Now I know what my distribution would be. The trick is, I had to know that the thing was actually three meters away, or two meters away. Right, I have to know the ground truth. So you have some sort of absolute sense or measurement. Prediction versus correction, remember this whole thing about tracking and inferences. I make some prediction, I take some measurements, I do my correction. The question is, what's that relevant trade-off? Remember we talked about this in Kalman filter, which basically came down to what's the relative size of the noise of my

measurements versus the noise of my process. Actually, we usually do it the other way around, the noise of my process versus the noise in my measurements. If one is too strong, you'll ignore the data. If the other is too strong, you'll ignore the predictions. So getting those balanced right is just a little bit of magic that you have to do based upon looking at the data. Here's a more interesting or I should say fundamental one and it's referred to as data association. Data association basically is what if we don't know which measurements to associate with which tracks?

11 - Data Association

So up until now, we've pretty much assumed that whatever we measured is associated to the thing that we're going to, that we want to keep track of. But in reality, there are a couple things happen. First of all, there can be multiple objects. That's what this cute little airplane thing is supposed to look like. And in fact radar, which is where the birthplace of multi hypothesis tracking came from. Had this problem, right? I get a while bunch of pings back, there's a bunch of airplanes, I gotta know which pings go with which airplanes in order to track, all the different airplanes. Or there can be what's known as clutter. Clutter means there's a lot of stuff in the environment that has nothing to do with me. By the way, clutter is the basis of camouflage, so when you take a look at this rattlesnake whose contour you can just barely see here, there's all this other stuff that has nothing to do with it being a snake. So it confuses your visual system. Fortunately when the snake starts to move if the brush is not moving also then you can do the association differently. But in general clutter means that there are a bunch of measurements that are not from the thing I'm really interested in tracking. The problem is referred to as the data association problem and the trick is to determine which measurements go with which tracks and which measurements should be ignored. Which measurements cause you to start a new track. These are all things that you have to do when you do tracking. So there is a simple strategy and that's what's indicated here I've got a point and I've got state now I take some measurements I might just associate, I dunno the closest measurement to my prediction. So you remember when we were tracking the contours for the condensation as our Blake algorithm? Right? They took the distance to the nearest strong contour as if they new somehow that that was the right contour. They just made that assumption. Unfortunately, that doesn't always work. That clutter problem. In fact, one of the points of clutter is for it to confuse something that's trying to do that association. So those of you who watch Tom Clancy movies or read Tom Clancy books, like I've done, etcetera, you know that submarines will jettison these things that spin around and make noise and bubbles. At least I assume they do. They did in the movies, they did in the books. Now the torpedoes have a data association problem, right? Which thing that's making noise or making a signal should it decide is the real state. Personally I would think seeing some. You know, many thousand ton submarine compared to this little thing spinning around. Wouldnt been that hard but obviously the torpedos arent that smart. But so it doesnt always work in the presence of clutter. There is a most sophisticated approach to doing this, and that is You do what's called multiple hypothesis. So multiple hypothesis is, well, okay, I've seen two things now, and maybe it's one thing, and I should associate the real object with the left measurement, and the right one is clutter. Maybe it's one thing, and I should associate the object with the right-hand one and the left one is clutter. Or maybe it's two things, that is a new thing should get created. Well like I said, there's this whole area called multi-hypothesis tracking where you track all these different things. You actually can do this using a, a particles. Allright? You can do particle filtering in a not so obvious way, right? Each particle Is a hypothesis about the current state. So, for example, a particle can say oh, there are two such objects, and here's where they are, or a particle could say, there is one such object, and this one is not the right one, et cetera. So you can do that using particle filtering. And in fact I one time told Michael Eishart I, I said Meaning I said, you know, for a lot of situations, I've found that particle filtering was just a convenient way of doing multi-hypothesis tracking. And he said absolutely. So Michael, if you're out there, I remember that conversation. I think he's on the West Coast having a great time. So. So good for him.

12 - Drift

The last issue is what's called drift, okay. Drift is the error that happens that accumulates over time. That basically what happens is you've got this dynamic model that's making a prediction and, and you've got its censored observation model etc. And you're really caught with two choices, okay, and both of the choices are lousy. Choice one is, I stick to my target model, right? I really believe the model. And I never change it, like its appearance, it's colored up just. The problem with that is what happens if the object is changing? All of a sudden, you know, it goes in the trees, shadows, things look a little different, and I didn't update my model. So, almost every tracker does some form of model update. The problem with that is when you do model update, you run into situation like this, okay? So this is actually taken from the Tracking People paper, although this was, they were explaining how not to do tracking. So they've got this little box that's stuck on the middle of the ice skater. And each time they're updating something about the description, and eventually the box slides down. And what's not shown here is, it might get stuck on the background markings in the, in the rink. And so what's happening is you keep changing, changing, changing and all of the sudden more of the mark, of the target model came from the rink than came from the skaters. So as the skater leaves, the model stays right there. And this is the, the problem with adaptive tracking. And so all real tracking, in order to fight drift, always has this balance between how quickly or how much do I change my model versus how much do I keep what I had from before. And you might ask, well, how come I didn't see any of this in all that Bayesian stuff that you did? The reason is, I gave you the sensor model, P of z given x. And I really didn't tell you how it behaves generically, or I should say, all the time. It just, we, we in fact, all we did was we generically gave the algorithm. P of z given x might be how well do I match the pixels. Well, but if the pixels are changing, I have to change my sensor model. So that's the place in the algorithm where that notion of how well do I hold onto my old model versus change it. That's the place where it stays.

13 - End

So that concludes this lesson. It concludes the lesson on tracking. It concludes what seems like months of recording about tracking. Somehow this ended up being a lot but its really a cool part of computer vision. You have probabilistic state. You have the notion of measurements and the idea of combining those to come up with what's really going on in the world, which at its heart is what computer vision is all about. Also for many of you, even if you don't end up doing computer vision this idea of maintaining a consistent interpretation over time as something changes becomes important whether you're tracking the state of traffic flow over a bridge, something, whatever it is this idea of maintaining Interesting interpretation over time. So some of the principles we learned here from computer vision, can be applied to those more generic situations. Anyway, that's it for tracking. Onward and upward. Pretty soon we'll start talking about recognition, and lots of other cool stuff. See you later.

8A-L1 Introduction to recognition

1 - Intro

All right. Welcome back to Computer Vision. Today, we're going to start heading up the Computer Vision food chain. We started with basic images as functions. We did some image processing, edge detection, manipulation. Then we did some low-level computation of vision and things like motion, stereo. We considered some geometry, such as things like essential and fundamental matrices that related how cameras were viewing a scene, panoramas, sift features, et cetera. Even the tracking of objects. But all of that stuff is really semantic-free. And what I mean by semantic-free is there was no notion of what the object was, what you and I would refer to it. So when I look at the table, I see a, a cup of coffee that I paid way too much money for at some really expensive coffee shop, because that's what we all do in academia in these days. But the idea is that I know that, that thing

is a cup of coffee. So for this section and a little bit of the discussions going forward, we're going to be talking about recognition. Recognition being the labeling of the object. In particular, we typically think of it as labeling objects with labels that humans would understand. The other thing I want to say before we get in is we're only going to do a little bit of this. And that's because object re, recognition has become hugely driven by machine learning. Machine learning is a general field where you have lots of data that some of which is, is often labeled or supervised, described what's there. And you have training data and you use that to be able to do label or categorize or describe some new novel input. We're going to talk a little bit about that, but only a little little, because machine learning is such a big field that if you want to understand more of the details, you, you almost have to take an entire machine learning course. So we'll be touching on the underlying machine learning methods, especially a couple lessons from now. But mostly, we'll try to talk a bit about how its been applied to a Computer Vision scenario.

2 - What Does Recognition Involve

So what does recognition entail? Well it's actually many different tasks. One example is somebody might show you a particular area and say, you know, is that a lamp? Okay, and that's verification. And the, this particular case what I mean is that it's verification of a general class. Okay? Is this thing a lamp? We'll get to identification in a minute. Another possibility is you might say, are there any people here? And that would be detection. That is, I haven't told you where in the image to look, I just want to know are there any people in this scene. Another is, I might have a specific instance. And I might show you an example, and say is that Potala Palace? That's in this particular case, of a location, of a thing. Sometimes, identification, we all know, is in faces, right? I show you and say, you know, is this Megan? And you say, no, that's not nearly as nice as Megan. But anyway, that's identification. A more interesting question is what we'll loosely call categorization, all right? So you're generally putting out a label. You know, here we have this label that in the background there are mountains, and there's trees over here, there's people here, vendors, this thing was a street lamp, these are buildings. And this is what's referred to as object categorization and we're going to talk more about that. But we might also want to just label the entire scene. Say you know, this is an outdoor city where people go and buy stuff. All right, so that was also, would entail some form of recognition.

3 - Object Categorization

So there are all these different ways you can think about what recognition is. And one of the two main distinctions we've already talked about, but it pays to point it out again, is this difference between single instance labeling, okay? So here's a picture of a car. And we say, you know, that's John's car. Okay, and so we want to be able to find that particular thing versus we'd like to be able to find all of these things and say, okay, those are cars. For the most part, we're going to be worrying about this problem, the generic categorization problem, okay? So object categorization described in a variety of ways. This is a slide from Kristin and Barrett Leibe, and that I changed a little bit. Generically it's kind of like this. Given some number of training examples, training images of a category, recognize some a-priori unknown instances of that category and assign the correct label. Now you notice that I put down small in parentheses, you know, right here it says small. These days not a lot of work is done on the small number? That is that, typically we have lots of large examples. And there's a question about how method's that go from taking large data could then use small data? To me that's a slightly more interesting problem, but we'll leave the small as being sort of optional at this point. But the basic idea is that I give you a set of these things and some sort of labels, and you want to give me back the correct label. All right. So one of the things we have to do, before we can even proceed is, we have to agree on a set of labels. And there's this fundamental problem that a single object doesn't have a unique label. So here we have somebody's dog, apparently named Fido, I don't know if anybody names their dogs Fido anymore. So you can say well this is Fido or you could say this is a German shepherd. You could say this is a dog. You could say, doesn't say, say it's a mammal, you could say it's an animal. You could say it's a living

thing although a living being is probably I don't know out there. But it, it is these things, right, so there's this question is what categories are best for visual identification or which ones do we agree upon.

4 - Visual Object Categories

Well, to get some inspiration about this, let's talk about what humans do, and visual object categories in humans. And there was work in what is referred to as Basic Level Categories in human categorization. It was st started by Eleanor Rosch, series of seminal papers back in the 70s. Continued on by Lakoff and other cognitive scientists. And what they found is, there are certain levels of description that seem to have a preferred place in human cognition. So for example, the basic level was the highest level in the category, which you could say that objects have the same, or similar, perceived shape. So if I ask you to think of the shape of a German Shepherd you can, but if i ask you to think of the shape of a dog, you probably can do that too. If I ask you to think of the shape of a mammal, what you're going to do is you're going to think of the shape of a couple of representative types of mammals. You don't actually have a shape for mammal. Typically also, you can make a mental image. If I say, you know, make a mental image of a dog, you'll probably come up with something. If I say make a mental image of an animal, you're going to pick some animal and then if I were to ask you what it was you wouldn't tell me it's an animal you would tell me it was a deer or something like that. By the way you can do reaction time experiments, where, let's suppose you want to ask people to push a button yes or no. I'm going to show them a picture and one task is you're supposed to say is it an animal? Another task you're supposed to say whether it's a dog. It turns out you're faster at knowing that it's a dog than you are at knowing it's an animal. And that's a very robust finding and you, you can, you can do that. And then there's this thing that typically this is the level that children understand earlier. So they'll learn about dogs before they'll learn about animals, they'll learn these words. And there's one written here about having motor actions interact. You can imagine there's a set of things you do with dogs that sort of span dogs, right? Right? It's probably not the same set of things you do with animals. Right? You probably pet dogs. You probably want to pet most dogs. Probably pet all dogs generically if you're a dog person. The idea is that you have these, these interactions. So there's this preferred level within the human. And you might wonder why I'm spending so much time talking to you about this. Well if you go to the MIT libraries and you type in, you could actually find this called NOC, natural object categorization, which by the way, was my PhD thesis back in 1987. Please don't go and read it, it's not so great, but I was worried about this notion of perceptual, organization and how you would organize your categories to make them good for perception. The one thing I lucked out on is that by the time I did my thesis, the AI tech reports were up to 1001, so I got tech support 1001 as the number, and and that's great because it sounded like it was an important one or first.

5 - Other Types of Categories

All right. Enough of that, but this idea of which categories are best for visual identification is a challenge for doing this labelling. So, let's assume for a moment that we're going to have some set of visual categories. A question you might ask is, about how many of them are there? It's kind of a weird question to ask, you know, how many types of things are there? But you know, look, we've got a finite amount of gelatinous stuffed between our ears, and there's this question of sort of, how many different labels do you deal with? So Irv Biederman, a psychologist and does work in cognitive vision and, and mental models for doing vision, back in the, in the 80s, he came up with this number of about 10 to 30 thousand, okay? Now, big range, but it's an interesting idea, because it's also related if you take a look at the number of nouns people know. That is, you could ask how many nouns do you know? Well, it's not going to be 2 million and it's not going to be 11, at least I hope if you're watching this, it's not 11. Okay? It's in the tens of thousands of range. Okay? So it gives you some idea of sort of the scale of the recognition problem. There are, by the way, other types of categories other than just sort of natural categories. So, for example, you think about functional categories. So, here's a set of chairs, and yes, you can sit in all these things, although

some of them look like they would be kind of stunningly uncomfortable, but they don't really share a visual property. So you can imagine that semantically, in your head, you know that they have a certain relationship. But from a recognition perspective, it's not clear that the same process that recognizes this as a chair, should also recognize that as a chair. Another type of category which is valid, but not perceptually relevant might be the sort of ad hoc categories of things you find in an office environment. That's a perfectly natural category. In fact, it's even a category which you can think about probabilistic reasoning as biasing things. Right? If I tell you you're in an office, you're much more likely to find a stapler, right, than to find a horse whip. So maybe that would bias your recognition. But the notion of it being a single visual category as you can see from these objects doesn't make a lot of sense.

6 - Using Recognition

So why are we working on the recognition problem? Well, recognition, it's a fundamental part of perception. Whether you're talking about robots, autonomous agents, whatever. The idea is that the ability to know what things are and therefore how you want to interact with them, is it really important? It, it's gives you a way of organizing and thinking about the visual content of the world. And perhaps even more importantly, it's a very human way of thinking about things, right? I, I give objects labels, and if I tell you to go pick up the cup and put it away, you know what I'm talking about. So we want our machines and systems to be able to do that too. So for example, autonomous agents, and here's a pair. The one on the left is a robot. This is from Karlsruhe, and Rüdiger Dillmann's work on, domestic robot. And this robot has to be able to go pick up cups out of dishwashers. It has to know that there is a dishwasher. It has to see that, oh, I put that cup down, it didn't make it, I'm going to go get it. It has to be able to be told that the cups go over there. All right? So if I'm going to communicate about these things to that robot, I'm going to use labels that talk about these objects. Even in autonomous systems. So here, I think this was, Stanley. I think that was one of the Urban Grand Challenge vehicles, one of the winners. It has to understand that this is a thing and this thing we know that it's called a car. All it needs to know is it's this thing that it's learned a lot of things about. And that this kind of thing which are cars do things like continue moving, might stop, have certain behaviors, likewise that this object there might move in your way and it uses a lot of machine learning to do that. Maybe not quite the traditional labeling in the way. But the idea is that these autonomous agents understand the nature of, of the world being carved up in objects. And I can communicate about them and it can autonomously detect them. Labeling people. That's a recognition task. This is something we do all the time. You know, every time you tag folks on Facebook, and it both finds that it is a face, and we'll talk about detection in a little bit. But also, you might want to know who it is. Peter Belhumeur and folks did these digital field guides where you'd like to be able to like maybe take a picture of a leaf and say you know, what kind of leaf is that? If you're into that nature kind of thing which I'm not. But it's this idea of being able to, to recognize things. And maybe you've found some really expensive stuff online and you say please find me more shoes that look like this. For any relatives of mine that I'm still continuing to subsidize. So, you could use computer vision to help you find less expensive shoes. Just saying. All right. The idea of, you know, can I find something that looks kind of like that, so I have to recognize that it's a shoe. Maybe it's already labeled as a shoe, recognize the type, that kind of thing.

7 - Challenges

Why is this hard? Well because unlike some early computer vision data sets we don't typically take an object put it on a piece of black felt, or velvet on top of a table and say okay, that is a fire truck toy. Right? Pictures show up more like this, and I think this also comes from Kristen cause she has a thing for pandas and I think also for koalas, all right. You get great variations in how things are illuminated, the pose of objects. The objects are sort of seen in the background with a bunch of clutter, they get occluded so here, you know, there's trees in the way, you know, the viewpoint, and then, oh, isn't this a really cute face of a koala. Yeah, but you know what to another koala he may

not really like this guy because he, like, ran off with his poker chips, or something, I don't know. The idea is that you know, they, they look different one to another so the idea that we have these variations in appearance. The other thing is that these objects don't show up again in isolation. They show up in these cluttered overlapping environments. You know, here's this picture of these two motorcycles one behind the other, people, of course, walking pedestrians. We don't get to look at objects in isolation typically. Another thing that makes this hard is the importance of context. And let me show you three pictures, okay? So you take a look at these pictures and you probably see a person here talking on his cellphone. You probably see here an outdoor scene here with a building and like some people standing et cetera, whatever. You probably see some guy, I don't know exactly what he's doing, bending down, maybe picking something up and here's his shoe over there. Right? Right. Well, hm, you see these regions so that the phone, the person, the shoe, see this single picture here? Let me blow up that picture a little bit more, okay? This patch of intensities is the same patch here, as here, as here, as there. So what's a cellphone in one image, is a shoe in another image. And that can only be done by the contexts of everything else that's around it. This also makes recognition difficult. Another challenge is the complexity. That is, we need to able to compute things and do it relatively quickly, because you know, there are thousands now, now millions of pixels in an image. We've got, I don't know, 3 to 30, some tens of thousands of object categories. We have all sorts of degrees of freedom of pose. Yeah, that was cute, I know, right? So even though it's just a person, a person can move in many shapes, although I can move in many fewer shapes than I used to be able to move. And there's so many pictures, right? There are billions of images indexed by Google Search. The last number I could find was that in 2011, six billion photos were being uploaded to various things per month. Okay? There was a billion camera phones sold last year. Okay? So they all take pictures. And another interesting indication of the complexity of recognition, and of vision in general, is how much of your brain and my brain too by the way, is devoted to visual processing. There are different estimates, anywhere from a, a quarter of your entire cerebral cortex to a half. And it's the cerebral cortex that sort of distinguishes you from an awful lot of other mammals. So the recognition, the ability for you to manipulate things so well visually, seems to be two things, very intense. It takes a lot of wetware in order for you to do it. But I guess evolutionarily it must have been very useful. Because you devoted an awful lot of wetware to handling this

8 - What Works

What currently works? Well actually, let's talk a little bit about what used to work, well, still does, what worked a while ago, and what's working now. So what worked sort of say yesterday? Okay. Yesterday being proverbially not today. All right, so things like reading license plates, well that's pretty easy because it's a fixed font. Zip codes, numbers on checks, those are getting much better, and handwritten recognition of digits has gotten pretty good. Fingerprint recognition. The analysis of those patterns, because the patterns share certain types of characteristics, and we could find those characteristics. Face detection, but you'll notice that here the thing knows who I am, but it hasn't yet figured out who Megan is. Right? Today we do recognition. By the way you know, these days typically when you pop-up or put an image let's say on to Facebook or some other thing, it'll suggest you the names of the people that are there. And that means that it's found the faces, and it's also recognized who they are. Now, it doesn't searches database of everybody, it only searches your friends. Right? So it doesn't have to try to recognize this as anybody on the planet, just the five friends that I happen to have. Right? So then you might ask, well, how does it know the picture of those friends? Well, other people have tagged Megan in images. Every time you tag an image with a face, person's name, you're actually telling Google, or whoever, or Facebook, or whoever you're doing, how to recognize that person. So without meaning, making it sound nefarious, it is the case that every time you tag somebody else in a picture, you're making it that much easier for computers to find that person in other images. You are teaching. You are not tagging, you are teaching the system what Megan looks like. So just, just, just think about that every time you identify people in an image. We've actually talked about sift features and using the location of sift features to recognize objects and, you know, sort of flat textured things like the book covers. All right. Now

what just happened? This is GoogleNet 2014, which was part of a competition. And you can see here that given this image, okay, it labels the monitor, the bookshelf. It even labels this dom, this, it says domestic cat. And it looks to me like this cat is jumping in the air, so that's pretty cool. What makes this picture hard, of course is the very intense color variation. So it was able to find, just basically through the color and texture, cause that's the only thing that's there, the fact that there were oranges and bananas there. So this is a pretty intriguing result, and then here's a result that, calls into question the importance of context. And maybe what it means is that context doesn't matter when things are very clear. All right? So I love this picture. I'm not sure you can read this, but it says hat with wide brim on the top, and it says dog on the bottom. Now, I don't know how many pictures there are of dogs with hats and wide brims wearing them. What scares me is, there might be tens of thousands, because people have way too much time on their hands or something. I don't know. But, it is an impressive result that it was able to recognize the hat on top of the dog, which you have to believe is not a likely position for a hat.

9 - End

So going forward, these days much of strong label recognition is really machine learning applied to patterns of pixel intensities, or features. And to cover this, as I said, would require a deep understanding of machine learning, and a class or 17 of them to learn that. So, instead, what we'll do is we're going to focus on sort of two general principals called generative versus discriminative methods, and their application and the representation used in computer vision. And then we'll spend some time on activity recognition from video because why recognize pictures when you can actually recognize activity. And besides that, it's how I earn my tenure. All right. Talk to you soon.

8B-L1 Classification: Generative models

1 - Intro

All right, welcome back to Computer Vision. This time we're going to start sort of on some of the small technical details of doing recognition. Remember we said that recognition is mostly going to be some form of a classification problem, now we're going to look at some methods and some representations. So today we're going to start with the one we talked about, supervised classification, and in fact, almost everything we're going to do is going to be in the supervised domain. So in the supervised world, basically I give you a set of labeled training examples, and the goal is to come up with some function, but if you give it a new example, it'll generate the correct label. So using the handwritten digit recognition as a example, so here we have some training examples, right, we've got a bunch of fours and a bunch of nines, and you'll notice, they're not, it's not trivial right, because these, these fours, you know, could have been a nine, but no, they're actually fours. So what I want to do is I want to come up with a function such that if I give it some novel input, I give it something that looks like this, it's going to say, what? Is it going to say four or nine? Nine Very good. Very nine. Megan's been paying attention.

2 - Supervised Classification

So the question is, how good is this function that we build in order to do the classification? And you can't really answer that and tell me, until you tell me what good means, all right? And what good means, two things. First of all, what kinds of mistakes does our classifier make? All right? And for each of those mistakes, sort of, how expensive is it to make those mistakes? And when we build a supervised classifier, or supervised classification, since we know what the desired outputs are for a given set and then we can run the thing as we're designing it, what we want to do is we want to minimize the expected misclassification cost. Okay? We want to make mistakes as infrequently as possible. And the more expensive the mistake is the less often we want to make that particular mistake. All right, to do this, to handle classification, there are two general strategies. The

first method, which is referred to as generative, you basically use your training data from class A to build a model of class A. Separately, you build a model of class B, you build a model of class c, you build a model of class d. And then when you're given a new example, you say well does model A explain it better, or model B, or model C. Whichever one explains it better, that's my, that's the one I pick. And it's referred to as generative because the idea is that I have a model that describes sort of all the different possible things that would be of class A. At least, that's the hope. Or you could even think of it as generating a set of examples that are class A. The other way of doing it, is referred to as discriminative. Discriminative says, well, here's A, and here's a bunch of things that are not A, and let me learn how to discriminate between them. Okay? So I'm not going to worry about modeling what all of A is. What I want to do is, if I only have As, Bs, and Cs, I have to look at how would I discriminate between A or Bs and Cs? Or I could also do A versus B, A versus C, B versus C and, and, and learn those, but the discriminative, the discriminative approach focuses on learning the boundary, the separator between our categories. Today what we're going to be doing is talking about generative, and then we'll move on to discriminative in a couple of future lessons.

3 - Generative

Let's talk about the generate method, and like I said, we're going to be given some training leg, examples and we want to predict the labels, okay? I need a little bit of an invitation because we're going to do a trivial amount of math. This little notation, 4 to 9 means an object is really a four but I call it a 9. We're going to talk about the costs or the loss, you're going to see L for loss, you know, what's the, the cost to me of calling a 4 a 9 or calling a 9 a 4? All right? And one of the things we're going to assume for the rest of today, and it's really not a big deal, you can factor we're going to assume the cost of calling an X an X is zero, okay? So nobody gets upset if I call a spade a spade, so to speak. So, let's consider the two class binary classification problem. Remember we just had 4s and 9s. So we have two kinds of losses, all right? So that's what this L means. An L of 4 9 is the loss of classifying a 4 as a 9. And conversely, L of 9 to 4 is the loss of classifying a 9 as a 4. Okay? Pretty straightforward. Risk is your expected cost, okay? So, if I've got a particular strategy, S, classifier strategy S. We want to talk about what's the expected loss, all right? And for our 4 and 9 example, the risk of our strategy is, what's the probability that I'll call a 4 a 9. And how often is that going to happen? Okay. Using this particular strategy, and I multiply that by whatever the loss is, the cost of calling a 4 9, and what's the probability of calling a 9 a 4 and what's the loss when I call a 9 or 4? All right? And that's the risk because the probability is, you know, 15% of the time that I'm running this thing, 15% of the times, I'm going to 4 is going to come up and I'm going to call it a 9. Okay, that's what probability of calling a 4 9 means. So this total quantity, that's what's referred to as your risk. And remember what we want to do is we want to minimize the expected risk.

4 - Minimal Risk Part1

So here we're going to work through this simple example again, using our 4s and 9s. We're going to assume that we have some feature that we're measuring, x. It's typically a vector, but I'm going to show it here as a scalar. So x goes from small x, so little x, to big x. All right? It's changing, changing values. Clearly, all the way to the left, I've got more 4s, and all the way to the right, I've got more 9s at the best decision boundary. Okay? And that's what's indicated in red here. At that boundary, if I call it a four, or I call it a nine, it should cost me on average the same. Because in fact, if, if calling it a four would be less risk than calling a nine, I'm going to move that boundary this way. And if cost, calling it a nine is better than calling in a four, I'm going to move the boundary that way. So at the best possible boundary, the expected loss is the same regardless of which label I announce. All right, so if we picked label four, okay, at that boundary. Okay, then the expected loss is what? Well, it's just, what is the probability that it's really a nine, and that I call it a four, the cost of calling it a four? And what's the probability that it's really a four, and the loss of calling it a four? Well if you remember, remember we assumed that the loss of x to x is zero, that is, there's no cost of calling a four a four. Well that means that this just reduces down to, well, if I'm at this location, x, what's the probability it's really a nine? And the cost of calling the nine a four?

That's the expected loss of saying four at that location. Likewise, what's the expected loss, if I say nine at that location? Well, symmetrically, it's just what is the probability that it actually was a four? And what's the loss of calling a four a nine? Okay, so those are the two different value, the losses, that I want. Now, remember we said before at the best decision boundary either labeling a four or labeling a nine, the cost, the expected cost would be the same. So the best decision boundary is where, is the x such that the probability that the class is really a nine times the cost of calling a nine a four, will be equal to the probability that it's really a four times the loss of calling a four a nine. When I set those two values equal, that's the x that I want for my decision boundary. What that means is, if I'm going to classify a new point, I'm just going to look at these values, right? At some new point x. What's the probability that it's really a four times the probability that I call a four a nine? If that's greater than the probability that it's a nine times the loss of calling a nine a four, if that's true, I'm going to call this thing a four, otherwise, I'm going to call it a nine.

5 - Minimal Risk Part 2

What I've got plotted here and sort of a notional plot, this is a loss, expected loss, right? On the left hand side, that's the probability that it's a four given an x times the loss of that it's a four and I call it a nine. Over here, this is the probability that it's a nine, but I actually called it a four. And the decision boundary is where these values are equal to one another. And of course, what is hard about this? Well, knowing what the losses are, that's easy, okay? You just tell me, right? You tell me, how expensive is it for me to call a smart person not so smart versus calling a not so smart person smart, will probably have something to do with their stature. Okay? So my losses might be asymmetric, or my losses might be symmetric. Like, I'm doing zip codes, right? And if I call a four, a nine you know, it goes to California and it should have gone, what starts with four? What zip codes start with four? I have no idea. I don't know, let's say Forest, Texas. So, if I called it a four and it really was a nine, it's supposed to go to California, it goes to Texas. If I call it a, etc., it goes to te. So assuming that it's not any worse ending up in Texas than it is, or ending up in California, which is a difficult assumption, I understand. See there are more people in California than there are Texas, so I have to in, so my loss of insulting Texas is lower than my loss of insulting Cal, oh nevermind. Okay, getting back to this. So I know what my losses are. The thing that I don't know is the probabilities. Right? For a given x, how do I actually know what the probability is that it's a four? I have to somehow learn that or know that. So who's going to help me?

6 - Example Learning Skin Colors

Whenever we do recognition machine learning, what's going to help me? It's the data. The data are the first thing that's going to help me. There's somebody else that's going to help me. I'm going to bring them along in a minute. So here we have some images of pictures, and as you can see in the red circles, there are some pixels that are from, say, skin. And there are pixels that are not from skin. And let's suppose I just want to label a pixel or little patches. Is that skin, or is there not skin? By the way, this is a cute, little thing for those of you out there. We did a bunch of skin detection et cetera. And in Sandy Pentland's lab, and I think Thad Starner was involved back when he was a student, it turns out the color of skin is pretty constant. The luminance changes, right? So some people, you may have noticed are darker than other people. But they're darker, they just have more of the same pignent, pigment, right, the melanin pigment. So when you look at skin in a color space, if you remove the illuminant, it's actually possible to detect skin for a very wide range of how dark the skin is, using the same color removing the illuminant. Now you have to worry about lighting and a bunch of other things. But it makes sense, right, because there's only one pigment. Anyway, we're going back, we're looking at skin color. There's a bunch of pixels that are skin color and a bunch of pixels are not. So, what am I going to do? Well, what I do is, I'm going to take the skin pixels, all the pixels that I, somebody supervised, remember supervised learning, have classified as being skin pixels. And I"m going to build a little histogram of their feature. Now, the feature that we're using here is hue. Hue is the, essentially the color with the amount of brightness and saturation removed. It's, you know, how red, greenish, bluish, yellowish is, is the color. And

what this is, is this is a little box that's essentially the percentage. It's a histogram that's the percentage of pixels, of the skin pixels, that have this hue. And a histogram is an approximation to the probability distribution. But notice that it's the probability of getting that particular hue given that it's skin. It is not the probability that it's skin given these hues, okay? This is the, remember the likelihood, all right? Well what do I do after I've done the skin pixels, well I do the not skin pixels, I build another graph, right? And in fact what's interesting is you'll notice there's a little bit of overlap here, right? That's not too surprising. There are some, in fact in real life there actually would have been some down here too, et cetera, whatever. Because, you know, there are skin colored pixels in the non skin world, okay? And there are some skin that, you know, people go and get tattooed or something. I dont know, whatever. You know, but in general I'm not going to have a clean separation, but I'm going to have a distribution that hopefully has some differences between them. And again, this is the probability of getting that hue given that it's not skin. And it's not the probability of the other way around. So all we have at the moment is the probability of getting a hue given that it's skin, probability of giving a hue given it's not skin. But what, what do we really want to do? Well, somebody gives us a picture. This is a stunning picture of the Beatles, actually. I've never seen this picture before. Well, that's not totally true. I saw it the last time I gave this lecture, but you know what I mean. Anyway, we get in a new picture, and what we want to do is we want to label each pixel as being skin or not skin. Or more precisely, we want to know the probability that it's skin or not skin, given say, some of the color. So, do I have the probability that something's skin, given its hue on this slide? Megan, do I have it? Yes. Yes. Wrong. No, I have the probability of a hue given that it's skin, right? What do we have to do? Well you've all seen this company wor, coming. Where else do I get help from, besides the data? I get a help from Dr. Thomas Bayes, okay? Bayes rule, remember Bayes rule? We did Bayes rule now a couple of times? Bayes rule says that the probability that it's skin, given x and that's referred to as the posterior. It can be written as a prior. If I pull out some pixel and I don't tell you anything about it, what's the probability it's skin? That's the prior. And I multiply that by the likelihood, P of x given that it is skin. Now that we do have that. Remember those histograms? That's the probability of getting a hue given that it's skin. So that we have from the data, all right? But the prior is added in here. And in fact, if we remove, since we've got our measurement, we can sort of pull that out. We did that last time for our particle filtering, actually. We can say that the probability that something is skin, given a measurement, is the likelihood, or is proportional to the likelihood times the prior. Which all brings us to the question of where do you get your prior from? Okay, we've run into this problem before when we told you to go to get in your car, drive to Greece, go to the oracle, get a prior. Well that's one thing you could do, but airfare to Greece is expensive. So we have to do something else and Steve sites, wrote this Bayes rule in use/abuse. It was kind of cute, I kept the joke. Thanks Steve.

7 - Is This Skin or Not Quiz

Okay, let's see if we understand the issue here. We already have two histograms, one that gives us the probability that an observes Hue value is from a skin pixel, and other capturing the probability distribution of it not being a skin pixel. Now along comes a pixel whose hue value lies somewhere here. Say this corresponds to bin value six. Using this, we can find out the bin heights or corresponding probabilities. Lets say that the probability that this value came from a skin pixel is 0.5. And that it came from a non skin pixel at 0.1. Okay, now that you have these two values can you decide whether x came from a skin pixel or not? In general is this condition sufficient? That is I'd say that x is a skin pixel if P(x) given skin is greater than P(x) given not skin. How about this deceptively similar but different condition. That is P skin given x is greater than P not skin given x. While we're at it, let's throw in a couple more choices. P(x) given skin times P of skin is greater than P(x) given not skin times P of not skin. Is that sufficient? How 'bout this last one? P skin given x times P(x) is greater than P not skin given x times P(x). Select the condition or conditions that you think you can use to decide whether a skin pixel is being seen given a value x

8 - Is This Skin or Not Solution

So the first one seems tempting. You have these values. Why not use them directly? But note that these are just likelihoods of observing x. They are missing prior probabilities. The probability that our random pixel, regardless of its value, is a skin pixel or not, is important. Take this image for example. All these pixels on the face and on the neck are skin pixels. But as you can see, this is a really small area compared to the total image area. If this is all the data that we had, then the prior probability of a pixel being skin pixel would be pretty low. Say around 0.2. The sum of this and the prior probability of not skin must sum to one. So therefore that value would be 0.8. These two values must be multiplied on their respective sides to weight the likelihoods. All of a sudden, the distinction is not as clear. Okay, so the first one is not enough. The second choice is exactly what we're after. If we find that the probability of skin given x is greater than the probability of not skin given x, then we have no reason not to believe that it is a skin pixel. Well, that was clear. Okay, the third choice is exactly the same as the first one, except, we have the prior's multiplied. So yes, this is a valid condition. You'll see why shortly. The last one is a little less useful. In fact, it's the same as the second form but with P of x multiplied on both sides. Okay, if it's the same as the second then I guess it's also valid, but not too useful.

9 - Bayes Rule in Use

But what I'm going to do is show you. It's sort of a simple version of what's called the likelihood ratio test. And here, we're basically assuming that the cost of calling a skin pixel not skin, is the same as cost calling a not skin pixel, skin. We can re-, we can re-weight these things if the costs are different. The likelihood ratio test says basically, look. It's just exactly what you think, if the probability that it's skin given the skew measurement, is greater than the probability that it's not skin, what am I going to do? I am going to call it skin, right, that assumes that the costs are, are equal. So using Bayes rule, p of skin given x is just p of x given skin times p of skin. P of not skin, given x, is p of x, given not skin, times p of not skin. So if this quantity is greater than this whole quantity, we're going to classify x as skin. This is just a direct use of Bayes' rule. And as I said before, we would change the weights on two sides of that equation if, if the costs of making the, the mistakes were not symmetric. Allright? So problem is that I really don't know the prior of being skin. Allright, I mean, I, I dunno [INAUDIBLE] Where is the prior? So, the one thing I'm going to do is I'm going to assume is it's a constant. I'm going to assume it has some number. Allright? So the prior for a whole bunch of pixels that I'm looking at, that are maybe of people or in an office where there are people around. The probability that something is skin is some number. Here's it's written as omega. So if you give me enough training data, that is you pay enough undergraduates to circle all the skin areas on an image or you do Mechanical Turk on Amazon to indicate, to have the world out there for a penny and a half a click to say what's skin and what's not skin. You could estimate the prior that any given pixel is skin. So, with this training data I can also estimate that prior and I can measure those likelihoods, right? Those are the likely densities, those histograms that I built, and remember I need two histograms. I need P of X given skin. What are the hues that I get on the skin? And P of X given not skin. Allright, so with that same training data that I used for estimating the prior, I can also estimate my likelihoods, allright. Which means I can more or less come up with a rule. Basically what I'm going to do is, I've got my priors, I've got my likelihoods, so I can just evaluate p of x skin given x. And if it's bigger than some number, some theta, okay? I'm going to classify this thing as skin, allright? That's it. Given some new image, I want to label all my pixels. And here is a picture, this is actually a picture of Gary Bradsky and the thing on the right is showing you the probability of something being skin. And Gary actually was using this to build perceptual user interface where you can move your head around And the world that you are looking at in some virtual rendering will change as you move your, your head. And in fact here's two more pictures showing you the orientation, right so here it's pretty much straight up, here he's, he's turned his head and all it's doing is tracking the skin pixels of his face, that's all, alright? And in fact, Gary made use of this in a cool little, track, remember we talked about mean shift tracking just recently actually, just a couple of lectures ago. Gary did sort of a different version of that called cam

shift, which was maybe a little easier to implement sometimes, that's a pretty highly cited paper, and it was basically using these color distributions. In order to be able to track your head and change what was rendering. In fact, the picture on your right comes from his paper, where they were using the Quake engine for controlling a first-person shooter, and if you just turned your head, okay, the thing would just movie with you, okay? And it was just done by, by tracking the skin pixels.

10 - More General Generative Models

A couple more points about generative models, they are pretty straightforward. First of all, we've been doing the two class case. Right? You're either a skin or not skin. Well, suppose I've got a bunch of classes, c sub i, and I've got a measurement, x. Which c should I choose? It's written here in green, c star. So, very simply, I'm going to take the category that maximizes p of c given x. That is, which category is most likely given x? Okay, that makes some sense, well, but I don't have that directly, so in order to do that, I use the likelihood times the, what's p of c? That's our prior. Okay, so I have to have my prior. So, basically in generative models, as you add a new, category, you just need to build a likelihood model and you need a prior model and that will allow you to compare, one category to the next in, in deciding a new label. The other thing is, you remember before, we were drawing these, histograms to represent those, those probability distributions? Well, normally, what you want is a nice, continuous generative model, so what you need is a likelihood density. Okay? So you don't just have a history, I mean you actually want a density. And remember, it's likelihood, so it's p of x given c. Not p of c given x. That we get from Bayes' rule. All right? So typically we represent those likelihoods using some parametric form. Here it's written as a, a mixture of Gaussians, right? So here, this particular class 2, and see it says p of x, given c 2? That's its likelihood. This is one little Gaussian lump. And we do that because we said oh, that looks you know, may, maybe most of the distribution of those pixel values is actually like that. But over here for class 1, you see that we've got two humps. Okay? And that's just a mixture of Gaussians. And it's pretty, it's not too difficult to estimate a mixture of Gaussians from a modest amount of data. One, so one of the reasons to use these kinds of parameterized distributions is it allows you to use much less data in order to fit your density. So some of you might ask, well wait a minute. Why am I doing this whole continuous fitting, why not just use some like k-nearest neighbor or Parzen window method or something else that you learned in probability class? Megan is not going to remember that question so I'm not going to ask her. Never mind. So why won't you do that? Well, you actually, you might do that. If you've got a lot of data, you might actually try to do some histogram KNN based method of estimating your density. But, you would really need lots and lots of data. Sort of everywhere that you're likely to get a point, because you need the densities pretty much everywhere you're likely to get a data point. And in some sense the whole point of generative models is to be able to use a modest amount of data so the reason to use the parameterized thing is that you don't need a lot of data. Okay? It allows you to generate a model that you can use for each category.

11 - Summary of Generative Models

So, in summary about generative models, there are some good things about it, right? It's pure probability, right, very, very clean probabilistic thing. In fact, it's been known for a very long time, which should give you a hint that it's not going to be the most powerful method used, because it comes from a time when computers were nonexistent or scarce. And we had to use a lot of mathematical reasoning, pure mathematics in order to do the work. But it is, it says here, firm probabilistic grounding. It allows you to use your priors, so if you have priors, maybe your context gives you a prior. I know I'm in a bedroom, so the probability of seeing a lamp is a lot higher than if I'm in a garage, okay, that kind of thing, so I can have a prior built into it. One that's in yellow here, actually, there's two of them that are yellow because I think they actually are important. One is that because you're doing this parametric modeling of continuous functions, you can actually get away with using a pretty small amount of data compared to other methods we're going to look at later, the discriminate methods. So that's one, and the second good thing about it is your models are about

each category, and just about the category. So when you tell me about some new category, I just have to learn a model for that category. I don't have to play with this model. Whereas if I'm trying to learn how to distinguish between an a and a b and an a and a c, now you tell me about d's? I have to worry about a's versus d's as well. So when you add in new categories, and you're trying to discriminate, you often have to worry about a lot of different categories. In the generative model, I just build a model for that new category. There are some other advantages of generative models that we haven't talked about. You can use unlabeled data in interesting ways. You can also use it to generate data, right? So if I say something is distributed over a Gaussian, I can actually make more examples, and there are times when that's useful. So that would be all the good news. What always follows the good news is the not so good news. So, where did you get those priors? Okay, so again, do you go to Greece, do you go to the Sears Prior Store? The sensitivity to priors really matters, and the priors can be affected by things like context. There's this other thing that's also kind of annoying. I'm modeling skin and I've got not-skin. Well, skin you might say is a plausible model. Not-skin? Modeling not-skin is kind of a weird thing, all right? What I really want to do is model sort of the, the boundary between all the stuff that's skin and everything else that's out there. And by the way, a lot of the stuff that's out there doesn't look anything like skin at all, so that stuff is easy. What I really need to worry about are the things that are kind of like skin, but aren't. In fact, that's the next example. It says, the example hard cases aren't special, and they should be, right? So I'm trying to recognize you know, rattlesnakes in the brush. I don't have to worry about what Maseratis look like. I do have to worry about the difference between what rattlesnakes look like, and maybe different kinds of grass, because there's a tight similarity between them. So this idea that modeling the hard cases is really what's important. And in fact, I think that's what's most important about discriminative learning. And then the last thing is, if you have lots, and lots, and lots, and lots, and lots of data, it doesn't really help. If I'm modeling my world as a, sort of a single distribution, once I have enough data, doesn't really, you know, I can fit some nice mixture of Gaussians, I'm done. It's not helping me comparing against other elements. So, it's not really leveraging having large data. So those are the downsides of generative models.

12 - End

So next time, we're going to to do a we're not going to give up generative models yet. We're going to show you a really cool way of building a sort of a low dimensional generative model called Principal Component Analysis. It was first done not first, but became popular when done for face recognition, and that's a form of a general model. But more important, you'll just learn about principle components. Which is used a lot in computer vision, and for all sorts of things. And after we do that, we're going to move to this other set of methods which are discriminative. Where instead of modeling the, the, categories themselves, instead we learn about the boundaries between the categories. And that we'll continue to do as we talk more about recognition.

8B-L2 Principle Component Analysis

1 - Intro

Welcome back to Computer Vision. Today we're going to be learning about a technique that is important, in general, but we sort of slid it here and we're talking about recognition under the generative model description. So, last time we introduced which I hope for you is the last time. Certainly for me it was the last time. They say, you never forget your first time. Me, I can barely remember the last time. Last time we introduced generative models for classification. And there are a couple of points about generative models. Meghan's still giggling in the back there. One of the points was that in general models you build the new model for every new object category. And you don't really think too much about the other objects that you know about. And you just you, you build the description. And then when new things come in you see how likely is the new thing for each of the description, each of the models that you have. But there was another point and that was

perhaps a little more subtle because it was kind of Flew by. And that is that generative models work best in lower dimensional spaces. And what I mean by that is since we're going to build these probabilistic descriptions usually some sort of probability density function, it's only plausible to estimate those, and you've got a modest number of dimensions. If you have very high dimen, number of dimensions, or even, even, even just a moderate number of dimensions. To estimate a real probability density requires lots and lots and lots of data. So today what we're going to do is we're going to learn about one of the most common and, and, perhaps, I don't want to sa, call it important, but it, but, just everybody would expect that you would understand it. Method of dimensionality reduction, which referred to as principal component analysis. You may have seen this in other signal stuff, and maybe even in one of your algebra, but it became, pretty significant in Computer Vision because of this particular piece of work that I'll tell you about. And it, because of that it gets used all the time.

2 - Principal Components

Principal components. So, fundamentally, principal components are about the directions. The component is really a direction in a featured space along which the points that you care about in that space have the greatest variation or variants. We'll make all of this more formal, in a minute. So, the first, it says PC here, first principal component, is the direction of maximum variance. Now a quick word of warning. You see here, I've got this little symbol here that indicates that's where the mean of the points is. Principal components is mathematically defined about the origin. So what we typically do is we talk about moving all of the points to have their mean at the origin. But then we forget that it's doing that, we have to do it in the math and we would draw something like this and say you see this? This direction, that's the first principle component of those points. It's the direction of maximum variance. They're not around the origin here so we have to shift it actually before we do the computation but when we think about it we're only talking about with respect to the mean, and the subsequent principle components are perpendicular or orthogonal to the previous ones, and describe a next greatest amount of variance. Now, in two dimensions, we're sort of done, right? So, we say, well, that's the first one. Well, the next one's gotta be there, because there's only two dimensions so it's gotta go to that direction. But if I was in three dimensions, I would have my first dimension and then everything would be along there, so I could remove that, collapse it down on a plane, let's say in that plane, what's the maximum variation that will be my second one, and then I would have the third one that will be left.

3 - 2D Example Fitting a Line

This notion of maximum variance should strike a little bit of recognition in your mind about then notion of fitting a line. You remember we did line fitting? Okay. In fact we came up with this equation of saying we're going to measure the error that's essentially just the perpendicular error of the points to this line. And the, and the equation of the line that we had there was a x plus b y minus d equals 0. Or it could be plus d, doesn't matter. The idea is that a and b was the normal with respect to the line and in fact that's what that says right there. And if I take the dot product of x and y on that, it's supposed to be a distance d away and that's what this equation says. Alright, so that was when we were trying to do a least squares fit. So then we, we did this quick operation. We took the derivative with respect to the d, came up with this formula for d in terms of the average x, the average y. We plug that in, substitute d equals a x bar b y bar. We get this new error function E. Just having substituted. And we wrote it, this way. Okay? And this is where b is this matrix here. And you can see it's just the x's and y's in a row with the means subtracted. Okay? So what we want to do is we want to minimize that quantity, the magnitude of B times n, where n's going to be the normal AB that we're going to solve for, and we said last time that what we want to do is we want to minimize Bn squared, but remember, since the normal has to be unit normal we say subject to n equal 1. Now, what might be not so obvious because we didn't mention exactly last time is that this distance, these square distances, this is going to be the axis of least, let's call this the axis of least inertia. Right? [INAUDIBLE] From your physics. If I've got an axis, then the inertia of the

whole thing is proportional of all the points, the square distance from those, of those points to the axis. So the line fitting here in 2D is an axis of least inertia. And like I said, this should sound familiar to you. And we, remember we did this. We, we looked at several ways of doing this. We had those in fact back when we did calibration, we said well. We have this A, M, because we're doing M matrix for the calibration. We said make, let the norm of that, well, we want to make it as small as possible subject to M equal to one. And remember we were doing this thing with eigenvalues and eigenvectors, so it shouldn't surprise you that we're going to be revisiting eigenvalues and eigenvectors.

4 - Algebraic Interpretation

Let me give you another algebraic interpretation of principal components, and eventually this will just get us to our formulas. Here we have our point P, and let's supposed we have some origin. If we want to minimize the square of the distance from the point to a line, we want to find the line to minimize this square. Well, you remember Pythagoreans Theorem, of course you do, right. It's that the sum of the squares of the two legs of a right triangle are equal to the square of the hypotenuse. If we want to minimize the square distance of the point to that line we can, the same thing as maximizing the projection of that point on the line. That's what this value is here, right? X transpose P. X is some unit vector through the origin, and if we project P on to it that's going to be the vector and you want to maximize that length. So minimizing the sum of the squares, the distance to the line is the same thing as maximizing the sum of the squares, the projections on that line. A way of writing that is we want to sum up all the squares. And this is just a little bit of a matrix manipulation trick, right? If I wrote my points down here right, of x1, y1 or x2, y2. And if this was my line so it's maybe, it's, A, B, right, if I multiply x transpose b transpose, so now this is B, xy, xy, bx, all right? If you multiply that all through, what you'll realize is, it's exactly that value, okay? It's the sum over all the points of that square of the magnitude, all right? So. What that means is algebraically, this maximizing the projected distance, can be written like this, right. So our goal is to maximize x transpose, b transpose, bx. But we've got a constraint, right. X is a unit vector. So that means that x transpose x has to equal 1. So what we have to do is we have to maximize this subject to a constraint. All right, so that's what this says here. And I wrote is as m. Maximize equal to x transpose mx. Where m is b transpose b. So we just have a single matrix. Now, I'm not going to tell you a lot about this method. But I am going to introduce this quick notion of what's called Lagrange multipliers. Here is e and here is e prime. And you notice that what we added to e prime is this term, lambda times 1 minus x transpose x. So Megan, if everything is solved correctly, 1 minus x transpose x should be what? Zero? Whoa! Yes, zero. So, when this thing is solved correctly, it doesn't matter what this lambda is, right? Because it will add zero to the term. That in a very hand wavy way, that is the method of Legrange multipliers. It says, I'm going to put in a term Multiplied by lambda that will be zero at the correct solution. And then, you solve everything and you've satisfied your, Lagrange multiplier method. It turns out in our case it's even easier. So that's our e prime. So, what we do now is, we take the derivative of that new error function, e prime with respect to x, and we get this. 2 Mx plus 2 lambda x is the derivative. And, now, in order to minimize this we set this equal to what, Megan? Zero, Zero, very good. We actually rehearsed that one this time. And so when you do that, you get this very simple equation that says, Mx equals lambda x, alright. Forget the positive or negative, doesn't matter because lambda can go in either direction, all right? So that's a very special x, right? So when I multiply x by a matrix M, if I get back a linear multiplier, in this case lambda, of x, that means x is an eigenvector of the matrix M. So that means that x is going to be an eigenvector of this B transpose B. All right? So, this axis of least inertia. This solution where we minimize the distance to the axes. The sum of the square distances, which is the same as maximizing the distance along. Says it has something to do with this matrix, b transpose b, where b was that list of all those points.

5 - Yet Another Algebraic Interpretation

Here's another algebraic way of thinking about this, okay? Using our old definition of b transpose b, okay, you'll notice that what it is is actually this matrix okay? Sum of x squared, sum of xys, sum of xys, sum of y squared. Well that's just the covariance matrix of the points assuming that there are about the origin. So I don't have to track that the mean, that would be the covariance. Speaking of which, yet one more way of thinking about this is that B transpose B here is the sum of this xx transpose. That's what's referred to as the outer product. And if it's about the origin, I don't have to subtract off the mean, but the, here it is written just like that. That again is the definition of the covariance matrix. This makes some sense, for those of you thinking about it, the covariance matrix is a relationship between the expected value of the squares, the sum of the squares of points distributed about the mean. So that's actually, if you think about it, the saying what we wanted to, to know. We wanted to know the greatest variance. For those of you that care, or are thinking about, if you think about a covariance matrix as an ellipsoid in space your eigenvectors are going to be, end up being, the major axis of these ellipsoids.

6 - Eigenvectors

Here's a quick question for you. How many eigenvectors are there? Not on the planet, in general. Okay? Okay, well, normally for a real symmetric, and by the way, these things are ty, typically symmetric because if you take a look here, right? I've got the sum of the squares, but on the, off diagonal terms, I've got the same, values. The sum of the cross terms, right? So if I have a real symmetric matrix in general if it's sized end by end, I normally would have N real eigenvectors. And distinct eigenvectors. There are some degeneracies if you have identical eigenvalues and then you have, ambiguity We're not going to worry about that too much but just remember that normally, there is i say normally is later it's going to be a little different. Normally with the N by N matrix, im going to have N eigenvectors. So graphically, we can look at that like this. So here i have a beautiful set of points, aren't those delightful? Distributed in two dimensions and which way do you think is the principal eigenvector? Well, probably, something along like that. Right, that's going to be the axis of least inertia, right, or of greatest variance. Get used to this idea that it's greatest variance, and in fact, that's what this nice little picture shows us. And this is showing us lambda 1, meaning the first eigenvalue, large eigenvalue and what direction is the next eigenvector? Well as you know, it's perpendicular to that, right? Cause eigenvectors are perpendicular to each other, and go, and the next one is the next most variance, et cetera. With two dimensions, it's hard to show. There's only two here. Why am I telling you all this?

7 - Dimensionality Reduction

Let's talk about Dimensionality Reduction. So, here, I've got a bunch of points, okay? And these points are in 2D, all right? It's called kind of green and red, but that's sort of a joke. We're not going to worry too much about why it was done that. What you should notice is that in the middle here are these orangeish yellowish points. Do you see why it's R and G? Red plus green makes sort of yellowish, or so, so along that diagonal it'll be kind of yellow. I don't know who did this originally, but it's kind of cool. All right, if you think about these orange points, let's think about how they're distributed, okay? So, they have a have a mean sit somewhere, and they've got a. Set of eigenvectors, right? They've got a co-variants matrix that describes them. They have a axis of least inertia and then the next axis. So here you're seeing the points, the orange points are x, x bar here is supposed to be their, their mean. And then we've got the big eigenvector v1 and then the smaller one v2. And the idea is. We can represent those orange points by only their v1 coordinates, plus the mean. And what that would mean is haha. Essentially that we're going to think of all the orange points as just being on that line. And all I'm going to tell you is where on the line they are. And we're going to essentially ignore the amount that they're off that line. So we've just reduced the dimensions from how many? Two. To how many? One. Okay, that doesn't sound like that big a deal, nor is it. But in higher dimensions, this could be a huge deal. Imagine you've got something in

10,000-dimensional space, and yes, in just a minute, we're going to do a 10,000-dimensional space. If you said, well, I'm going to represent them by one number, or even 30, that would be a huge reduction. So if I say, well, what direction does it vary the most in? And I just give you that value. If that's good enough for what we want to do, you've reduced the description from being a lot of numbers to being a much smaller number. In fact, we can sort of express that here algebraically in terms of just, thinking about it, whatever dimension x happens to be in. So if I've got a whole bunch of data points in some N-dimensional space, what I want to know is the direction of projection. And we'll just say it's v. All right, that if I projected those points, after subtracting out the mean, that I'd have the greatest amount. Right, the greatest variation. And that's what that says here, right? So take x, subtract out the mean, dotted with the v, summed over all the x's, take the norm. Take the square. Well that can be written as an expression like this of just v transpose Av, where A is just this outer product. Okay, that's the co-variants matrix that we're, we're familiar with before. And as we said before. The eigenvector with the largest eigenvalue lambda is going to be the one that captures that greatest variation. In a minute I'll give you a little argument about why it's the largest eigenvector with the largest eigenvalue. Or you can just take my word for it. And, in fact, the smallest eigenvalue would be the least amount of dimension, so basically, what we're going to have to do at some point is take the eigenvectors of this co-variance matrix.

8 - Guess Principle Components Quiz

What if your data distribution looked like this? Can you guess the two largest principle components or eigenvectors? Let me give you some options. Do you think this could be one? How about this or these two? Remember, you want to pick the first and second principle component. Type 1 for lambda 1 and 2 for lambda 2. Leave the remaining two boxes blank.

9 - Guess Principle Components Solution

So did you fall into the trap? Let's look at the distribution again without the clutter. If you were to approximate this with an ellipse, what would it look like? Well, something like that. Which is the major axis here? That's right, horizontal. And that would be the first principle component. As odd as it may look, the second principle component would be orthogonal or perpendicular to this. So, this is one and this is two. Intuitively, this doesn't look right, does it? But it's exactly what you told PCA to find. The axis with the most variance would be the first competent and the one orthogonal to it would be the second. PCA is good for characterizing a single class of instances. It can reduce the effective number of dimensions need significantly. But, what if your data came from two classes or sources? If you wanted to characterize two or more classes at the same time PCA might not be a good choice. This is especially true if the axis of most variants for the source distributions are not perpendicular. There are some other approaches for characterizing multiple non-octagonal classes such as ICA or independent component analysis. One big difference here is that ICA does not require the components or basis vectors to be orthogonal. Anyways, this is probably going off track from the current discussion. Let's focus back on PCA and see how we can apply it to images.

10 - The Space of All Face Images

What does this have to do with images? Well, let me tell you. I'm going to introduce this using the notion of face images 'because frankly it was this eigenvector approach to doing face recognition that really, I think, got people's attention. Think of an image now, not as a two dimensional thing, but as just one great big long vector. Okay? So it's just got a whole bunch of values in it. So for example, suppose I had a vector that was 100 by 100 long. How many pixels is that, Megan? 100 by 100. 10,000. 10,000. You got it right. 100 times 100 is 10,000. So, that's just a 10,000 long vector. If I don't worry about the fact that they're pixels located in a particular position, which, by the way, is going to be an interesting problem later, if I just think of them as values, that's a 10,000 dimensional space. Right. There's 10,000 dimensions, each dimension has a value. What's the value? It's the value of the pixel that corresponds to that location. This location is another pixel. It's

value 18, fine, that's it's value here. So the idea is that an image is just a 10,000 dimensional vector. Question. How many of those 10,000 dimensional vectors, how many 100 by 100 pictures, are actually pictures of faces? I mean, just of a face? Answer. Not a whole lot, okay? In fact, most values, all right, if I just randomly put numbers in tho, that vector. Right. I get out, you know random pixels everywhere. Right? The idea is, that, faces are a very special set of vectors. They're not all possible vectors, and they're not distributed everywhere in this image space. They are on some particular, what's referred to as a subspace, sometimes a manifold, okay. They're in a subspace of all images. Okay? By the way, we sometimes call that subspace, face space. What's face space? Is the subspace in the space of images that are the pictures of faces. All right. So here's what we want to do. We and this is color coded so we want to build a low dimensional, because we're going to try to get a nice dimensionality reduction going on here, linear, we'll talk about the use of linearity to be able to just do dot products together coefficient, subspace, so it's going to be some reduced space, that explains most of the variation seen in face images. And that's just drawn out here, this looks just like what we had before. Where we had blue dots and, and orange dots. Now I'm telling you well actually, even though this is drawn as just a two dimensional space, 'because I can only draw two, it's a 10,000 dimensional space. 10,000. You like that. Okay, fine. 10,000 pixels and the idea is let me find the direction or directions in that image along which most of the face images lie. And that's what it shows here, that the cyan color is sort of, ideas that the face images are there, and then we've got all these other images that are not images of faces. They're cats, well they're all sorts of things, okay, whatever the internet has pictures of and all other possible things, all right.

11 - Principal Component Analysis Part 1

So how are we going to do this? Well, it's not going to surprise you. We're going to use principle component analysis because when you see a picture that looks like this, where I want the extended dimension, I want to use that extended variance. So here's how we're going to do principle component analysis for this. Assume we have M data points, okay? So x1 through xM. Alright, where they are in some D dimensional space. R to the D. Where D is big, really big. At least 10,000. Maybe it's a million, okay, so if I had a 1k by 1k image. That would be a million pixels, so I have a million dimensional space. So, what we want is some direction in that big space, that captures most of the variation. Okay? And furthermore, and this is sort of important. Not sort of important, this is really important. If I talk about a direction, and let's suppose u as my direction. How would I find out where it lies along that? Well, whats nice about this is because we're doing everything linearly we're just going to take the dot product of x minus its mean. Why are we subtracting the mean? because none of this stuff works if you don't subtract the mean. Remember, we're doing variation about the mean. So we subtract the mean and we take the dot product, and then u X of, u of X i that would be the coefficient. So we want to find the u that captures most of the variance. All right, well you know already how to do this because we've been talking about it right, so here's this same expression that we had before, but I sort of used the notation we just developed, right? We have, we're going to say, the variance along u. Well we want it over all the possible points. So what we say is okay, well we're just going to take that average. Here is x minus mu. Dotted with u, we multiply it also by its transpose. So that we get the square value. Okay, so that's just the same thing that we had from the previous slide, I'm just going to rearrange that a little bit. Right, so I can pull the U's out here, and the, the another U out there because it's transpose, transpose. And what I have left here in the middle, well what is that? Right, that's just the covariance matrix and we'll call it the covariance matrix sigmund. We've talked about how for finding this, the axis of, of, maximum variance, we need that covariance matrix. But think about this just a little bit more. So that variance of u is u transpose sigma u, all right?

12 - Principal Component Analysis Part 2

All right. Now, follow me carefully, all right. u is some vector. Which vector? u, not me, u. A matrix times a vector, since everything is linear, if I describe u as being made up of a bunch of

components that add up to u, it's the same thing as sigma times those bunch of those components. So in particular, the components could be the eigenvectors of sigma, so I could express u to be eigenvector of sigma point to whatever direction they point. I can think of u as being made up of the some of those, all right? So u is a unit vector. Okay, it can be expressed in term and it's what it says the linear sum of those eigenvectors of sigma. The question is which of those directions would return me the largest value, okay? Well, we know that sigma times an eigenvector is equal to lambda eigenvector. If I'm going to pick a single direction that will return the largest vector back, and then I'm going to take the, the dot product itself to get the magnitude, I would need the eigenvector with the largest eigenvalue. And at the heart of it, that That's actually why the eigenvector with the largest eigenvalue is the direction along which you get the greatest variance. So, you can either spend a little time thinking about that and understand why that's true or you can believe me. I guess I could have written it out more, but I think most of you, at this point know that I take the, the eigenvector, the largest eigenvalue of the covariance matrix, that's the direction of the greatest variation. So that's what this says here, that the direction that captures the maximum covariance of the data is the eigenvector corresponding to the largest eigenvalue of the data covariance matrix. Ta-da, that's what we're doing. So, furthermore, that's the longest one, I can take the top, I don't know, k of them. Orthogonal direct, and they're all orthogonal to each other. And that will capture the most variation that I can, of k orthogonal vectors. So I can say well I've got this 10,000 dimensional thing. Maybe I'll take the top 10, 20, 30, 40. That captures the, as much variation as I can with 10, 20, 30, 40 orthogonal vectors. So that's what I, I want to use. But in order to do that, in order to do PCA in images, on images I have to teach you a trick. And this is the PCA for d much, much, much bigger than m or n trick. All right. And so let's do the trick.

13 - The Dimensionality Trick

So here's the dimensionality trick. All right? And the notation that I've, I'm showing you here comes directly from the paper Turk and Pentland I'll mention later that does eigenfaces. But it'll be clear. Let's suppose each this cap phi i is a face image. So it's really long, all right, d long, and by the way lets assume we subtracted the mean. Okay, so we've actually removed the mean from all the pictures that we have, and so, we're, we've got the mean at the origin, fine. All right. So now lets define a new matrix called C, which is just the average of AA of phi of the sum of phi phi transpose. Okay, this is the sum of all the outer products and in fact this can be written as AA transpose if A, this is just what it says here, is just the stacked up face images, right? So A, as the stacked up face images, its going to be d, really big, by M, M is however many face images I have. So its huge by reasonable, that's what A is. And that means AA transpose, so if I take d by M times M by d, I get a what? d by d, all right, d by d, that is a really big matrix, okay? So if d was 10,000, this would be a 10,000 by 10,000 matrix. How big is that, Megan? A lot. That's a lot, exactly. That's a, a, a, a hundred million. Okay? So if it, if it was a million long vector, a d by d would be a million times a million. Which would be a million million, which is a big number. That's a thousand billion, which is a trillion. That's a lot, you wouldn't ever want to do that, okay? The idea of trying to find the eigenvectors of a ten thousand by ten thousand matrix is painful, even on today's computers. And of a million by a million matrix, is even more painful. But fortunately, you don't have to. So C equal to AA transpose is a huge matrix. But now, instead of considering AA transpose, consider A transpose A. That's only an M by M matrix, right? because remember, A is d by M, so A transpose A is M by M. Finding the eigenvalues and eigenvectors of that, pretty easy. Well, suppose I've got some eigenvector v i that's an eigenvector of A transpose A. So if it's an eigenvector of A transpose A, that means when I multiply A transpose A times v i, I just get some multiplier, lambda of v i, the eigenvalue. Premultiply that equation by A. When you do that, now I got AA transpose, and I made this blue to make the difference clear. A v i, and by the way I just pulled the lambda out on the right hand side, it's just a scalar, I can do that. Look at that equation, what do you see? What do you see? What do you see? What you see is that the A v i's are the eigenvectors of AA transpose. Remember AA transpose is that big, ugly matrix, C? So all we needed to do is to find the v i's, which are the eigenvectors of the A transpose A, premultiply them by the matrix A and I now have the eigenvectors of AA transpose. All right. So what this basically

meant was we took the we took the eigenvectors of an n by n matrix and created The eigenvectors of a d by d matrix, without having to explicitly compute the eigenvectors of d by d. This is much easier to compute.

14 - How Many Eigenvectors Are There

Quick question. How many eigenvectors are there? If I had more than D images. Okay. So D is 10,000, 10,000 dimension, let's say, or a million. So if I had, you know, 10,000 images and I was looking at all those, I could get 10,000 eigenvectors. But clearly the sum of those outer products. If I've only got a small number, right? The whole idea is that M is going to be a lot less than D. So how many of them are there? Well your intuition might say, well I've got M independent things, so there must be M of them. But actually your intuition was wrong, and in fact, a better intuition would be the following. Suppose I gave you two points in three dimensions, okay? How many vectors can span those? Just one, right? Wo, I go between them. And the reason is, I have to think about subtracting off the mean. Right? And once I subtract off the mean, actually I only have one degree of freedom, right? Because the origin is going to be halfway between those two points. So once you tell me where one of the points is, I know where the other one is. So there's only one degree of freedom. So with two points, there's only one possible eigenvector. Let's think about it this way. If I have three points in space, okay? How many vectors does it take to span those points? Well, they lie in a plane, don't they? Of course they do. Three points to find the triangle. So with three points, there's only two eigenvectors. In general, there are M minus 1 eigenvectors. And the reason there's M minus 1 is, I take my endpoints and I subtract out the mean. So that removes a degree of freedom. So that's a trick question that I always give on the final of my course. So if anybody's taking the final, now you know the answer. Okay, there's M minus 1.

15 - Eigenfaces

Let's apply this to face recognition and it's referred to as using Eigenfaces and it comes from this paper by Matt Turk and Alex Pentland, Sandy Pentland. It's Alex Pentland, but his name is, he goes by Sandy. They were at the MIT media lab at the time. And one of the reasons this paper, besides the fact that it works so well, is just known so well, in my humble opinion, is because of this term, Eigenfaces and face space. What a great, what a great phrase. In fact, I can show you the Eigenfaces, I will in just a minute. All right. So, here's what they do. They assume that most face images are going to lie on some low-dimensional subspace in the big image space. Determine by, I don't know, let's say, k eigenvectors. Okay? K directions of maximum variance where k is going to be way smaller than d. So, d might be 10,000 or a million. K is going to be like 20 or 200. Both of those numbers are way smaller than 10,000 or a million. So, what it does is they use PCA, like I just showed you, to find the vectors, or what are called the Eigenfaces. We'll look at them in a minute, u1 through uk, that span, that subspace. Okay? So, you take all your images, you find your eigenvectors. And now, what you're going to do, and this is the really cool part, is you're going to represent your face images in that data set as just the linear combination of those eigenvectors. Or another way of saying it is, I'm just going to have the coefficients of the 20, of the 20 eigenvectors. If it's 200, 200 eigenvectors. And I'm going to represent my entire image, all 10,000 numbers, by just the coefficients of these eigenvectors. And I multiply those coefficients times the eigenvector, sum them up, that would be my new image. All right. So, it's a tremendous data reduction. So, take you through an example here. So, and some of these images come directly from the old papers, some from some newer work, but makes the same point. So, suppose I have some training images X1 through XM. So, here's a picture of faces. Now, notice, it's just the cropped part of the face. So, they're going to try to do recognition just on the cropped part of the face. So, the first thing that's kind of cool to look at is this is the mean image, mean face. Not the mean face, the mean face, the average face. Now, nobody wants to have the average face. Too bad. Whoever that is, he or she has the average face. Why did I say he or she? Well, by definition, it's hard to tell. Right? because it's an average over everybody. There is some interesting work done on showing how average faces appear more beautiful than real faces. This was done a long time ago. It's just a study of the

appearance of beauty, anyway. It was interesting image processing approach to thinking about, could you predict when a face would be deemed beautiful. All right. You can imagine here's a question. Show a picture to a computer, face image, part of the digression, and say on the average, on a scale of one to ten, how likely are people to rate this thing as beautiful, as this person is beautiful? And could you do that? I think these days using machine learning might be a different approach. But at the time, there was work done looking at average faces. Sure you wanted to know that. Anyway, oh, and so what do you do with the mean face? Like what you should do with a mean face, you throw it away. Well, what you really do is subtract it out of your population. Isn't that great? We subtract all the [LAUGH] mean faces out of the population. No, we subtract the mean face from all the, and what we have left is our distribution. And then, we start computing the eigenvectors. And here they are. Okay? So, here's some top eigenvectors, and one of the things you'll notice right away, fact, let's take a look at this first eigenvector. Actually, the, the second one. Is that it's much brighter on one side than the other, because some of the variation might just be due to the lighting coming from one side or the other. But later, you'll have these different eigenvectors, which look like these kind of ghostly images. But these are eigenvectors. So, they look to you like pictures and they are, but what they are is they're a 10,000 dimensional vector, which is just an image. Remember, I can go between the image space 10,000, right, and the images, so I'm just showing you these eigenvectors as images. Okay? And if I want to take a dot product of a real picture and these? That's easy. All I do is overlay them and multiply. Multiply every pixel and sum them up. because that's what a dot product is. So, when I show this to students, sometimes they don't understand how come the images are eigenvectors. And that's because in image space, that's what it means, a vector is an image.

16 - Eigenfaces Example

So it's a little bit easier to understand what these eigenvectors do, if we look at it this way. Here are the principle component eigenvectors, one through whatever. Here's two different reconstructions, all right. If I take the average face, and I just add three times, and sigma k is just the standard deviation of the coefficient, don't worry about that. If I take the average face and I add in some of that component, I get this kind of thing. And if I subtract off that kind of component, I get that thing. Notice, that that just moved the lighting from the left to the right. So one way of thinking about it is that these components, the first components just change with the direction, lighting direction was from the left to the right. Whereas, let's see, let's take another one. Here, this component here, if I add positive, I don't know, it tends to look a lot more feminine. When I go negative, it tends to look a lot more masculine. Okay. So each of these, different eigenvectors are adding in some other different variation, some other different element. So, how do you use these? Well, a couple of things we have to talk about. First is, I have to be able to get the coefficients. Well, that's real easy, just like I said before. I take in some image, x, here it is, and I just subtract off the mean, the mean face. And I take the dot product with, in this case, the first eigenvector, second, third, all the way up to the kth eigenvector. Those dot products will give me these coefficients, or weights, w1 through w k. What's cool is that vector of numbers, just k of them, maybe it's 20, maybe it's 200, that's my entire representation of this face. So I've taken this 10,000-dimensional vector, 10,000 elements, 10,000 pixels, and I've reduced it to 20 numbers. So how do we make use of this vector, these w's that are the coefficients? Well, the first thing is, I can actually use them to reconstruct the face. Right? Each one of these w's tells me how much of each eigenvector to add back in. And in general, what I can do is I take a picture and I say this picture is represented by the mean plus the w's, each of the w's times the eigenvector, because that's what this is. And if I were to keep a lot of them, I would get a really great reconstruction. If I keep some of them, I get an okay reconstruction. Well, how many of them do I want to keep? Well remember, I wasn't doing this to do a reconstruction, I was trying to do recognition.

17 - Recognition with Eigenfaces

So how do we do recognition? Well, actually it's pretty straightforward. I have some new novel x comes in, some new novel face image. All right? All I do is I project it into the subspace. What does that mean? Just like before. I subtract off the means and I compute the dot product, the coefficients W1 through Wk. By the way, I can then do something that's optional put here. Supposed I'm not really sure that what you gave me was a face image. Well, what I can do is, I could reconstruct using those coefficients. And now here's the thing. If you gave me some other 10,000 dimensional vector, right? You know, three guys on a bicycle. And I try to make that image by just summing up a hundred eigenvectors from the face space. How well do you think I can reconstruct that image? Not so well. The only images I can reconstruct well are faces. So I could do the reconstruction and compare them and say, aha, bad choice. This is not a face image. It also means, by the way, if I'm looking for faces in a picture, I might do that. In fact, in the original, Turk and Pentland, they actually did detection that way. There are now better ways to just detect raw faces, but the ability to reconstruct is a way of knowing that yep, I actually have what I was looking for. I'm going to use, to see if I can restructure, do some tracking, but hang on, that's for, that's for the next lecture. But for the recognition part, it's optional. Let's assume it really is a face. Well, then, all we can do is, we can say, all right. You gave me these W's, right? 1 through k. Let me find in my list, the one that's closest in of, of those k numbers. I might use a Euclidean distance, some other distance metric, but basically, I have a description of a new face coming in, and I just see which of the descriptions in my database is closest. And that's why you can think of this somewhat as a generative model. There, there's actually more detail to use it in a more probabilistic way. We'll actually talk a little bit about that in the tracking. But the idea is that, each one of these descriptions gives me a probability if you will. Or likelihood of what this person's face might look like, because it assumes that this has captured most of the variation, and it, the other stuff doesn't matter too much. And if I, if a new one comes in, I can just compare them. And if I get closer, that's more likely, and that's why it's a generative model.

18 - An Old Cast of Characters

So when this was first put together here's an old cast of characters. And there's two funny things for me to tell you. The first is, you talk about, you know, missing a trademarking opportunity, right? So when Matt Turk and Sandy Panaro were putting this together. They had in mind this notion of mug shot books. You know, mug shots, where page through this book of all these face images, and you say, aha, that's the person. So they were putting together something that was going to be kind of like a mug shot book, but of faces, and it was going to be an electronic program. So what do you think they called this electronic program that was supposed to be a book of faces? Photo book [LAUGH]. So, you know, there you go Zuckerberg. Hope you're happy. Facebook was invented and called photo book and so then Facebook could be Facebook. So that's humorous story number one. Humorous story number two is actually not humorous at all. It actually speaks to, sort of, the productivity of the media lab. The vision and modeling group at the time. This, old cast of characters here, bunch of these people were students, in fact almost all of these people here were students at the time. Oh, and there's Mike Boe. He was a research scientist. But, some of you may know Astand Sclaroff, who was chair of, Boston University Computer Science. Well, that's him right there. Some of you, ever heard of G Plus? All right? So Bradley Horowitz at Google was instrumental in making that happen. He's right there. I think that's him or is that him? Those two may be the same person. Okay? Anybody up in Berkley? Trevor Darrell? Computer vision researcher, or here at Georgia Tech, there's Professor Irfan Essa. Neither he nor I look like we used to look like when we were at the media lab at the time but that really is him. So it just tells you a little bit about sort of the density of interesting vision research going on at the time. But basically what they did was produced a database of eight thousand faces and it was able to do recognition.

19 - Limitations

You think it's perfect. Of course, not. There are some problems. One problem is, remember I told you that the way you do the dot product, since these are images, vectors, you can just line them up and multiply them. That was the dot product. That's how you take the dot product of these two 10,000-dimensional vectors. That should have made you wonder, well what happens if it's misaligned? Right? The eyes aren't exactly in the right place. Okay. The scale is a little bit different. The answer is, it doesn't work very well. All right. It really requires good alignment scaling. Also the background has to be not too, too terrible. That can be done but it is a, a sensitivity of method. Another cute little story. At the time they had these 8,000 faces and they wanted to put them in a scale, in a, in a scaled lined up way. And so, you could imagine trying to write a computer program that would sort of find where eyes were, and then line them up, or you could pay an undergraduate. So, they paid an undergraduate to click on the pupils of the eyes of each picture of the 8,000 pictures. I think his first name was Wasi, I cant remember. If you're out there, my apologies. And what was great as Sandy will tell you is as a student who's working on these 8,000 pictures, he came back and said, I have a new, new appreciation for what order of magnitude means. Because the difference between doing 80 pictures and 8,000 pictures as you can imagine took a long time. But you basically you have to line this up. Now, You can use methods to automatically align the eyes its scale, et cetera, and then that kind of thing will work. Another limitation that is sort of of a more theoretical basis for PCA in general, Principle Component Analysis. Typically the way you do it is you take your entire population, you find the, the direction of variants over the whole population. You project down along those dimensions and then you try to discriminate between the different classes. In this data here, we have blue points and we have red points. But, if the first thing you did was find the axis of elongation. Okay. The axis along which there is the most variation. All right. And you projected these points down, well clearly, you will confuse the blue points and the red points. All right. So as it happens to be that the way of separating the classes. Right? So the direction along which you want to separate the classes is not the direction along which the, all the points are distributed. There is methods in comp, in pattern recognition Fisher Analysis, Linear Discriminant Analysis, for using not just the structure of the population, but the structure of the classes. To determine which way to separate. We won't talk about them here because it's kind of an old method. But it actually talks more about the discriminative methods that we'll talk about not next time, but the time after. The idea is saying that, well I don't need so much to model the, the points themselves, because the points are elongated this way I need to model the separation between the points. So a limitation of PCA is that sometimes the direction of maximum variance is not always the best direction to project on for doing classification.

20 - End

That ends the lesson on PCA. It's a very general method. People use principle components all the time. And the face recognition method is just what we use to introduce it. Although I will say, it's become very well known because, not just because how well it worked and caught, sort of caught people by surprise, but just it had this very clean interpretation. In fact, now people, you'll hear, instead of eiganfaces, you'll hear about eigangaits. Right? So you, you take measurements of people walking, you try to find a principle component analysis, project them down and do it. There are eigan body shapes, which is terrifying to think about, but basically the idea of, I've, you know, people vary in different shapes, and I should be able to take a small number of variations and say I can fit you by just a small number of control knobs. If you're doing computer graphics animation, and you want to be able to control the different shapes of people to build different characters, you want to be able to just turn a small number of knobs and have people change their overall build, their height, and things like that. And you can use this, principal component analysis, or more sophisticated versions of it, to do that. The method that I've shown you here also is a linear method. Right? All these things are orthogonal. And I could compute the coefficients by just taking the dot products. And that's a nice property to have, it makes things easy to do, but it's also a liability. If I have a, just say, a three-dimensional space, suppose I have points that actually just lie along a curve,

okay? So they are all on a single curve, which means if you just told me the value along that curve, that would represent the point completely, assuming I knew the curve. But it's not a line, it's a curve, okay, so it, it bends. So there are non-linear methods of doing principal component analysis that allow you to capture, sort of non-linear, lower-dimensional manifolds. We won't talk about it here, but, in fact there's one's called non-linear kernel PCA. But for now, all you need to know about is what we talked about here, linear orthogonal decomposition of principle components.

8B-L3 Appearance-based tracking

1 - Intro

All right. Welcome back to Computer Vision. Last time, we introduced principal component analysis as a way of doing dimensionality reduction, going from a high space to a low space. And we said it only works if the signal actually is present in that lower dimensional space. In particular, you can only reconstruct an image well using a small number of low dimensional. A low dimensional small number of basis images or eigenvectors if the image itself was kind of like that, right? I can only make faces out of eigenfaces, I can't make bicycles or tortilla chips. So that gave us a little insight, as you might use that to do detection, right? So if I want to find a face, I might look for patches that can be well described, reconstructed, using just that low dimension. So, there's a related task that we can use this method for. And that's what's called appearance based tracking all right? So here's the basic idea, learn something about what a target looks like. Maybe not just a single image but the possible variations by capturing a low-dimensional space, being able to reconstruct. And then you're going to track that target around by finding the place that you can reconstruct the best, all right? So that's the basic idea and what's cool about this is instead of like some of the previous tracking. Where if you learn new appearance every time step, you might drift that is you eventually get stuck on something that's not the object. Here you might be more robust to that.

2 - Visual Tracking

So, as I mentioned in sort of a conventional or traditional tracking approach, what you do is you build a model before something starts moving, right? So you might cut out a patch, find a contour, some method of describing the object. And then you do one of, remember we talked about optic flow where you essentially connect the dots. You do patch tracking with particles, which if you're in the OMS class, you did as a problem set. Or you might even solve some interesting, complicated, optimization problem. And then you somehow try to incorporate some invariance. So you might use edges, because you know that those don't change a lot depending upon how the illumination changes, let's say, all right? But the problem is, if I'm going to track an object over an extended period of time, the object and the environment tend to change. And what I would like to do is I would like to not just track the object but in some sense track the changes, incorporate the changes slowly as the object moves. So some work that we're going to talk about and these are some slides taken from, from Yang. Who showed a piece, who did a piece of work that I'll talk about along with Ross and Edal they took some inspiration from two places. First of all Michael Black and Allen Jepson way back in 1996 did something called Eigen tracking, so you guys know all about Eigen faces. Well, Eigen tracking was the same idea but the idea is so, here we have Coca Cola cans and here we have these Eigen images, up this one's a 7 Up can. How did that get in there? Okay, we have Eigen images of these different objects and the idea was we're going to track this thing by finding the places we can represent well by using those Eigen images. But one of the cool things they did was, they said, you know ,remember what didn't work so well in eigen analysis, the eigen face, was the alignment? Things had to be aligned perfectly? Well, suppose we decompose the problem into the geometric part. That is, how is my object moving, maybe rotating, scaling, to what does it look like. Now we've already seen this, right? When we were doing the good features to track, we talked a bit. In fact, there was an image of of a speed limit sign getting bigger and

bigger and it was being tracked and the idea was it's the same image patch, it just deforms. Well, there they were using a single image patch. In the eigen tracking, the idea was that we're going to look for a deformation. So could be affine, could be otherwise, where we got, where we have remote translation, rotation, scaling maybe even shearing. But now instead of just taking a single patch, we actually have a set of eigen vectors that describe the coke can. So for example, maybe if I rotate the coke can a little bit or maybe some, some other change, I could represent all those different appearances. When they did it, it was done in sort of a non linear optimization way and they, but more importantly they took a fixed basis set. So it took a whole bunch of images of coke cans, created a basis set, said that's our Coca Cola basis set. So we're going to need to do some improvement on that. The other inspiration here is also something that we've covered, which was particle filtering. Remember Isard and Blake, they instead of having a single proposal the idea is that you'd have a bunch of particles and each particle would represent a possible location. And that allowed us to handle sort of non parametric densities, right? Use the sampling based approach and there what they were using were like contours, right? And they would propagate this over time, they would, they would move the particles forward.

3 - Incremental Visual Learning

To do incremental visual tracking using principal component analysis, what we want to do is you want to build a tracker that is not just based upon a single patch but sort of constantly update some form of a model. between me and you, that model is going to be a basis set, so were going to keep changing the basis set a little bit as the, as the model moves. We want something that runs quickly and can operate even if the camera is moving. And, of course, the, the general challenge here is, we would like this thing to be able to handle if the pose is changing, so how the thing is moving, you know, rotating, translating. Maybe parts of it get occluded, okay? And we'll talk about how to handle occlusion in just a bit. You have illumination change. And you want to avoid drift, that is, you don't want to start, you know, tracking someone's face and all of a sudden you realize you're just tracking the bottom half of the face and then it just gets stuck on a file cabinet. Right, you want to stick to their face. So the paper that some of this is pulled from is this incremental learning for robust visual tracking. And I'm just using this. I mean it's, it's, at this point known well. And also, it gives you a sense of how to use principle component analysis, which we just learned, particle filtering, which you know, in order to do incremental tracking. This doesn't really belong in the recognition unit, but since we couldn't do it until you knew about PCA, here it is. So here are the main ideas, all right? First, we are going to use some form of a particle filter, but the particle, are going to represent the deformation. And the deformation may mean just an a fine that is six degrees of freedom. It could be similarity, it could even be just translation. But the idea is that the particles are going to capture how the geometry changes from frame to frame. The second main idea is that, we're going to use the subspace method that I talked before about, detection, right? So, the idea is that we have some small set of eigen vectors, they can reproduce only images that kind of look like some thing that comes from this class. And we're going to use that subspace in order to say, ha! That's a good example of the thing I'm tracking, because I can reconstruct that image really well. The third thing, which is kind of the new, new and cool part is, that we don't fix the subspace. We let it change incrementally as we get new it, so we, we track this image, we track this object, we bring in the new image and we might say, oh. Maybe it's changed a little bit. Let me change my subspace just a little bit. So by changing my subspace a little bit, I can track changes in appearance but it's because it's an entire subspace. I don't get fooled by a sort of small variation that's just local in time. And I can sort of hang on to the object. You'll see an example of that as we go forward.

4 - For Sampling-based Method

So just the nomenclature that we're going to use here, and this is taken directly from the rosetal paper. We have a couple things. First of all, there's the actual state, and they use L for location, but L is not, maybe not just be the position. It might also be the scale, rotation, etc. And then they've got an observation that's called F, and you know why it's called F? Because what we were actually

tracking were faces. But just think of it as the observation, we're actually talking about them as being Zs previously. So the goal is to predict the target location based upon our last belief about where it was and our current new measurements, and in fact, what we need to be able to evaluate is this probability of the location given my last belief plus my new measurement. Now, we've done this before, right? We've said that this equation was made up of two components, and we'll just copy them back up here. The two components are the, we'll do the purple one first, that's just our dynamics model. That says, if I know where thing was at time T minus one, what's my belief about where it might be at time T? Now in their work here, they used, the fancy word for it is Brownian motion. For those of you remember, that's little particles just vibrating around doing whatever they want. Another way of thinking about it is, they don't have a velocity model. They basically say, if something is here In the next frame it's going to be somewhere around here and maybe rotated a little bit, and it doesn't remember anything about velocity. So, it's just based upon position. But the new part that's kind of cool here is this observation model. This observation model is the probability of the observation given the, the state, given the location. And what they're going to do there is they're going to use that eigenbasis approximation. So what that means is, I'm going to say, if I can reconstruct this patch well from my basis set, that's going to be a high probability. And if I can't reconstruct it well, that's going to be a low probability. That's fundamentally the observation model that they're going to use.

5 - Dynamics Model

Real quick the state model l of t for location. So, you can use just x and y if you think it's just translating. You can use the similarity transform which is x y rotation feta and a scale, that would be four parameters. Or you might do a affine defamation if you're going to allow things to skew a little bit. It doesn't really matter. You just have to have some model and some way of saying I'm going to be able to represent these elements. What's my dynamics model? Well my dynamics, dynamics model that they use was actually very simple. They said, and this is just written P of L1, given L0, you could write it as T plus 1 given T, but then I gotta write all these T plus 1s, all right? And what you see here is a bunch of independent normal distributed variables, right? So, what this says is that the x value is just normally distributed about the old x value, with some standard, with some variance, sigma squared x. Same thing with the y. Same thing with the theta. Same thing with the scale. So this is a similarity transform. What is says, is nothing about how I move an X tells me anything about how I moved a Y. That's pure Brownian motion. So that can be illustrated like this, right? So here's Lt, this is where I think the thing was at anytime t and what this Brownian motion says well Lt plus 1, maybe I'm right near by, maybe I'm at the same place but, oriented differently, maybe I'm over here, maybe I'm over there, maybe I'm over there, maybe I'm over here. With a likelihood of being in any of those locations depends upon how far I've moved. That's what these things here say. Okay. So in other words, moving further away from my old Y gets less and less likely. What you can do is you can imagine distributing a whole bunch of possibilities, and that's what's illustrated here is Lt plus 1. And, the idea is that the farther Lt plus 1 is from L, from Lt, in terms of x, y, theta and scale or any of the other variables, the less likely it is. So that's our dynamics model.

6 - Observation Model

So now we need our observation model, okay? So we're going to use, it's, it's referred to as probabilistic PCA but we're basically going to use PCA. And given some possible location, we have to evaluate that observation as saying, well, how likely is it that that's the right place? And what it's going to mean is, how well can I generate that? And the math that was done here, this is the math that sort of justifies this idea of distance from face space, all right? So, here's the idea. Let's suppose I have some eigen vector subspace. This is face space. And that's written as B here. Okay? And so that's just shown as this plane in 3D. So the idea being I'm only using two dimensions, let's say. So I'm assuming everything's in a plane. And furthermore, it's got some mean value there. And in this plane is in face space. All right? And what this little ellipse shows you is, this is sort of the

probabilistic distribution of how the faces fall within that space. And now I have some observation z here. What this equation is saying here, is that the probability of getting some point z, our observation, all right? Is distributed about the mean, okay? So then what's the most likely face? The most likely face is the mean face. The mean is the average, right? That's the most like, that's the center of the normal distribution. And what this equation says is that, the probability sort of breaks down into two parts. There is a distance that has to do with sort of your, your co-variance or your variances in the face space, and then there's a distance that's sort of out of face space. The amount of difference from the reconstruction. So that's written like this, where the idea is that this BB transpose, that's the co-variance matrix sort of within face space, and then we're going to just say some epsilon I, I is an identity matrix, epsilon is some small number. So things get to come out of that plane a little bit. Okay? So faces are mostly in face space and they move away a little bit. Now what happens is, as epsilon goes to zero, or just as epsilon becomes small, or another way of saying that is, that the variance within face space covers most of the faces, and there's only a little bit of variance left. What this says is that really, you can think of that as this variance becomes bigger and bigger, that the only thing that matters is how far you are out of face space. So it's mostly a function between the point z that you've observed and the face space that's spanned by these vectors. So the math behind this can be written in the following way. What this says is, that p of z given B. So the probability of some observation given face space, is an exponential, so it's going to be decreasing, as a function of the square of this term. And what this term is, is it's the difference, so z minus mew is this, okay? Subtract off the reconstructions. So this little blue thing is the reconstruction. So if I subtract that off, all I get is this distance to face space. And so what I'm going to do is, I'm going to evaluate the distance to face space for the reconstruction. And later when we're tracking something else, it's going to be the distance to whatever my eigen space is. So a question is, why is that term the reconstruction? Why is that term the projection of z onto that plane. Well, it's actually pretty straightforward. So if B is D by K, all right? B transpose is just K by K by D, all right? So what is z minus u? Z minus u is just the vector of the new observation minus the mean. Okay? So B transpose times z minus u, that's just the dot product of all the eigen vectors with z minus u. That's just the co-efficient vector. So this thing right there, the transpose z minus u, that's the co-efficient vector. And we can call that gamma. All right? Well, given the coefficient vector I can reconstruct. How do I do that? I just multiply that, I multiply B times the coefficient vector, that will be the, each coefficient times each of the eigen vectors summed up. So that's the reconstruction. So when I'm all done, this z minus mew minus the total reconstruction gives me just the amount that you're out of that plane. That you're out of the face space or the eigen space. Another way of saying it is, that's how much of the signal you cannot reconstruct, using the vectors that are just in that basis set.

7 - Incremental Subspace Update

So far it's really pretty much been the tracking like we saw before. But now the part of filter tracking, we've got a model tracking it. But the thing that we're going to do now that's a little bit different, this is sort of the cool part is. We're not going to assume that our appearance and observation model remain constant. Basically the idea is we're going to, sort of, update our, the model as we get new images coming in. Now a sort of a standard way of doing this, and, by the way, this works really poorly so don't do it is. I have a model, appearance, I look around, I say that's my best one, and I say, okay, I guess it's changed a little bit, let me pull out that model and I'll find that in the next image. And, eventually, what happens is you get drift. And the model lands on that. There's this really, really cool thing. Oh, you can't see. Okay. I'm going to pretend like there's this really great car. Sitting right over here and you know, eventually and it just gets stuck on there. And the thing that I was tracking would go, it would go away. But if I don't update my model then as things change, I can't find it so well anymore. So what we do here instead is with every new appearance, we're going to change our eigenbasis. Right? This is a, this is the set of eigen vectors that lets our It says reconstruct. We're going to change that a little bit. And that allows for a small variation in appearance, but it still going to try to do things like track the general class. And so the trick is just as it is written here. Given the basis set, B of t minus one, that we use to find the loc, the observation at time t minus one. We, that allows us to compute a new eigenbasis, B of t, that we're

going to use now to evaluate our measurements. All right? So the idea is we're updating this eigenbasis, but it's adding sort of it's a small variation each time, so we can't radically change what things look like. We can change them a little bit. And this referred to as an incremental subspace update. As you remember from PCA we said the whole reason that we're doing this dimensionality reduction is the idea that we believe our images lie in some subspace. But we have to find that subspace within the bigger feature space. So that's why this is referred to as an Incremental Subspace Update. And why are we doing this? Basically, we are accounting for appearance change. So we might be tracking the, the object in the image. It's translating, it's rotating, it's maybe shearing, but if I rotate out of the plane, as far as the camera is concerned, my face is changing its appearance. As I go sideways, it usually goes worse, but that's okay. The idea is that I want to model that change. So we're learning a new appearance representation as, as the image area changes. In the papers that I've been telling you about, they base it upon, on this recursive SVD algorithm, which is kind of nice. So you don't have to do the whole eigendecomposition, decomposition again. So you can do this quickly in real time. That part doesn't matter so much. Essentially what you have is an algorithm, that allows you, it's kind of like a running mean algorithm, where you're using the most recent examples more strongly than the older examples.

8 - Put All Together

So putting it all together, so maybe you construct the initial eigenbasis as your way of, of doing this, this tracking. You pick some initial location, remember, we had some prior location. From that location, you generate possible new locations. You evaluate those possible new locations, and then you select the most likely one. Remember we're not taking the entire distribution, we're just taking the most likely one. And then, in the incremental part, we're going to update the eigenbasis using that new detected location. And then we just go back up and go over again. That's why it's called tracking.

9 - Experiments

So, show you results of some of their experiments since they have the results easily available on the Web. Now, when this paper was written, which was a while ago, they were running 30 frames per second on a 320 by 240 image. So now it can go faster because time passes and machines get faster. They are also only using 500 possible samples. That's 500 possible new proposals. Now maybe you can do 5,000. They use 50 eigenvectors. Maybe you could use more. It is the actual numbers don't matter yet. The goal it was nice and it was kind of in real time. In fact, that's why I put down it runs at XXX frames per second using Matlab. Where XXS is, XXX is going to be more than two and less than 90,000, so it's in there somewhere. It depends on how fast your computer is. So here's some results that they show you. Now up here this is the main image, and you can see they're tracking the face there. What you're going to see here is going to be mostly about where the image is being explained best. We're going to talk about that more when we talk about occlusion. And what you're going to see down here are about 10 of the, of the eigenbases, right? And you're going to see that they're changing as the appearance changes. So if I press, there we go. All right. And so you can see that it's doing a pretty fine job of explaining the the face. Sorry, of tracking the face. That middle row that I was talking about, that has to do with what am I seeing versus how well can I reconstruct it from the basis set, and it's using that as its evaluation. And you can see, with really large lighting variations, it tracks really well. You saw it got stuck there at the very end, and that sort of, as, as far as it could go. Yeah, here's another example where you're going to see a lot of sort of change in orientation and change of appearance, and this is Sylvester the cat being tracked. And you can see that it creates a lot of really sort of creepy lookingSyl-, Sylvesters on the bottom. Those are the eigen, the eigan cats. The images of, of the thing, and, and that's how it's doing its tracking.

10 - Occlusion Handling

So let me show you something that, that same group and others pushed on a little bit more recent of handling occlusion. So the question is, you know, if, if a thing gets partially occluded, we have this problem, because you remember the Eigenface stuff is, it's capturing the entire image. As a vector, and all the pixels all the components in that vector are created equal. So if some of them are sort of being blocked or obscured, it's not going to be able to explain that data, it's not going to be able to track, give you very high probability at that area. But it turns out you can, you can address that if you think about the following, all right. By the way that's Greg Dudack a while back, hi Greg. Picture there. When I reproduce an image, so let's suppose I've, I've tracked this thing, I've completed my Eigenbasis and now I, I'm trying to find my best one, and I do. I can now say of all those pixels, which ones am I reconstructing well? The ones that I'm reconstructing well, those are the ones that I have a lot more confidence in. So let me create a weight mask that says, you know, those pixels are fine, but anywhere that I find something that I can't reconstruct well, I'm going to highlight those pixels as ones that I'm not going to pay attention to. By the way, the reason I'm saying highlighted is on the video I'm about to show you, for whatever reason, they have, sort of, bad pixels as being bright. So you can think of them as being highlighted for not counting as much in the current or the next iteration. So what you're going to see here in this video again is, so here's the, Greg's face. That' s going to be tracked. And this is the, sort of, the location of what they're doing. But what you'll see here is, there'll be this black image, and then, all of a sudden, you'll see his hand swipe up across his face. Now, of course his hand is not easily reproduced with the face Eigenbasis. So you're going to see that light up, and that's it saying okay, the weight of those pixels should be reduced. So let's see if this works. There it goes, its playing, okay. Same thing we see before. You see his Eigenvectors changing on the bottom as he rotates in and out of the plane. Whoops! There was his hand. Did you miss that? Let go back again. Now get ready, in a minute, he's going to swipe his hand over his face, there, there it goes. Did you see that? You can play it backwards, as his hand came up, the weights got brighter where the hand was, saying ignore this, this place. And now it tracks him for awhile. He's smiling, he's getting large, and moving around and you can see it's doing a good job tracking. You can see the difference between how well it can be reconstructed. And the reason it can handle things like this is remember the state that's it's tracking is position translation rotation scale. So you can normalize it back to the original sort of canonical view and explain that view from the Eigenbasis. All right, so that's some pretty cool tracking work.

11 - End

It's an interesting example of using PCA, the stuff that we just talked about on image patches and it's funny. Like I said, I couldn't show you this during the tracking part that we did because you didn't know about PCA yet, or you certainly didn't know about it from me and this really wasn't recognition, but I thought it fit in well here because now that you understood PCA as a low dimensional representation, we can use that for a lot of things. So. Now we get back to recognition. Actually next time we'll get back to recognition.

8C-L1 Discriminative classifiers

1 - Intro

All right, welcome back to Computer Vision again. We're going back to recognition now. Remember, before we were talking about supervised classification? Supervised classification is I give you a bunch of labeled examples, right? Typically referred to as training examples. And you're supposed to come up with a function that'll label some new examples. And we showed this one where, we have a bunch of hand-drawn fours and hand-drawn nines, and the idea was, given some new input, is it a four or is it a nine? The idea was, since we know what the desired labels are from

the training data, we're going to try to build a classifier that's going to minimize the expected misclassification, or minimize our risk. All right, and we talked about that there were two strategies. One was called generative, where we probabilistically model each of the classes and then, somehow, picked them, we'll talk about those again just for a second. And then there is this discriminative, and discriminative says, I don't care so much about modeling the classes, what I care about is modelling some sort of boundary between the classes. And that makes it easier to make some decisions. So, we talked about the risk of a particular decision as being, if you're making a particular decision, what's the probability that that decision's wrong, and what's the cost of making that wrong decision? So, for example, let's suppose that I'm going to label something as a four. Well, if I'm going to label it as a four, what's the risk associated with it being a four? Well, that risk would be, well, you know, what's the probability that it's actually a nine, okay? And what's the cost, to me, of calling a nine a four, all right? So that's the risk of labeling something a four, right? because it actually might have been a nine, and what's the cost. Likewise, suppose I'm going to label something a nine. What's the risk of labeling something a nine? Well the risk of labeling something a nine is the probability that it was actually a four, times the cost of labeling something a four as a nine. Right? And, where should my decision boundary be? Well my decision boundary should be that the risk of either decision would be the same. Okay, and that was the idea of the generative model. And so that's what's being shown here is, so here are my, all my real fours so I'm pretty sure that they're four, so I'm going to call them fours. But as I'm getting over to the other side, it might be a nine. Here are all my nines, and as I'm getting closer over here, it might be a four. And this boundary in the middle, that's where the risk is the same.

2 - Generative Classification and Minimal Risk

So, to do this even risk assessment, we said we needed to know two things. One is, we need these loss functions. We need these costs, right? Somebody has to tell me, because I can't know that, it's not defined intrinsically in the data. It's the cost of, you know, either societal cost, economic cost, you know, whatever. As I said before, the cost of somebody calling somebody stupid, who's really smart and large might be much worse than the cost of calling somebody smart whose actually stupid and large. Think about that one, okay? The, but the basic idea is that the cost come from outside the data. But the other thing that I need is that I actually need to know the probability that something really is a four given some measurement. The probability that really is a nine given some measurement. All right? And we need to be able to evaluate those. The way we did that, if you recall, is that we, we talked about estimating the likelihood. That's the probability of the measurement given that it's, here we were doing skin colors. Given that it's a four or nine, or given that it's skin or not skin. And then, how did we go from likelihoods to actual probabilities posteriors? Well, remember we used Bayes rule. Okay, and Bayes rule would allow us to take these likelihood, right? The probability of getting the measurement given the class, skin, etc. Multiply that by this prior, just what's the probability of getting a four or nine? Or, in this case, of getting a pixel being skin. Or getting a pixel not being skin. And you multiply them, and you have to normalize them. And you get this posterior, which was the thing you wanted, okay? And that's what this equation down here says. The probability of skin, given x, proportional to P of x, given skin, the likelihood times the prior. And of course, then, the question came up of, where does that prior come from? And again, we had to learn it somehow from training data. So we showed an example. This is from Garrett Bradsky of classifying skin color. At the end of the day, you would basically take this value and you would compare it to some threshold. And you'd say okay, if it's higher than the threshold, I'm going to call it skin, otherwise it's not.

3 - Some Challenges for Generative Models

LIke you said, it's sort of an old school way doing it. They were the first methods being used in what's called pattern recognition. Pattern recognition's entire field or discipline that looked at making these kinds of decisions. And one of the reasons was it was very easy to model this analytically. Because you'd use things like Gaussians, or whatever. Or mixtures of Gaussians, and

you could do this with modest amounts of data in a smallish moderate number of dimensions, and because a lot of this was done before we had serious computers, or before we had serious data. This was really important. But sort of in the modern world, or not even the modern world anymore, but the world 20 years ago, which I like to think of as being modern, because back then I was in my 30s, and I'm hoping that I'm not like, really that old. But now I'm like, man I'm old, okay fine. But in the modern world there are some liabilities of the generative method. First of all, these days, many of our signals are high-dimensional, right. So we get some description of a signal, whether it's stock market data or whatever it is. I might have a feature space that's tens of thousands of dimensions, or at least many hundreds of dimensions. It's a very high-dimensional vector. And if you actually want to represent the density, in that high-dimensional space I, I wrote here that it's called sort of data hard. And what I mean by that is the thing becomes exponential in the number of features that you have. So you have to have sort of massive, massive, massive amounts of data. And even more mass, when, you know, we tend to think sort of Google styles data being is massive. No. You know, when you've got tens of thousands of dimensions, it, it's, the amount of data you need to densely sample that is, is exponential, and we just don't have it. But, there's even a bigger liability, okay? The bigger liability is that we don't actually care about modelling the coal class. Right? If I've got sort of skin things and not skin things. Imagine a whole bunch of colored pixels that have nothing to do with skin, I don't have to model that as not skin. We only care about making the right decisions. So we should be modelling the boundaries, the hard cases. The cases where I'm not really sure. And when you do a general model of a category or of a class, you're worried about the vast majority of them, so the skin pixels, not this, just this boundary that's next to the non-skin ones. And I'm not even totally sure what it means to model non-class, non-skin pixels, because that's everything from you know, the pixels on colored shirts, to the pixels on grass, to the pixels on tree bark. Yeah, it's kind of a weird, not, non skin's not really a great class. What I care about is, find me the pixels that may be labeled as skin, or maybe not. And finally, I'm going to give you a feature vector, a bunch of features that describe an instance. And we might not have any idea, a priori, which features actually are good for discriminating between the classes. So part of our building a classifier is essentially doing feature selection, right? We have to figure out what parts of the features, or what parts of the signal, are the sort of informative ones. And, so, for training a classifier, our general model has no way of talking about doing that. But the methods we're about to talk about, which are called discriminative methods, inherently do that.

4 - Discriminative Classification Assumptions

For the rest of this, and next, and next, we're going to be talking about discriminative methods for doing recognition or classification. By the way, I'll probably use those words interchangeably, classification, recognition. Probably a little sloppy of me because we talked a little bit about the difference. I'm just basically talking about classifying something as being an A or a B. Or even more precisely, an A or a not-A. Before we go forward, we're going to make some assumptions, okay? Our first assumption. We're going to assume that there's a fixed number of known classes, right? So I'm going to tell you, there are four things in this universe you have to worry about. Now, they might be As, Bs, Cs, and none of the above. But it's not like later I'm going to give you some Ds or later I'm going to give you some Es. No, I'm going to tell you the classes that exist a priori. So when I train this system and I'm defining the boundaries, I'm only worrying about the classes that you tell me in advance that I know about. The other thing I'm going to assume is that I've got plenty of data. I've got training examples that come from this class, and it might not be, shall we say, that data hard. You know, I've got an exponential amount, but I've got lots of data. So it's possible for me to typically find points that tend to be, you know, might get confused with some other class. For the conversations we're going to have today and the next time, we're going to assume an equal cost of mistakes. So it costs you just as much to call an A a not-A as it does to call a not-A an A. In fact, one of the methods we'll talk about, the fact that some of the distributions might be quite asymmetric, right? So you might be looking for, say, I don't know, faces. We're going to do that one. In an image, you may know that most places in an image, in most images, are not faces. So you might take advantage of that fact in terms of how you search for them. But we're

not going to worry about the relative cost. Remember that loss function between making the two kinds of mistakes. And the other thing is, that we're going to assume, is that, and this is actually good because it's true. It's always good to assume true things. We need to build a representation of each instance. So we're going to build a description. But we don't know a priori what features actually matter. So we're going to write down a whole bunch of things about each instance. And then we're going to train a discri, a discriminative classifier to make the distinction, the the classification, and it'll have to figure out where the information is in the feature. All right, so that's a lot of assuming, but I'm good at making assumptions.

5 - Basic Framework

There are essentially two parts to building a Discriminative System. We have to build an object model that is a representation of each of the training instances. And here we're going to be talking about images. The methods that I'm going to show you of course, the overall classification method could be applied to future vectors from all sorts of things. That in fact if you take a machine learning class, many of you already have. You learned about discriminative classification for arbitrary. vectors, so we'll spend a little bit of time talking about features that come from a computer vision or an image element. Right? So you would use them for biological data of describing genomes. But the representation you might apply to other kinds of pictures. So you have to build that model. And the second thing you have to do is you have to train the classifier. Then when you run this thing, what you have to do is you have to figure out, I need to generate new candidates to test. So I give you a picture. What does it mean to say, tell me, you know, what's in it. Is it the whole picture, parts of the picture, we'll talk about that in a minute. And then once I generate candidates I'm going to have to score them. I have to evaluate them with a classifier of yay or nay.

6 - Window Based Models

For computer vision object recognition, we're inherently interested in finding objects in images, okay? And that finding means essentially two parts. There's sometimes one. There's this notion of detection and labeling, right? So here in this picture of you know, person standing next to a car, I have to not only sort of tell you what's in it, I have to say oh okay, over here and I'll draw a box because a lot of these things are Windows based. Saying okay, that's the car and by the way over here, that's the person. So in general that's what you have to do. And the, sort of the most common approach to doing that is referred to as windowing methods. All right? So there are lots of different ways of thinking about Windows. We can start by thinking about sort of whole images, right. So, here I have a picture of a couple of training examples of koalas and a couple of training examples of pandas. This slide borrowed from Kristen Grauman, Kristen has a thing for cuddly folks with big ears. So if any of you are cuddly with big ears, you can give Kristen a call. I hope Kristen's okay with that. So you know, clearly here I have my koalas, we've talked about koalas before about how they're not bears, right, that koalas are these things. And here we have pandas, which are bears but are just mean. Anyway, and we'd like to do is we'd like to somehow learn a way of discriminating between them and say, you know, which ones are which. So, one way of doing this would be to just build some sort of a holistic description of the entire picture, with the assumption, we'll get to this part later, that the picture thing that I'm looking at is mostly the koala, or mostly the panda. So what would be the simplest possible description? Well here's one. How about I just build a histogram of the pixel colors? Now, I'm showing it here in a single image so it's, it's black and white. You can do this in R, G, and B, and have, you'd essentially have three histograms. And you could imagine that basically, you could try to determine that, you know, these things look kind of, pretty much different than this, so that might be a good descriptor to try to find where the boundary might be. You might try something even simpler. I don't know, if it's simp, yeah, I guess it's simpler. It's not a very good idea. All right? The same way we did in PCA, you can take all the little pixels, put them in a great big vector space and you could say, okay, here are a whole bunch of koalas, here are a bunch of pandas, find me the boundary. You know, do you think that would work well, or do you think that would not work well? What do you think Megan? Do you think it would work

well? No. No, okay, one of the problems is, you know, if I shift the picture just a little bit, all those pixels move around in that feature vector. So a small change in the image is going to give me a large change in my feature vector, which is not something you want to have if you're going to have a good ability to discriminate between them, so that won't work. And what about the color thing well, you think if I just did that color and intensity thing, that would work pretty well? No. No. Okay, look. Suppose I make it brighter today or there's a shadow being cast upon the thing etcetera, whatever. I get, you know, major variations in sort of the distribution of the intensities, okay. Do you remember when we were talking about trying to do tracking or trying to find things you know, motion. We were looking for what were called interest points. I hope you remember that, okay? And if you remember interest points, house features, those kind of things, they were based on the idea of places where you had high gradients, okay. In particular, those were where you had variations in the direction of the gradients as well, and the idea was that gradients corners were quite robust with respect to changes in illumination. So if we take our koalas and our pandas, we can consider sort of edges, contour. The idea of these intensity gradients, right? Those things tend to sort of stay where they are, in terms of, you know, if I change the intensity. All right. But of course we do have this problem that is, I move the image just a little bit, I might get shifts in the location of where those pixels are. So now, remember sift features, so we did that whole thing with sift features, which took histograms of the orientation, and we computed these little feature vectors. Right? Well here, imagine taking your picture, and you just cut it up into, say, four quadrants, right? It's easier, right? So, one, two, three, four. And I, suppose I just compute some histograms of the orientations of the gradients in each of those regions. And I do that for each of these pictures, okay? It's robust with respect to illumination changes because the gradient's there. Is it robust with respect to small shifts in the picture? Megan, yes or no? Yes. Yes, it is. So the reason that it's robust with respect to these small changes is that, you know, if I move this guy's ear a little bit around. You know, a bunch of pixels. The distribution of the gradients is, in that quadrant, is going to remain about the same. That's why down here it says locally orderless. What it means is that I'm not worried about exactly which pixel you're, you're in. I've got a general distribution in, in one or two different areas of the image. Okay?

7 - Discriminative Classifier Construction

We're going to talk a little more about this. But, basically, what I just talked about, for example, whether you're doing the whole histogram the histogram of the whole image, which doesn't work very well, but you could do it. Or if you build like these orientations of gradients in each of the quadrants, you're building a model, a description of that instance. By the way, you take each of those four, those histograms before and you just string them into one great big feature vector. That's your description. So now what you have to do is you have train, or sometimes referred to as learn, a classifier. Given a bunch of those feature vectors describing koalas and given a bunch of them describing pandas, figure out how to discriminate one from the other. So here's the basic description, all right? So let's suppose we're trying to learn cars versus not cars. So how many classes is that? Two. Sorry, how many classes are that? How many is that? It is six. There are six classes, but the number of classes is six. Actually, the number of classes is two. I don't know. It's called a binary classifier, because there are two classes. So, for example, suppose I have a little picture here of a car, and I have to decide, is it a car? And what do you think, you think that's a car? Yes. Good, she got it right. Yes. It's a car. Okay? Now here's another picture. Okay? Taken. Oh, somewhere in China I think? I don't remember where. Sorry. So, is that a car or not a car? No. No. Very good. What we need are methods of doing that. Now, there has been a huge effort in machine learning and applied to computer vision over the last really 15, 20 years, of methods of doing discriminative classification. What you see listed here are a whole bunch of methods, a couple of references below them of how to go about doing that. We're going to talk about three of them. I've just highlighted them here, nearest neighbor, SVMs, boosting. There are others as well. [COUGH] the general approach, and there's a huge literature on this now, especially with these large scale object recognition competitions, of what's the best feature vector you can use, learning different feature vectors, sparse representations, a bunch of other things. And then, given those

representations, what are the methods that you should use to make the classification? Here are the three that we're going to talk about. If you take more machine learning, just remember later you could apply those methods to those other representations. Our fundamental assumption is going to be someone and hopefully that's somebody you don't have to pay. But somebody's going to give you a database of lots of examples of the things you want to find and lots of examples of the things you don't want to find. And then you're going to train the classifier. Now once you do that, you have to come up with some way of test, given some new image. How do you test it? So remember we talked about you have to generate these new candidates and you have to score them. So here's the basic idea. You start off with a window, and then, and Kristen is really good at this animation, right? And you might take that window of a particular size, and you scan it all over. And what you do is you apply your car, non-car classifier to every one of those windows. And you say do I see any cars? Do I see any cars? Do I see any cars? Not very clever, but it just works very well. Right? Because the classifiers are quick and they're accurate. So, as opposed to having a different way of trying to do detection versus labeling, I, basically, do detection by labeling. There is a problem, of course, though. I picked a window. Well, suppose I'm nearer to a car. Does the picture get, of the car get bigger or smaller? That one's an easy one Megan. If I get car, bigger does, closer does it look bigger or smaller? Bigger. Bigger, very good. So what do I have to do? Well I just have to try bigger windows. And I slide those all over the place. And if this feels kind of date intensive, you're right. And there's various tricks to doing it. But that's the general idea is that you take these different sized windows and you apply them to different places in the image.

8 - Window Based Object Detection

Taking a little breather here just to remind you what we're doing. So we have this, for basically, we're doing window-based object detection, right? The training, we basically get it, see it says obtain training data. Doesn't say how. You just hopefully can go to the web, or like I said, you pay somebody to click on things, because you can do that these days. You define some features, so you have to figure out what it is you're going to measure about it. And then you're going to train a classifier, and that's what's shown here. So I take my windows that can, and I I, I have positives and negatives, that comes from this great big set of training examples you see over here on the left, I've got these cars, then over on the right, you've got these non-cars. So it is, you use those training examples, you extract the features, and then you train your car versus non-car classifier. At testing time, what you do is, you again, you're going to take your new image, and you're going to take this window and you're going to slide it all over. Until, hopefully, you end up, you land over here and you classify it as a car.

9 - Nearest Neighbor

What about those discriminative methods we were talking about? As we said before they find a division, a surface between the classes. We're going to talk about a couple of different methods. The one that I want to talk about today is nearest neighbors. And then, we'll do separate lessons for each of the, the other two. So the simplest method we're going to talk about is nearest neighbor classification. And it is a discriminative method because we are using the boundaries between I wont say classes, I'll say examples, right. The boundary is always between examples of the classes. But we actually don't have to do very much training. So the idea is we have some future space here labeled as blank. It should be x1 and x2. This picture by the way is taken from a classic book, due to Hart. It was, the original was due to Hart, the new version is due to Hart and Stork. And it's a good way of learning sort of classical pattern recognition. So the idea is I've got a bunch of examples. The negatives, so the not a's are these little black points, here. So the, the, the black thing. And then I've got some positive examples, right? And those are the little red ones, there. And when I come up with a new point, all right? So there it is in the green. Basically, that's my new novel test example. And how am I going to label that? I'm going to find the closest training example, so in this case the closest training example is that one, and so I say, ah-ha. The closest one is a positive, so I'm going to label this new one a positive. All right, now drawn on this picture

actually are the divisions themselves, and the computer scientists among you will recognize that as a Voronoi partitioning. Right, when you partition a space using a, a Voronoi method, you essentially carve the space up into these little chunks. Where this chunk means that if you're in that chunk, this black point is the closest one to you. And so all nearest neighbor is doing is giving you a Voronoi partitioning of the space. It's shall we say pretty easy to implement. It has a couple problems, one is it doesn't work very well. We'll get to that in a second. It's also, very data intensive in terms of the memory of what you need to know. Right, so if I give you a million trainee examples, how many of them do you have to remember, Megan? A million. A million, that's exactly right. Furthermore, every time a new point comes in I gotta find the closest one. So if I was really dumb, you know I don't have a compute, Masters of Computer Science Degree from Georgia, I might list go through all of them one at a time. If I'm a little smarter I'll use something called a KD tree or some other hierarchical representation but you're still searching through a lot of these things. So at test time, it's very painful as well so not only is it a lot of memory stored, etc. But let's get back to that doesn't work all that well thing. Well one of the things that might happen is that I might have, you know, an occasional kind of spurious point. Or I might be in an area where I really don't have too many points nearby. And what I'd like to do is be able to make a more robust decision. Okay, and the way of doing that is referred to as k-NN or K-Nearest Neighbor. And it's really very simple. It's basically the idea, if I've got some new point, and it's written here as an x, so that point right there, I don't just find the nearest point. You gotta think like a computer scientist. You find k. So k might be one, might be three, might be five, might be seven. Whatever choice you choose to make, you would look for k. So, in k-NN, in 5-NN, for example, you know, if I have this point x, okay? What I'm going to do is, I'm going to look for the five nearest neighbors, right? And I'll just, one, two, three, four, five. Okay, and in this particular case three of them are black, two of them are red, black is negative, red is positive so I would classify it as what? Negative. Okay? One of the funny things, well, one of the interesting things about k-NN is it works really well, okay? It does have this data intensive problem, and there are methods that we now use that are kind of, they're sort of related to k-NN. But this idea of getting a loose consensus is very effective. We're not actually going to talk about a thing called random forest later, but this notion of consensus. So I don't get the support from just one place or even one classification method little classifier, this notion of consensus is very powerful.

10 - End

That ends the introduction to discriminative methods. So we talked about that basically the idea is that we're going to build a representation and then somehow we have to train a classifier that's going to look at the division of the boundary. The simplest method we showed was nearest neighbor or KNN. Oh, I didn't say, would k of set of, of say, mm, six be a good idea? No. No why not? because how does it work? You'd find nearest k and you pick whichever one has the most. Why are there nine Supreme Court justices? Because you don't want a decision to come out four-four, if there were eight. So k typically is odd, so that you get, you always a majority one way or the other. All right. In the next two lessons we'll talk about some much more sophisticated methods, and in particular, their application to Computer Vision.

8C-L2 Boosting and face detection

1 - Intro

All right, back to computer vision, some really cool stuff today. We're going to be talking about boosting and one of the, applications of machine learning classification to computer vision, that really caused a bunch of us and the rest of the computer vision community, to say, oh, we really gotta pay attention to how all this, notion of integrating, discriminative classifiers, the notion of detection, and image features and representation as a, as a holistic thing. So last time we introduced discriminative classification and the basic outline was there are two elements. There's the train, and

there's test. For training, you basically have to figure out what representation am I going to build, how do I construct a model or, or a description of each instance? So I describe these training instances, and then I have to build a classifier, train a classifier. And we said once we've done that when new things come in, might take some work to figure out how to generate candidates in the new image. Or you might just slide windows around to different sizes and then you have to score them, all right. We talked about a relatively simple method last time called nearest neighbor, as it just very date intensive. You pull up the nearest one, which was implicitly representing the boundary between every single point, remember that Voronoi tessellation. Today we're going to talk about a different method called boosting and in particular, we're going to talk about it in the context of a particular method that, that used it that sort of shows you the power.

2 - Boosting

Boosting is an iterative learning method, right? So you, you, you massage the data and you keep co, constructing a better and better classifier. And it focuses this idea on every iteration of trying to look at what's called the weighted training error. So initially, you have a whole bunch of samples, and you weight them equally, right, so they all have a weight of one. And then in each boosting round what you have to do is you have to find a weak learner, we'll talk about that in a minute, that achieves the lowest, or not even the lowest. You have to find the weak learner, and then for that you want to find we'll just call it you want to find as without working too hard, you want to find, pick the weak learner. It gives you the lowest weighted training error. And then, what you're going to do is whichever ones you made a mistake on, you're going to raise their weights. And then whichever ones you make a mistake on using that current weak learner, you're going to raise their weights. You could also alternatively lower the weights of the other ones. Typically, raise the weights of the ones you've made a mistake on. And then what you can do is you can take these weak lay, learners and you're going to combine them in a very simple linear way to build your final classifier. So the question now becomes what is a weak learner, all right? Well it's simply a function that partitions your space. It doesn't always give the right answer, but it gives you some information. And the idea is you're going to combine these. So this is actually easier to illustrate than to talk about, regardless of how much I like to talk. All right, so let's develop our intuition. All right, let's suppose we have children, okay? So as everybody knows, especially in the American world somewhere around the first of December, there are bad children, and there are good children. And somebody makes a list and does something about it, I don't know, whatever it is. And suppose we want to a learn, a learn, we want to figure out a classifier that can classify children. And maybe we've got two dimensions that describe children. You know, whether they put their socks away and whether or not they buy their parents presents. And what we're trying to do is we're trying to use these features to discriminate between them. So the first thing we do is we try to pick a weak learner, and we say well, how about this line right there, okay? You know, what we'll say that these are the bad children, and these are the good children. Okay, now of course, that's not perfect, is it, right? We have some errors here. Okay, so the, the, the red is bad and the blue is good by the way. That's not meant to have any military implications whatsoever. So what we've drawn here is this is our weak learner, right? Our first level weak learner, and you know, it does an okay job, all right? So remember we said in each boosting round we want to find the weak learner that achieves lowest weighted training error. And then we're going to raise the weights. So what do we mean by weighted training error? Well here was the line that we drew. This is a better version of that line, but it has some errors, right? And in particular, what we want to do is increase the weights of the error. So if we had said that the ones above the line were bad or red, and the ones below the line were blue or good. The, the blues that are actually above the line, and the reds that are below the line, those are an error. So, what do we do? Well, now, we just do it again. We have to find a new weak learner. Well, how 'bout this one? Okay, that's pretty good, actually. Here's, it's called a weak classifier, weak learner, same thing. You can see that it does a pretty good job. In fact there are no errors set over here if this is the good side, all right? And on the side that it calls the bad side, there are only these two errors over there. But in particular, this nice big weighted one from before has been classified correctly. So that's our second weak learner or weak classifier. Increase the weights

of the mistakes there. Say oh, okay now I need another one. Well, how about this weak classifier. Well, how many errors does that make? Well, notice this particular one, it's only got this bad one and it's a small weighted error. And the idea in boosting is that you can take all of those learners, those weak learners or weak classifiers. And we're going to combine them in a way in order to make a decision, all right? So, in general, in boosting, you're going to come up with a final classifier that is a linear combination of the weak learners. And typically the weight is in some sense proportional to how accurate it is, a fixing the weighted error from before, reducing that, all right? There are a variety methods of doing that, and we're not going to go through any of the exact formulations. There's one of the schemes is called AdaBoost. And it's the one in fact that was used in the algorithm I'm about to show you. But there are lots of things and you can go look up different ways of, of running boosting.

3 - Viola Jones Face Detector Introduction

So now, let me introduce you to the computer vision paper that sort of motivated us all to pay attention. And, that was this paper by Paul Viola and Mike Jones. It was called Rapid Object Detection using a Boosted Cascade of Simple Features, that objects were faces. And, that's part of what caught everybody's attention, because everybody's been looking for faces for a long time. This was in CVPR 2001. I will tell you that I was at a presentation that Paul gave. I think it's actually even before this was published, where he pulls out a camera, connected to his laptop, sweep, sweeps it around the, the, audience, and it finds all the faces. And, this is on a generic laptop, wasn't very complicated. And, it was it wasn't doing training, he'd have trained it before. Training is expensive, but we all said oh, and from then on we paid attention to Paul. Not that we didn't pay attention to him before, but this was, this was a really big deal. All right, so there were several good ideas, in fact, better than good ideas in this paper. First one, which didn't seem like such a great idea until we saw how easy it was to do it was to use rectangular brightness patterns as the features that you're going to compute. And, let me show you what, what we're talking about here. All right, so here's some rectangular filters. And, a rectangular filter, for example, this one here, right, just means that you take, see that's pli, its white here and black there. So, you just sum up all the pixels in that area, and you subtract off all the sum of the pixels of that area. So, you can think of it as a large size kind of a gradient, right? So, there are ones that would compare horizontal, you know, the pluses over here, and the minuses over there. There were ones that were vertical, minus here, plus there. They were of different shapes. They also had some three way ones where you'd have plus, minus, and plus like that. And then there were these cross ones where you have plus like that, and minus like that. Just a whole bunch of them, and then those would be applied to the image. And here, we're showing you two different features, again, comes from the paper, right here. And, the idea is that you might put it down here in the picture, and put it down here on the picture. It just looks at the difference in average intensity of adjacent regions, or three of them, two, you know, three regions in the face. Now, the reason this turned out to be such a great idea, is that you can compute this thing very, these things very, very quickly if you're going to compute a lot of them. And, the idea was to use something called the integral image. And, this image here, this is the integral image, and I'll explain what it is. The integral image works the following. The value at some integral image, x, y, would be the sum of all the pixels in the top left-hand corner behind it in the image. Okay? So,you go over to the, so if you have my original image here, the integral image is the sum of all of these pixels, and you just put its value right there. Okay? That's the integral image. Now, you might ask, why is that so important? Why is that so important? Very good. So, I'll show you.

4 - Computing Sum Within a Rectangle

Let's suppose there's a rectangle here, as A, B, C, D. And you want to know the value of the sum of the pixels in that rectangle, okay? So what you do is, and that you want to know the value in the original image. So what you do is you take a look at that rectangle, and you go over to your integral image, all right? And you realize the sum of that rectangle compu, in the original image, is

computed by looking at just these four locations in the interval image. And it's written down that the sum is equal this will go through that, right? You start with A. Well, what's A? A is the sum of everything up to here in the original image. Okay, but I don't want everything there. Right? Like, I don't want this stuff, and I don't want that stuff. So you say, okay! No problem! I'll subtract off B! B is just this stuff, right? B, the value at the location B, in the integral image is the sum of all of these pixels in the original image. So if I just take A minus B, it was this whole thing, minus that part. So it's just that region. Okay well, I still have got a section in there I don't want. Namely I don't want this. Well that's easy. That's C. Okay, so I subtract off C. And I'm almost done. The observant of you will notice that I've now subtracted this region off twice. So what do I have to do? I have to put it back in once. Well, do I have some value that's the sum of that little square? Of course. That's D. And that's why adding D back in gives me the sum of that, of the pixels in that rectangle in the original image. So only three additions, and/or subtractions, right? I have to add, subtract C, subtract B, and then add D to A. So I access the image four times, do three addition, subtractions, and I get the sum of any size rectangle! So once I compute the integral image, and that's actually very quick to do, because the integral image, that means I'm going across the top row, I just look at the current pixel, and I know the sum of everything behind me because I've already computed that, just add that in. So it's recursively very quick to compute the integral image. So I can compute any size rectangle quickly. What this does mean by the way is, if I'm looking to, for big faces and small faces, I don't want to change the size of the image that I'm looking at, instead just change the size of the features that I'm looking at because it's really easy to look at a small rectangle or a big rectangle once I've computed an integral image over, for that original image.

5 - Compute Integral Image

How about you write the function to do this? That is compute an integral image. At every location (y, x) and the resulting image I, you should store the, the sum of the pixels from (1, 1) through y, x. That is from the top left corner 'til that pixel including y x coordinate. Here's some test code. We're using a magnified image of blood cells, for no apparent reason, and this is how we should be able to call your function. You should then be able to display it scaled, like here. Or just by passing an empty vector in as the second argument. Note that the maximum value in an integral image is in the bottom right pixel. So this max expression can be replaced by I end, end. To check whether the function is working correctly, let's define a window. As a ground truth, we can select this window within the original image, and compute its sum. Note that when reading the image, we converted it to double type. So we shouldn't face any overflow errors. The equivalent expression, using the integral image I, looks something like this. I'll explain this later, but it's basically A minus B minus C plus D. These two values should be equal. All right? Give it a shot.

6 - Compute Integral Image

The easiest way to do this is using cumulative sums. You can call cumsum with a matrix and a dimension specified. Here 1 tells it to sum up rows. So the first row gets added to the second. And then that result gets added to the third and so on. Let's see what this looks like. I'll comment out the rest of the code, since it's of no use right now. You can see the effect of rules being summed. Getting brighter and brighter as you go down. Here is the original image for reference. And here is the row-wise cumulative sum result again. All right, so this takes care of summation in one dimension. As you might have guessed, we can take the cumulative sum along the other dimension of this intermediate result in order to get our integral image. Let's clean up and do that. So, as we said, the first cumulative sum is an intermediate result. Now we summed this up along the other dimension. That is, along the columns. The first column gets added to the second and that result gets added to the third, and so on. And that's it. This is our integral image. Lets see what the result looks like. You can clearly see a gradual monotonic increase in brightness from top left to bottom right. That is to be expected, isn't it? Lets check if the two computed values were the same. And yes, they are indeed. Okay. So what about this crazy formula? Within the larger image, we've defined a window using x1, y1, x2, y2. The top left corner has coordinates x1, y1. And the bottom

right has x2, y2. The first value, I(y2, x2), is the integral image value here in the bottom right corner, which is basically the sum of the original image from the top left till that point. We call this A. Remember B was this rectangle up here? Note that it needs to go to the column value x2 on the right. But needs to stop one short of y1, otherwise the values at row y1 will get subtracted from our final result. This is why you have a minus 1 here. Again this was B. Similarly for c, it needs to go down to y2. But needs to stop one short of x1. Hence, x1 minus 1. Note that the region d that has been subtracted twice is only up to x1 minus 1, y1 minus 1. And that's exactly what we add back in. Clear? Now think of an edge case where x1,y1 equals 1,1. That is the top left corner of our desired region is at the top left corner of the image itself. These minus 1s can then result in invalid indices. To overcome this problem, Matlab for instance, pads 0s on the left and top, that is a single row of 0s. It increases the resulting width and height by 1, and it requires you to change the final computation expression. But this is just an implementation detail. Overall, I hope you can appreciate how vastly significant integral images are. Once you've computed an integral image, you can replace each of these summation operations, which involves probably hundreds or thousands of additions, with just two subtractions and one addition right here.

7 - Weak Learners

So now that I can computer rectangles very quickly, I might want to look at a lot of rectangles. And, in fact, for training their boosted classifier, these guys looked at over 180,000 possible rectangles that could be applied at, sort of, a 24 by 24 window. Okay? And here's basically what they're going to do. They're going to look for face in 24 by 24 windows only. And they're going to take your new image, and scale it to a couple different sizes, and say, okay can I find it in anywhere in that 24 by 24 window? Okay, so we said the first main idea was to use these patterns. Well, remember we just talked about boosting where use, you look for weak learners? All right? Well, basically they're going to choose whichever features they can find that tend to be the best weak learners. So they're going to use, in fact they use a thing called AdaBoost in order to do that. So you want to see which weak learners reduced the weighted training error the best? Don't you? Yeah. Don't you, don't you? Okay, well turned out these are the first two features selected. All right? That is for a 24 by 24 window, if you have a face in here somewhere. Those are the features that they're going to use. And you might wonder why has it become those features? Well take a look at it this way, right? It's basically looking for eyes, sort of darkish patterns in there compared to what's down here. Okay? And then, there's this patch here that says the middle of the, it tends to be brighter in the middle than around the edges. Just these two features, these were your first two weak learners.

8 - Cascade

All right, so we've said before, so far our main ideas are, we're going to use those rectangles as features. And it's really easy to compute them because or the integral image, and then we're going to use that discriminative ones as the weak learner, the weak classifiers. All right, well we're using this with a within a boosted framework so not surprise you that we're going to combine them to make our final classifier, that's what boosting does. But the other idea they had, was to form this thing called a cascade of classifiers. And it's such an important idea that I made it in italics, yellow, and red, okay? We're going to form a cascade of these classifiers, and we're going to reject clear negatives quickly. What's this all about, all right? So the second big idea of this paper, besides the integral image one, was the cascade. And here's the issue. Even if our filters are really fast to compute, so even if we can get these you know, compute these rectangles on a new thing very quickly there's an awful lot of places to go search. How can we make the detection more efficient? And this was the insight. Almost everywhere in a picture is not a face. I don't care what the picture is. You know, unless you happen to be looking at the, you know, the 1972 New York Yankees like this, you know, we got, you know, 75 faces. But even then, most of the locations are not a face, there's only 75 locations that are a face, and all the other locations, remember Kristen's little, those are not faces. So, what you'd like to be able to do, is quickly know that something is not a face.

Right? So you can move on. And, if I say that something's not a face, I want to be sure that it's not a face, so I can move on. So what they built was a cascaded set of filters. Okay, so at each level, there's going to be a boosted filter, but here's what they did, all right? So, you have a whole bunch of, of images, windows, scales, et cetera, whatever. You train a classifier, it's called stage 1. All right? And Whatever it says is not a face, you're sure that it's not a face. So when you adjust your classifier, you adjust it so that it has no, what are called, false negatives. Everything it says is not a face is really not a face. Everything that you say is a face, you're going to put into another classifier. So you're going to train a second stage classifier, and what it's going to do is, remember this is on training data so it's labelled, you're going to give it a whole bunch of positive faces. But what negatives are you going to give it? The negatives you're going to give it are the false positives from the last stage. So stage 1, remember, we set it so that everything it says is not a face isn't, but it says a whole bunch of things are a face. Some of those are wrong. Those are false positives. So use those negatives, plus real positives, to train the next classifier. And we do that again, and what that gives you is a cascaded set of filters where when you apply this in testing data, you have to make it through all the filters, all the stages, in order to be labeled as a face. But to be rejected, you can be rejected quickly. Its just like dating in college, right? You can get rejected quickly, but it takes you a long time to get accepted. All right. Just as a quick summary, we have a whole bunch of images, faces and non-faces, we're going to train a cascade of classifiers, they used AdaBoost, all right, which is a particular boosting algorithm. And we're going to select out features and thresholds and weights, which is do, you have to do in AdaBoost. And then, what you're going to do, is for some image, you're going to apply it to the window. That's where the face really is. We clicked the window, we slide it around, and there it is. And it finds the right face.

9 - Viola Jones Results

I have to tell you this worked extremely well. These are some examples taken both from their original paper and some others, so they were showing it running on all of these different images. Some of which are shall we say, you know, typical computer vision images where the face is right there, some of them are much harder, right? Now you'll notice, and we'll get to this in a minute, that this guy is missed. You know why he's missed. It's not, because he's not good looking. I have no idea whether he's good looking, that's up to you guys. It's because he's turned this way, and the thing was trained on frontal faces. Here's some more examples. You know, I love the Star Trek one that it worked really well, but for some reason Scotty there, James Doohan, I don't know if it's the mustache, whatever it is. This thing likes Vulcan's more than it likes Scottish people, can't explain why? But basically the idea is the thing was able to work, extremely well. It also dates it a little bit, because here's a picture of the Seinfeld Cast, so that tells you something about when this work was done. Some interesting errors. Notice this one down here, okay? So we got all the faces on the soccer player, football for those of you not in the states. All right? But, take a look at the ball at the bottom there, right? Now if you kind of squint at that a little bit, oh, I'll just take off my glasses and look at that and say, oh yeah, it's George and they say, no, that's a soccer ball. The distribution of the pattern was kind of close and it labeled it. Here's some more examples. And again, you'll see that this profile was not done so well, so what do you do? Well, how about training some profiles? Again, train the classifier using the same types of rectangular features and now all of a sudden you can find faces site, you know, these faces here and those faces there. Couple of cute things that were done by folks. They found faces in video. That was done easily, but the video came from broadcast video that had annotated, subtitles, including characters, and they started to be able to associate different faces with different characters. So eventually, as you can see, it would recognize Buffy the Vampire Slayer, because after all who would be complete without recognizing Buffy the Vampire Slayer, but the work was very cool. So here's an article pulled out 2007, so most of you used Google's Street View. Well, when Google's Street View first came out, there were people's faces in it. Actually it's not totally true because they knew this was going to be a problem. But as you could imagine, people didn't want their faces, and by the way, not just one. In the United States you can push back a little bit in terms of faces. In Europe, you have a much greater right to your own face, or I should say to the image of your own face. So it was very important that Google be

able to remove, the faces of people from their street view. So to remove them, you have to detect them, and they're not going to pay 9 million undergraduates to sit there and click on all the faces, so that you run the recognizer. Here's a thing pulled out from iPhoto. Most of you have an image in our introduction you know, on the back of your camera where it finds all the faces. Many of those systems use either the Beulah Jones method or some version of it. I do like the, Lana had this picture of things that iPhoto thought was face. Here's a picture of a sort of a teddy bear made in chocolate chip cookies, and iPhoto thought it was a face, because the pattern is about right.

10 - Summary

To briefly summarize the Viola-Jones face detector, couple of key ideas. First of all, they're going to use these rectangular features and really the big idea was the integral image that makes it easy to compute those rapidly. Second, they used AdaBoost for doing both feature selection and for training the classifier in general. We didn't talk a lot about that here. Sometimes I talk more about it, but, these days you just go look up AdaBoost, it's easy enough to implement. The really useful idea was the cascade filter, an, and other people had done cascades and various kinds of things, but, the way it was applied to face detection was here. Training is very s, not very s, training is slow. Remember those 180,000 different possible features, and you try them and you're looking for all these weak learners. So there's a lot of potential weak learners to choose from. But, detection run time is very fast. So when I pick up my little camera or I point my I my smartphone, it can find the faces very quickly. And it is really, really effective. It is rare to see a paper turn into a both commercial technology and impact the field so dramatically, so quickly. And, you know, that's some measure of the effectiveness of the paper. And neither Paul nor Mike gave me any funds to endorse them. All right. So let's summarize boosting real quickly. It combines feature selection with classification. Remember that selecting of the weak learners? So you can think of that as sort of selecting, not to, you can think of that as, as finding features. And pretty much any scheme you come up with for defining weak learners and trying them will work. The complexity is linear in the number of training examples. You're basically trying different weak learners. You pick however many you want. And you apply them against your various training examples. We mentioned that here because the method that we're going to talk about, some of the bets we talk about later become sort of somewhat quadratic because you have to compare each of the points to each of the other points. You don't have to do that here. Testing is very, very fast, it's ve, very easy to run. You can write boosting code, in fact, most, many people have. You might ask, why wouldn't you write your own code? Why wouldn't you write your own code? because the me, some of the methods that we're going to talk about are actually, the bookkeeping is very difficult, and people use other libraries. And then when the stuff doesn't work, it's much harder to track down, was it the classification method, was it the parameters within, etc.? Because it's kind of a black box to you, boosting you can write your own. There are some disadvantages. It does need lots and lots of training examples. But remember, that was an assumption that we made, not this lecture but the one before, that when we're doing discriminative training, we're assuming we've got lots of examples. Now it's sometimes found to not work as well as some other methods. For example, support vector machines which is something we're going to talk about in a little bit. And lately, there's been a lot of work on random forests. Which I won't do in this class, but is sort of the, I a even newer approach to, to doing that. And it's been particularly taken it a bit on the chin for many-class problems, so not a binary classifier, etc. But it is still a, a pretty effective way of doing these things.

11 - End

All right, that concludes the, the boosting discussion and also the context of face detection. Like I said, it's rare that a single paper can influence a field so much, like this one did. But but it did. And not only influenced the field, it influenced the people sort of learning and thinking about the field. The representation, the classification method and then the whole cascade, it just worked. It just worked really, really well. And, as I said, unlike SVM methods, which we'll learn about lex, next time, you can actually just write the code yourself, both boosting and random forest. By the way,

random forests are sort of related because it's this idea of adding these weak eh, this consensus of weak learners. So, for those of you who want to think about random forests, because you know about them, you can think about them in that context. These boosting algorithms, these consensus algorithms, are just much simpler to implement yourself. And therefore they sort of become fun to play with. Next time we'll talk about support, support vector machines, which are not fun to implement, but just work really, really well.

8C-L3 Support Vector Machines

1 - Intro

All right. Welcome back to Computer Vision. If you remember last time, we started talking about discriminative classifiers. So we first did generative, then we did discriminative. And the idea of discriminative classification or recognition is that instead of worrying about modeling our classes, we're going to learn a boundary. We're going to learn a surface, and some feature space that divides our classes. And we've talked about two simple method, well actually, one simple method, nearest neighbor, a much more interesting method of boosting. Last time, we were talking about the Viola-Jones face detector which used boosting. But today we're going to talk about support vector machines. Support vector machines became a very significant method of doing classification, and especially is applied to computer vision. It really and today, and still today is is, is fundamental

2 - Linear Classifiers

So, in order to learn about supported vector machines, we have to talk about what are called linear classifiers. And linear classifiers are exactly what you might think. So here I have some data points. I got blue ones and, and pink ones. And the idea would be, what if we just wanted to find a linear separator that separated things. Well, in this particular case, we could do that, no problem. So it would be something like that. Ta-da, there's our, our linear classifier. And you know, what we'd like to do is to talk about finding these kinds of things automatically, and then the whole point of support vector machines, is to do these in well, what you'll see are some higher dimensional spaces which will give us some very effective classifications, some powerful classification machinery. So, let's talk a little bit about lines. So here's just lines in, in regular 2D space. So here's an equation of our line, and this is the normalized equation, and we used to write ax plus by plus c equals 0 when we did a bunch of things. And I'm not doing that here because in SVM land, where it's derived, b has a particular value, it talks about, it, it, it's written in a particular way. So I wrote it as px + qy + b equals 0. And really, what I mean to say is that we're going to let some w, here it's, be pq, and pretty soon those will go away and we'll only talk about w. Let w be pq, and if we have some point, x, which will be sort of little x, little y. Then the line, px plus qy plus b, that's the same thing as this line w dot x plus b equals 0. All right? And w here, this vector here, that's going to be in this direction there. Okay? One thing that should be clear from this, right? I could scale w and b and it'll still be 0, right? Multiply both of them by 3 and I get the same value. All right. Now, if I have some point, x0, y0, I might want to talk about the distance cap D of that point to the line. Okay? So here we have our, our point and then there's this question of what that distance D is, all right? Well, given the equation of the line, we can express that distance in just the, the following way. It's just essentially, find the dot product of that on w, subtract off or, or think of that as being connected to b. Whatever that value is, that's how far off the line it is, and you then have to normalize because from a distance, by the magnitude of p and q, so that's why it's divided by the square root of p squared plus q squared. And that can be just written like this. Okay, so it's just w transposed, x plus b, the line equation up there, divided by the magnitude of w. Okay, so that's the distance D.

3 - SVMs

If we want to build a linear classifier, so we want to find some function to separate the positive and negative values, and the idea is that we want a line x dot w plus b such that, if the xi is positive, so if that value is greater than zero we're going to call it a positive, so that's here. So if it's greater than zero we'll call it positive, and if it's less than zero, we'll call it negative. But let's take a look at these data. So here's a line that separates the blues from the pinks. That's a good line. Good line? Sure, it's a great line. But here's another line, all right, and here's another line, and here's another line, and all of these lines separate the blues from the pinks. So the question is, of course, which line is best? All right? And support vector machines are a particular take on how to decide which line is best. And then we'll go to hyper planes and higher dimensions there. But it's basically having to do with which classifying separator is best. So, Support Vector Machines, SVMs, are really this idea of the optimal or the best possible separating line, or in higher dimensions, plane. And the way we do that is, so you see this line here? This line is drawn down the middle there. Well, sort of down the middle, okay. Well, great, now it's a ribbon. Okay, fine. That line down the middle. And the dotted lines here, right, this is sort of the closest distance of the, the points. So those two lines are drawn such that this line is the closest blue, here's the closest red, and this separator line is down the middle. The idea, the Support Vector Machines, is that we want to maximize this distance between those dotted lines. And that distance, that's called the margin. Okay, it's written in green here. That's the margin between the positive and negative training examples. Right? So, here, so it's from here to there, okay, and the idea is to try to find the line that gives us the best margin.

4 - How to Maximize the Margin

All right. So to do that, is just a little bit of math, it's not a horrible amount of math, it's just sort of pretty straight forward. So here's our line, and let's set our line to be wx plus b equals 0, and that's going to go down the middle of the line. I'm sorry, down the middle of the set, that's going to be our separating boundary. Now, what we can do is we can say the, the lines that are the, the margin lines, the ones that hit the nearest points, we can say that the value on one side is w.x plus b equals minus 1, and on the other side w.x plus b equals 1. Now you might ask, how come I can set them equal to 1? How can I set them equal to 1? Megan's asleep over there, but she got to it eventually. Well, remember I could scale, w and b, right? And still have wx plus b equals 0. So what I can do is I can scale them up to make it so tho, those distances are 1. What we want is, we're going to define a little auxiliary variable called y, corresponding to the xs, and the idea is that when I have a positive x, we're going to find y to be plus 1. When we have a negative x, and what I mean positive, is a positive example and a negative example, you know, this is a baseball, this is not a baseball. All right, so when y equals minus 1, we've got x.w plus b to be less than minus 1. So that's a way of saying that, you know, we want all the positives to lie over here, and we want all the negatives to lie over there, okay? Notice, that on this drawing, there are a couple of special points, okay? Now, in today's age, we like to tell points, all points are special. But in reality, just like real life, only some of us points are special, okay? In particular, this point is special, this point is special, and this point is special, because they define our margin. All these other points out here, they don't impact our margin, all right? And this is part of why discriminative classification is, is sort of better, not sort of, is in some sense better than generative. We don't actually have to worry about representing all these points out there. We're just representing the boundary but getting back to what we're saying, that we have these three special points that define what our margin is. And those, are referred to as our support vectors, these are vectors, right? They're points in our feature of space, so you can think of that as a vector, and these are called support vectors because these are the ones that sort of define for us that, that best separating plane. So those are support vectors, okay? So at our support vectors, x.w plus b actually equals 1, right, okay, and the distance between those lines, that's going to be referred to as our margin. So we can compute, the size of M, as a function of these values, right? So we have the same equation we had before, the distance between the point and the line, x.w plus b normalized by w, okay? That's the, that's the, the, the distance between a point and a line. Well, so for support vectors, w.x plus b is just either plus 1 or minus 1, okay? It's

plus 1 on one side, minus on the other side, divided by the magnitude of w. So that means that M is just, you know, 1 minus negative 1 over w, because of that value, right, and so it's just 2 over w, okay? So the margin M is just 2 over w. What we need to do is we want to find w that does what? We want to maximize the margin

5 - Finding the Maximum Margin Line

So we write that like this, all right? So just as we said before, we're going to maximize two divided by the magnitude of W, subject to the constraint, we want to correctly classify all the points, right? So for the positive X's where Y is plus one, we want X dot W plus B to be greater than or equal to one. And if, XI is negative, we want W dot, X dot W plus B to be less than negative one. All right? So this is a standard, what's called a quadratic optimization problem, we want to sut, we want to minimize this term that's over here, okay? Because if you minimize the square of W, then that's going to maximize two over the norm of W. Subject to a constraint, right? Here's our constraints. You'll notice that what we did is by multiplying by the YI we can say that this thing is always greater than or equal to one, right? because YI flips the sign, because YI is positive when it's a positive example, and YI is negative when it's a negative example, so multiplying it through keeps that constraint always greater than or equal to one. The solution to this, and I'm not going to go through the solutions, but it's sort of a standard quadratic optimization and some of the multipliers, things like that, but what matters is, is the form of the solution, not how we got there. It's just the linear sum for W of some weights times, and it's the YI times XI. So, remember Yi sort of flips, flips the sign. All right? And the XIs that you use are the support vectors and the alpha Is are these learned weights. And another way of thinking about it is that the weights, you can think of that of alpha Is for all of the XIs, but they're only non-zero for the support vectors. Okay? So when I, when I solve this, I end up with a whole bunch of alpha Is, most of which are zero. But the alpha Is for the support vectors are not zero. Okay? Well then I can just sort of substitute in, we have this equation that for any support vector W transpose X plus B is equal to positive one or negative one, depending upon the YI, depending upon whether it's positive or negative in this example. So that can be written like this. Remember the YIs are plus one or minus one, depending upon the example. So I can write, just substitute in for W. Okay, so I substitute in for both B and for W. Given this, we can now create a classification function. So remember, what we were doing was we were drawing a line W X plus B equals zero. Everything on one side of the line we were going to call positive everything on the other side we were going to call negative, right? Our support vectors are out here where that function equals one, but, but if something's on the inside, if it's greater than zero, we'll call it positive, less than zero, we'll call it negative. So I'm going to build a classification function, which is right what it says here. It says, okay, I'm just going to take the sign of W dot X plus B. Greater than zero, I'll call it positive, less than zero, I'll call it negative I substitute in the, for W this expression here, okay? And take that sign, and if this function, F of X is less than zero, I'll call it negative, and if it's greater than zero, I'll call it positive. Okay? So this all depends on you being able to solve for those alphas, and I'll tell you the support vector. Machinery will do that. And realize by solving for the alphas, by finding some non zero alphas that's selecting your support vectors. Okay? A critical, critical, critical element of this equation is that it only depends on this dot product. Okay? It highlighted here in red, I'll highlight it in red again. The only thing I need to know is the dot product between some new point X and the support vectors XI, and then I multiply it by the alpha, sum them up, and build my classifier. That dot product element, or that dot product structure, is what makes what we're about to do work.

6 - Questions

Having done this, what we, let's see, what have we done so far? Well what we've done is, we've figured out the best possible line in sort of two dimensions. Okay? Well great except what if, a, suppose my features are not in two dimensions? Okay. And also, I've been sort of assuming that I could actually draw a line. Well, what if my categories are not linearly separable? All right, and hm, what if I have more than just two categories? Well, what we're going to do is, we're going to knock

down those questions one at a time, going forward for SBMs. So we're going to start with, what if the features are not two-dimensional? Okay? Well actually, nothing about what I showed you in the math required that it be 2D, right? W.X. I didn't tell you that X has to be two dimensional. No, it can be any dimensional. Or D dimensional, N dimensional, whatever you want. It just basically, you're now instead of using a line you're creating a hyperplane. The math is exactly the same because remember a plane is set, remember how a line was defined, in fact so, in 2D a line was defined by the normal. Well in 3D, oh this is going to be priceless, a plane is defined by the normal, right. So as long as they have this normal W, I'm getting a plane. In 3D, it's a real plane. In ND, it's an M minus one hyper plane. But other than that all the math is the same. And in fact, using just that machinery you can actually do some pretty cool stuff in computer vision.

7 - Person Detection

Probably the best known example and this is one of the, the even though SVMs had started making their way into computer vision beforehand. I, I think the one that really sort of helped put it on the map was this paper by, this work by Dalal and Triggs in CVPR 2005 for just doing person detection and basically they were taking upright person detection. The thing that you need to know is basically they cut, I'm not going to do any more than that, they cut their pictures up into cells and they built little histograms of the oriented gradiance in that cell. Okay? So the features were called HoG's. Histograms of Oriented Gradients, that's what this HoG's stands for. And then you take one here and here and here, and you just stack them up into one great, big, long feature vector. Okay, you just have a great, big, long, feature vector. They took a whole bunch of windows that somebody had put, drawn a little box around and said, that's a pedestrian. Then they took a whole bunch of other little windows that said, not a pedestrian. They played with rescaling things etcetera, whatever, training it and basically trained an SVM. And how well did it work? Well obviously relatively well or I wouldn't be talking to you about this. Here you see, taken from this movie, the ability to detect people, here's some more of their work. You know, here's a snowstorm. I suppose it's in England, but I don't know. You know, you don't get too much snow. Here's another example. And and every now and then you'll see a false alarm. Now remember, it's doing this all instantaneously. It doesn't remember anything from the previous image about there being a person in here. Here's another example. I'm not exactly sure why they chose these examples, but you know, whatever. Now, you see where it's losing that person in the background? One of the problems there was that there was no gradient between the side of her dress, and the, the dark background behind her, so it sort of failed in terms of the signal. And here you can see, you know, pulling, pulling out some things and not others. A, a lot of work has been done to improve this kind of stuff going forward. Pedestrian detection has become a big deal in computer vision. Partially because we have databases to train on and to test on, so you can compare your results to somebody else's.

8 - Non Linear SVMs

So that was pretty cool, right? You basically take this great big long feature, based upon these HOG features, Histograms of Oriented Gradients. You take a whole bunch of positive examples, whole bunch of negative examples, you throw it into this training machine. You go get a latte. And then another latte because it takes a while and it comes back and it says here are your support vectors. Here are the alphas. At testing time, just build yourself a feature vector, just take the dot product. [SOUND] Multiply it by the alphas. Need some other sound effect. [SOUND] And sum them up. [SOUND] I don't know. Anyway. And then if it's greater than zero, call it a, a person. If it's less than 0, not. What if the data are not linearly separable? What if I can't just draw a line, okay? So let's explore this, all right? So some data sets, you know? Linear, linearly separable, to start out with. So, here, I have them along the line. So I got my pink points on one side. My, my blue points along another. And yes, I could draw a boundary. And I would have a support. I, I would have a margin between them. But what about something that looks like this, okay? So here again, I've got 1D, all right? And I've got my pink points here, but I have some pink points there. And clearly,

there's no single place that I could draw a line and say all the points to the right are pink, and all the points to the left are going to be blue. So what do I do? I'm going to map to a higher dimensional space. Doesn't it sound cool when you say that? Map to a higher dimensional space. Well, here's a very simple example. All right? So here I just have the pink points along the single dimension x. What if I plot them in 2D. You might say, where does 2D come from. Where does 2D come from? She's on the ball now. Well how about I just take x and x squared, okay. So I just take x and x squared. So these points are the same points, right. In other words, well actually, this shifted over a little bit, this point should be right there. Yeah, these are pretty much, right? These points are the same points, but I just took whatever their x-value was, and just drew it on the x squared line. Okay? Well, that looks nice. Let's erase that. Now, is there a line here that separates the points? You know, a line that looks something like that? Well, if I push my magic button, There it is, okay? And there's only three little support vectors. One here, one here, and one there. And all of a sudden, it's magically delicious. No, magically separable, okay? So, that's a pretty cool trick. It's about to get even cooler in a minute. Here's another example. Here we have a bunch of points. This comes from Andrew Moore. Pink points in the middle. Blue points on the outside. And in the 2D space here, is there a line that separates them? No, of course not, but there would be this curve here, right? As drawn there and that has something to do with sort of how far away it is from the center. Well suppose I map, using some function cap phi, the points x to phi of x. And suppose that puts it into a space where along here is the distance to the origin. Now, and it's a little hard to see the idea is that this plane in here is separating that. And the key to making this work was this mapping function, right. So we map phi of x, so it takes x from its, its original space into some higher space. That's actually called the lifting space but it just, it's to a higher dimensional space.

9 - The Kernel Trick

Now comes the cool thing and it's so cool we call it a trick. I mean, not too many things in science are called a trick, and this is called the kernel trick, all right? We saw that a, the linear classifier only depends on dot products, remember that? I circled it. I jumped up and down about it. So let's define some function K. Of two points xi and xj as just being the dot product, okay, between them, all right. If I map a data point to some high dimensional space, so my function phi here maps x to phi of x, all right? Then the dot product. Whoops, I actually should write it this way, right? The dot product k of xi, xj would become phi of x, transpose phi of xj. That's what's written right there. Okay? And in fact, what I'm actually going to do is, I'm just going to redefine k, instead of just being the dot product of xi and xj, I'm going to have a new k, right, which is just this dot product of the, the higher dimensional space. K stands for kernel, okay, and it's a, it's, it's a kernel function and it can be thought of as a similarity function. Similarity functions the idea is that they get bigger, the more similar things are and you know, dot products can go from, you can even think of it as minus 1 to 1, it's true. So the idea being the most similar is 1, the least similar is minus 1, okay? And a kernel function is a similarity function that is the dot product, sometimes called the inner product, of the higher dimensional space. Okay. So, I'm going to show you an example in a minute, but basically the idea is, if I give you a function of x1, of xi and xj, some function K, as long as there's some higher dimensional space in which that function is just the dot product of that higher dimensional space. That kernel is a dot product, and therefore, it can be used in our linear classifier. There's a whole bunch of theory there about what are called Mercer kernels, which have to do with what are, what types of kernels are acceptable. You basically have to end up with these positive definite matrices between your points a couple of other constraints. What, most of that doesn't matter. What matters is that you have to is, is you, is that you can show that essentially it's a dot product and it turns out a lot of functions are.

10 - Example

So let me show you an example. This example comes from Andrew Moore and it's a really nice example. Suppose my vector x is just two-dimensional to start with, and I, I need to be able to say exactly how many there are, so I can show you the whole thing written out. Okay? Let me define

my kernel function as this. Okay? So my kernel function is just xi transpose xj, 1 plus that squared. Okay? This is referred to as a polynomial kernel, because I take 1 plus the dot product of xi and xj and I raise it to a power squared, so this is a quadratic. Let's write out what that actually is, okay? So, we need to show that this function, in order for it to be a kernel that we can use for the SVNs, it has to be a dot product in some high dimensional space. I don't actually care what that space is, but it must be a dot product in some space. So, just for this, th, demonstration we'll actually show what that space is, but in general we, we don't, we tend not to worry about that, all right. So, here I just wrote out K, so it's one plus xi transpose xj squared. Multiply this whole thing out, okay, so just using the fact that x is only two-dimensional and it's x1 x2. You get this, you know, big, ugly expression. And that's only squared. Imagine if it's raised to a higher power or imagine if x, instead of being just x1 x2 was, you know, x1 through x5, right? You get all of these terms, all right? Here, we only have, one, two, three, four, five, six, six terms, okay, because it's squared. But that expression can be written as the following, okay? And you have to, sort of work this out for yourself, right. So let's just take a look at this term, okay, well, 1 times 1 is what? One, very good. So here I have x i squared, x j squared. Well, if I put xi squared and xj squared here. xi squared times xi1 squared times xj1 squared is xi1 squared xj1 squared. This sounds like a Dr. Seuss thing okay. What's going on here is now let's take a look at this term here. Right? These two, times each other, is that, and so on. Basically, this big expression is just this term, which is made out of the xi components, dotted with, transpose, the same term made out of the xj components. Okay? So the way of thinking about that is that's a dot product between the phi of xi and phi of xj, where phi of x is just defined by this. So here the idea is, we went from being a two dimensional space to being a one, two, three, four, five, six dimensional space. Now, we don't actually need to know that space. We're just going to use the Kernel function in our SVM and the way we're going to do that, is we're going to use that, that kernel trick. As, so instead of computing explicitly, that high dimensional space, right, we just define some kernel function, which we know is a dot product in a high dimensional space, but we don't actually even look at it, we just know that it is. In which case, we convert our xi.x for testing our new vector x to just the kernel of that. All right? And so we don't actually have to do the thing in the high dimensional space, we just compute the kernel in the lower dimensional space.

11 - Examples of Kernel Functions

So what are some examples of good Kernels? That become generals. [LAUGH] Anyway. All right. Fine. Well, the simplest kernel is actually just the original linear space. Right? So if. K of xi, xj is just the dot product of xi, xj. That's what I was showing you before for the Dalal and Triggs Pedestrian Detection. Right? You just have a great big long feature vector and it, I don't even have to map it into a higher dimensional space. I'm just going to assume that that's going to work. Now and the number of dimensions is what? It's just N, where N was the length of the feature vector. So that's the lifting space. Now something that I'm not going to teach here even in some higher dimensional space. So when you pick, your system might not be completely linearly separable. Right? And there's ways of learning support vector machines in that situation. I use what's called Slack Variables. For those of you take a machine learning class, you learn a whole lot more about how SVMs and Slack Variables work. But basically what it does is it says, let me try to find the maximum margin possible. Allowing for the fact that some of my points are going to end up on the wrong side. Okay. So that, that's how you handle that. So even if you have this great big long feature vector, if things aren't totally linear separable, you're okay. So that's, when you have a large input feature vector, often its the case that people use linear SVMs. So heres another kernel. Okay. That people use, and its very commonly used. And it's a Gaussian kernel, sometimes called RBF, Radial Bases Function. So it's a kernel where just as, as point moves away from the support vecrum as xj moves away from xi it just falls off like a Gaussian. Just like a Gaussian bump. Now, you might wonder, I did. How is that exactly a kernel? Well, and, and also because that means that there is some space in which this exponent, exponential, is a docked product. Okay? And I realize I had learned this before and I had forgotten it. That this dimensionality of that space is essentially infinite. And there are a couple of ways of, of thinking about this. But the one that's the simplest is

it turns out you can write that exponentiation or that decay exponential this way. Okay. So here is my exponential the sigma square rule, sigma square equals one. And notice this is just some particular value, so you just, the magnitude of x. This is magnitude of x dot. So for some x and x dot, it's fixed. But notice that you're taking dot products, and you're taking an infinite sum of them. So it's the dot product, raised to this j power. Let me remove that. Right? So, it's the sum of, and the these terms come out because they're constant. It's the sum of this dot product raised to the j power, j equal to 0 equal to infinity. Do you remember that e to the x was an infinite exponential series? Right? So, way back in Mrs. McGillicutty's you, maybe you didn't learn this in McGillicutty's Algebra class. Maybe you learned it in Miss Thompson's Calculus class about infinite series and, right, that an exponential, e to the x squared, was x plus, x squared over two factorial plus x cubed over three factorial, in an infinite sum. That's, see the factorial here Here on the bottom. That's related to the, to the sum that's going on. So the idea is that this Gaussian kernel actually has infinite dimension in its lifting space, but you know what, we don't actually have to do that. We can just use the Gaussian kernel.

12 - More Examples of Kernel Functions

There's another interesting kernel that's used in computer vision I want to mention it explicitly, because it, it works so much better than what people might've tried originally. Often in computer vision we'll build histograms of some distribution to describe something. So, one we'll actually talk about it just a little bit when we talk about video activity recognition. You might build up what are called a bunch of video code, visual code words. All right? So, you you find a whole bunch of interest points in video or set, and you say well, they're all different kinds. But let me create a whole bunch of buckets, maybe a thousand different buckets, and I'm going to map each one of these points to being in to one of those buckets. So, if you give me a chunk of video, it could be described by the histogram of how many of each visual code word show up. So, I've got two different pieces of video, and I want to compare them. I want a similarity metric between these histograms. So, people used a variety of things. You can think of them as distributions, so you can use KL divergence or symmetric KL divergence, or Bhattacharyya distance, which is another was of computing, of comparing, densities. But Jitendra Malik and students came up with what is called a histogram intersection kernel. And the histogram intersection kernel is incredibly simple, all right? So your xi's and xj's, they're now the vectors of the histogram. And you just sum up over all of them, the min of the two values, right? So in bin one, whichever one is smaller, you take that. Oh, and by the way, the histogram's been normalized to sum to one. And if you take that value and you sum them up, you'll realize the minimum value is zero, right? If, if one bin is empty when the other one is full, always take the empty ones. So, I can sum up all zeros. And the maximum it can be is one, because if they're all identical, then it doesn't matter which one I pick. And the histogram has been normalized to sum to one, anyway. So, it's got, so it can have a large number of dimensions, if you have, for example, a large number of code words that your histogramming. And it's very effective, and, sort of caught people a little bit by surprise, I think, because people had been doing things like Euclidean distances on, on histogram, which, turns out to be a bad idea. And this is just much more effective.

13 - SVMs for Recognition

So let's just summarize. Using SVMs for say recognition, okay? So you have to define your representation that is you need some base feature space, all right? Some feature vector in which you're representing. So whether you're doing HoG or, or whatever it is. You select a kernel function. I'm not going to tell you at the moment yet how you do that. Given a kernel function, you can compute the kernel between all your inputs, all Xi's and all the Xj's. That matrix is what you can then use to find your support vectors. Remember to, to find the support vectors, it has to be able to compute the dot product of Xi and Xj, but in this case, we're going to use the kernel of Xi and Xj because it's the dot product in some space. And what's cool is you'll notice here that there's a boundary that's not a line. It's a line or hyperplane in some high dimensional space, but back in the

original space the boundary is whatever it is. And in fact if you use RBF, a radial basis functions, you can sort of see it move around your points elegantly. And that's part of what gave the power of SVM, is that you can essentially bend the surface where you need to near your support vectors. So, training. This is how you do the training. And then at recognition, to classify, it's very simple. You take a new example. You compute the kernel values between it, and only the support vectors. Multiply by the weights, the alphas. And if it's positive, it's a, it's a positive example, negative net. So, here, it's sort of faces and non faces.

14 - Learning Gender with SVMs

Let me show you one of the earlier examples using this in a way that was kind of intriguing. Moghaddam and Yang, they did some nice work on gender classification, okay? So given a face, can you tell me whether it's a man or a woman? That's it, that's it basically. And what they were doing is you can see they were just cropping out sort of the center of the face here. In order to do that, they had some they had some machinery to take this image. I think this is still the the old Mayor, I think it's Mayor White of Boston a while back, and essentially finds the face, scales it, finds the, the, the features, can warp it and straighten it out and then cut out just the part of the face that they want, okay? So they can automatically go gener, remember, generate the candidate? They can automatically generate the candidates, all right? So they trained it. Their training examples had about 1000 men, 700 women which is a pretty reasonable ratio in academic labs, but really, they should have tried to, to balance it. The images were shrunk down to being only 21 by 12 pixels, okay? So, it's a relatively small feature. It's still got, whatever, 21 times 12 is, let's see, 210 and 42 times, so 252 numbers, right? But, they're not just going to base it on that. They actually used a Gaussian kernel, okay? A Gaussian RBF. So that's defined as xi and xj, and when you compute some new input, x against your support vectors, so here is kx, so x is your new input, and the xj is each of the support vector. So you have to compute this quantity at testing time, in order to determine whether it's a man or a woman, okay? What's interesting is, remember we had support vectors which were actually the points. All right, and these are the points that are sort of the harder points. Right? They're the ones near the boundary. Well, we still have support vectors. But now, you can call these things support faces. Remember eigenfaces? And by the way, Babock Magadon was a student of Sandy Pendleton. Sandy Pendleton, Matt Terk, the people who do the eigenface, and then Babock did his own thing on parts of faces and face space. All right, but what you can do is you can now find the support vectors which in some sense, are the the faces that are closest to the boundary, right? In some sense they sh, not in some sense, they are the faces that are sort of harder to, to label. Okay? So male versus female, you can see these are pretty similar, all right, these are pretty similar, and yet, one is, one is male, one is female. Here's another example. Maybe this one's not so hard. So, this is just some of the support vector's faces. Well, what's kind of nice about this, is that they trained a variety of classifiers, all right? Here they are, some simple nearest neighbor, which you could see down at the bottom. A Fisher linear discriminant, which says, let me try to find a single line that separates things the best, but in the original, dimensional space. Then they tried a variety of methods, and what's really cool is the SVM, all right, the Sport Vector Machine with the radial basis function, the Gaussian one, only had an error rate of like 3.4%, which is pretty good. In fact, how good is it? Well, you should see what, how well can humans do on this data set? 'because remember, they just cropped the middle of the face, all right? So, they did a test with people, where they had 30 subjects 22 male, eight female. That's also closer to an academic lab, typically. And their test data was 254 faces. 60% male, 40% female. I don't know why they wouldn't use 50 50, but that's what they did. They used low-res versions and high-res versions. We'll get to that in a minute. And then the task was, is it male or female? We'll give you all day to decide, but you have to make a choice. You can't say I don't know, and you can take as long as you want.

15 - Human vs Machine

So that was how well humans could do. If you take a look at how well machines do. Okay? The SVMs performs better than any single sub, or I should say than every single subject. In other

words, the machine was better than the best person. Okay, and the overall performance, you can see the SVM didn't care at all, really, about the low-res versus the hi-res. In fact, it had slightly higher error rates on the high resolution than low. Which and, you know, could be that difference is not statistically significant. Unsurprisingly, the humans did much better at the high resolution remember we saw was 6% and the low resolution, you know, they did awful. So the machine did better than the humans. This is something we see a lot, if you have lots of data to train. The machine learning methods can be quite sensitive to learning the distinctions, sometimes better than what, humans can do. The one you have to be a little careful of in this particular thing is, remember that low resolution image I showed you? I don't, I, I, I went back and I read the paper. On the left is the original 48 by 84 image. Okay? In the middle is a, 12 by, is the 12 by 21 image that they showed. But of course, it's blown up to be the same size, and when you blow it up to be the same size, you put these edges in here. Now we talked about this. Right? What happens if you just replicate the pixels? Remember aliasing? What you've done is you've, you've put in these high frequency things that really don't belong there. If you were to smooth that image, you would remove that aliased stuff, or you could reconstruct it better, you'd get a picture that looks like this. And I'm going to guess that this picture would have been easier for humans to deal with than this picture even though they have exactly the same amount of information, right? This picture is just the blurred version of that one. Okay? So that, that's something to, to remember. In fact, let's do a quick psychophysics experiment. You ready? All right, can you recognize that picture? Say no. No! Okay. I'm going to take that same picture and now I'm just going to blur it. Can you recognize it now? Do you remember the picture? Come here, come here, come here, come here. I'm going to make you get on camera. On camera. On camera, there you go. So, so who is that? That's you. That's my, that's your hero, you're supposed to say. That's my hero. There you go, yes. That's just, that ugly puss is me. Now you know when you were looking at just this picture, that was probably hard to tell. So you have to be a little careful on how you do things and you know, for your psychophysics. So that, that ends that digression. By the way here were the hardest examples for humans to classify, okay? So you might take a look at them and, and try to decide what you think they are. I'm not going to do that because I don't know who they are and I don't want to insult anybody. These were the actual true examples. I mean, sorry, the true labels. So did you get them right? I got one wrong. I'm not going to tell you which.

16 - Multi Class SVMs

All right, last question. Remember we said features are not 2D, no problem, just hydro planes. If the labels are not linearly separable, we do this really cool thing with the kernel trick and that puts us in a higher dimensional space, and we separate them up there. Well, what if we have just, we, we don't have just two categories, what if we have more than two? Well, you have a couple of choices, none of which are particularly satisfying. Basically, SVMs allow you to create binary classifiers. This versus that. The simplest thing you can do, well, not the simplest, one thing you can do is you can train, it's called one versus all. You can train the As versus not As. Right? And the Bs versus not Bs, and the Cs, you, you get the point, right? And then at testing time, you apply all of these machines, and you assign it to the SVM that is it's actually interesting, we haven't talked about this, maybe two of the machines say that you're on the positive side of the line, but remember that W dot X plus B, or the kernel, plus b, that actually gives you something like a distance away from the decision boundary. So, you could actually pick the one that had that highest value. The one that seems to be farthest away from its decision function. So that's one versus all. The other thing that you can do is you can do what's called one versus one. You learn an SVM for each pair of classes. So A versus B, A versus C, A versus D, B versus C, B versus D, C versus D, if you had A, B, C, D. And then what you do is you apply all of those to some point that comes in, and, essentially each SVM is trying to vote for which class it belongs to. Now, if I apply a D to the A versus B, I'm going to get some random value, right? But if I apply the, a D to the A versus D, and the B versus D, and the C versus D, I hope that three of those would vote for it being D. So you basically count the votes. It's not very satisfying either way, most people I know actually do the one versus all, so they do n of them.

17 - SVMs Pros and Cons

So, we'll talk about the pros and cons of SVMs. I mentioned before that nobody writes their own SVM code. This is both good and bad. We'll talk about why it's bad in a minute. But what's nice is their variety of libraries, whatever language you happen to use, everything from MATLAB to higher speed languages. And you can use that. And because of your choice of kernels. And then we can choose all sorts of, it's a very flexible system. So you can pull, look at a variety of representations, then given your representation, a choice of kernels. And then even then, there's a bunch of parameters. So, it's a very flexible, way of doing things. What's very nice is, after you do the training, so the training, you don't care how long the training takes. You can drink lattes all day at Starbucks, all right? But after the training, you often come back with you only have a modest number of support vectors, and that means that at testing time, it's going to be very fast. You're taking your new value. You compute the kernel function of it in each of the support vectors, sum up the weights, and if it's greater than 0, it's a positive. If it's less than 0, it's a negative. All right. And then the one, the biggest pro is it just tends to work extremely well in practice. I will tell you that there was a while that boosting, remember the Viola-Jones thing? Lots of people like the boosting. It's clear, it's elegant. You can understand what's going on. My understanding is that for an awful lot of systems, SVMs just beat boosting. There are of course some cons to it. Remember we said there's no direct multi class. You combine these two binary things in sort of an unsatisfying way. It's hard to know what the right kernel is, and it's dismaying to say which kernel did you use, use this one. Why? Because it gave me the best results. It, it's, it's very unsatisfying. And yet we don't have a lot of the, there is some theory but not a lot of theory about why you use some particular kernel. I will tell you that the radio basis function kernel. The two kernels I see used more of, actually three, are exactly the ones we talked about. Linear where you just have a large feature vector and you don't do any kernel trick at all. because you're already in a high dimensional space. The Gaussian one which essentially lets your surface, move nicely near their support vectors. And then this third one is this histogram intersection kernel. Because we have a lot of histograms that are used as features. Then I put down as a con that nobody writes their own SVMs. So that's both good and bad. Good, there's a lot of libraries. Bad is I've had students, they're doing SVM and they're getting certain behavior. And all they can do is tweak the parameters and they don't have a good intuition for what's going on underneath. And part of that is that they're using a package and they're sort of at the mercy of the package. So it's good that they're using a package, but you're, but you lose a certain intuition of to what's actually going on. It is sort of very compute heavy during training, all right? You're basically computing a matrix of kernel values between every pair, all right? So if you've got you know, tens of thousands of training examples, right? Let's suppose you had 10,000 training examples. That's a matrix, that's a 100 million big. Now, you might not do it quite that way, but the, in, in principle, you, you have the entire pair-wise thing. I guess the matrix is diagonal, so you only need half of it. That's fine. So learning, the training and learning part, that can take a long time for our large scale problems.

18 - End

All right. So that ends our conversation about discriminative classification methods. We've seen sort of the general approach to finding a window and somehow deciding that that window is class A or class not A. And we've talked about generative methods which are not used nearly as much for recognition, and discriminative methods. The ones we talked about are boosting SVMs and they have a deep history in computer vision. Recently other methods, and we mentioned some of them have become big computer vision. But one that I think is really being pushed on most these days is what's called random forests. And one of the reasons is they've become they've been shown to be incredibly effective for some problems that people might not have thought. Like labeling depth images as to whether a point comes from an arm, or a, a upper-arm or lower-arm. A leg, a torso, etcetera. And I'll tell you that's actually how the Kinect does its whole skeleton thing. Maybe we'll talk about that in a future lecture. But random forces also, you actually can write your own random forest code. You might now want to because the packages will work better. But they're pretty easy

to understand what's going on. So that's another method that has become big in discrimination, in discriminative learning. I'll also end by saying, 15 years ago when I was teaching computer vision most of the students who took computer vision had also taken graphics. There was an interesting images and things. And so they were familiar with things like Lambertian shading and other sorts of models like that and geometry. Much less familiar with things like feature vectors. Now there's been a tremendous shift because of the dominance of machine learning in computer vision, a lot of the students will take a lot of machine learning along with the computer vision. That's one of the reasons why I'm not going to go more into things like how SVMs work boosting, etc. Because if you really want to understand that, there's such a wealth of stuff you have to do that's actually a whole other course. In fact, machine learning has grown that I think the, the Udacity machine learning course that I know about currently is one flavor of machine learning, and then there's others. And in Computer Vision, you're looking at the one's that work on large feature vectors. So, we're going to stop here with recognition, and like I said, if you want to do more, you really want to do that in the context of machine learning classes.

8C-L4 Bag of visual words

1 - Intro

All right, welcome back to Computer Vision. If you recall last time I said that, that was going to end our, recognition, discussion. When we talked about SVMs and discriminative learning, well I lied. Basically what I'm going to talk about is what's referred to as a bag of words approach to doing recognition. And it, it was used somewhat in static images, a lot in video but it, it really should follow right after doing the SVM.

2 - Indexing Local Features

So, you remember how we did panoramas? We basically found interesting points in an image, and we described them. We built these descriptors, these sift descriptors. We found punitive matches, possible matches between them, and then we did something like RANSAC, or something like that, in order to find the alignment. Well, we can actually do something for recognition that's related. Now essentially what we we're going to do is to say we're going to do the same thing. We're going to find interesting points where these are sort of highly repeatable points that you'll find. And what we're going to talk about is describing the patches around those points. And then we're going to use the collection of those described patches as a way of generally characterizing the image for recognition. So, as we've talked about before, each location in an image. So here we have different locations in an image, all right. And a bunch of different training images. And each one of these has a descriptor that, in this what these curves are meant to show is that they land somewhere in feature space, so this they land in the descriptor's feature space. And you can imagine that you get a collection of these, so that there's a whole bunch of descriptors in a given image. And then we're going to make the following assumption, and we actually made this assumption before right? We assume that when we have points that are sort of similar in the descriptor space that they are or nearby in the descriptor space, that those patches are either the same patch in a new image a similar patch. So one way of thinking about this is it's a way of just giving a description of a little bit of local content in the image. And so now if I get a new picture that comes in, right? I can say let me find the patches in it. And see if I find lots of the same local patches that were in my first image, and say, okay, if I find a lot of those, then I would know that maybe this new picture is the same thing, maybe same category or same object, as the first picture. The problem, of course, is that any given picture can have thousands of patches. So a collection of different objects, and different images. There could be millions of these things. And the problem is, if I come up with a new image, and I find a patch in it. I have this problem, of possibly having to search over millions of different patches. I have to do something a little more clever.

3 - Inverted File Index

So the question that we have to address is if we've got these sort of thousands of features per image, and maybe millions or hundreds of millions of images or patches. You know, what's an efficient way to search to figure out which image had about the same sets of patches or at least has a high overlap in the set of patches that I find in a new image. And what we do is we take a little bit of inspiration from other systems that have lots of little instances in them, namely things like words and indices, right. So if I've got a book, it would be really hard to find all the examples of where it says Karl Marx in that book by flipping through that book. So what do we do instead, we build an index. Right? There's an index in the back, it's an efficient way to find all the pages that contain a particular word. Okay? So what we want to do is we want to find all the images from a known database that have a particular feature that we find. So now what's going to happen is I'm going to find a bunch of features that are in a, new image. I'm going to look up the different images that have those features and maybe if I find an image that has a lot of the same features as my new image, then I say, aha, maybe it's the same kind of thing. And this comes to the notion of what we call visual words. Okay? So, the featured descriptors we sometimes refer to as visual words and we're going to get to code words in a minute. But the idea is that, and I think really this notion of words comes from this whole analogy with text processing. Right? So we've got words are the local features and documents are the entire image.

4 - Find Likely Page Quiz

Let's say we have four pages of text. Now we pull out all the individual words from these pages to make an index, just like the end of any other book we list out the words, in say, alphabetical order. We then mark the pages on which each word was found. For instance, say, and was on pages one and four, bread on two, bus on four and so on. Now say you're given a test set of words, gong, fly, pepper, ant, shoe. Can you mark the most likely page this set of words came from?

5 - Find Likely Page Solution

Okay, this is fairly straightforward. Let's use the index to tally page counts for each test word. For instance, gong appears in pages two and four. Fly is in one and three, okay? We're even. Pepper is in one and two, and one and four. Oh, and shoe is in one and three. Clearly, the winner is one. This is a very simple and naive bag-of-words approach. But sometimes it works pretty well. For more realistic cases, you might have to do something slightly more complicated, like matching word distribution histograms. But the central idea remains the same for both text as well as visual words.

6 - Visual Words

Here's an example from Sivic and Zisserman in ICCV. And the idea is that here we've carved up feature space, sift or otherwise, and the idea is that I've got in this particular case three different code words. Here's a particular example. So they are brought in from real images. So you see one of the code words that was found, I'm going to draw an example, is basically a dark patch with a light thing sticking out of the corner. And these are taken from lots of different images. So that's like the eyeballs that we saw before. And the idea is that all of the points here are an example of that. So you could just say, how many of these code words show up in a particular picture?

7 - Bag of Words

At this point I'm going to skip over a bunch of details. For example which features are we going to use to make their code words? What size should there be when you do a vector quantization? Sometimes it'll tell you how many code words it thinks it needs or sometimes you give it a number. We're not going to talk about that, and we're not even going to talk about the clustering or the

vector quantization algorithm. Instead we're going to focus on how would you use that in a way that relates to the SVMs that we just talked about in order to do recognition. So, what we do is we're going to continue with our analogy to documents. So here I have two different documents. And if you take a look at the document on the left you'll see a whole bunch of words in here. Right, sensory, brain, visual, perception, et cetera, nerve, image Hubel, Wiesel. Hubel and Wiesel, by the way, are people who won Nobel Prize for understanding what goes on in the visual cortex of the cat, all right? And here's another document. China, trade, surplus exports, domestic, foreign, increase, trade, value. Notice that what's in the magnifying glasses are not ordered. It's just the words that happened to be in that document. It's like you stuck all the words in a bag. And that's where we get this notion of bag of words. Okay, and the goal of what we're going to talk about now is to go from object recognition and map that onto a bag of words. And in the document world, you know, one way of doing this is I have my bag of words and I say, what is the distribution of words in this document? And if I wanted to do document retrieval, I want to find other similar documents, a simple thing you might do is find other documents that have sort of the same histogram distribution of words. And that, that's likely to be a matching document. Things are much more sophisticated now in terms of how that works. But it's, the basic idea is that I've got these code words that represent or these words that represent something about the semantics. And so a similarity in distribution tells me that, maybe there's a good chance that it's a similar type of document. So we want to do this in computer vision. So that's notionally represented here. So I've got a bunch of different objects and I compute my features over all of these different objects. And I come up with what's shown down here, and I think this side is originally from [FOREIGN]. I'm not sure. And all of these, these are all of my code words for all of my different objects, right? So I take all my objects, and it's kind of like if you take all the documents in a, a library in United States, and if they're all in English, you took all of those documents together. All of the words would be the set of the words that span all of those documents, right? And in fact there are many more documents than there are words. And in fact if you think of important words, words that actually have a lot of meaning, there's way more documents than words. So given a collection of words, visual words, for every document, I can say how many of each word is in each document. And that's what these histograms are meant to represent, right? So you know, the bicycle has got these little bicycle pieces in it. All right, and something that might kind of look like this. I don't know, maybe, you know, this brown part down here looks a little bit like that brown part down there. But the idea is that there's many more of the bicycle elements than there are of the, the violin elements. This is referred to as a bag of visual words, for obvious reasons. And the idea is that you're describing the entire image just by the histogram or the collection of those words. And the, and you want to do access the same way that you did in the documents world.

8 - Comparing Bags of Words

So one way of doing this is, just compare that distribution to the distributions of the word, of the, of the images that you know. And you can do it as say, like a, a scaler.product, right? So just a histogram. If you normalize it to one, you can think of it as a normalized vector, and I can pair, unit vector, I can compare unit vectors, for example, by taking dot product and finding nearest neighbor. So, that's illustrated here. So I have some exampled, d sub j and I want to, I've got some query, q, and I want to compare them. And so, here are my two histograms. So my histogram here is 1, 8, 1, 4. My histogram over here is 5, 1, 1, 0. So remember, it's the same set of words, right? We're just going to say for all the words in English, what's the histogram? For all the words in my collection, what's the histogram? And I'm just going to compare, say a similarity. So here is one example of where I take the dot product between the d and q, divide by the length and that's essentially giving me a similarity between them. And this is just written out for i equal 1 to v, where the idea is that you have v number of words. And again, we didn't talk about exactly how you decide if what v should be. That's sort of part of the art of doing this work. So that's essentially doing a nearest neighbor look up, right? Given some input, nearest neighbor, all right?

9 - Object Classification with Bag of Words

But when we date discriminative classification we said, you know nearest neighbor has some power, but, it can be kind of expensive in terms of storage and access, and maybe there are better ways we can do things. And in fact, we talked about training classifiers, so we talked about boosting or SVM. So, we can do that for this example, too, right? I have a whole bunch of images of violins. I've got a whole bunch of images of people. I've got a whole bunch of images of planes. I take all those images together, I find all the visual words. All right, that gives me a vocabulary. I can build a histogram for each of the objects. And you know, so I've brought a bunch of training data, so I got a bunch of histograms for planes, I've got histograms for cars, histograms for faces. And what do I do? I just train a support vector machine to make the decision, you know, is this a, a person, a plane, a car, et cetera. And in fact that's exactly what was done. This work comes from 04. There was a database, still is a database referred to as the Caltech 101. It was one of the original datasets of enough images of enough object types to start to explore this notion of database access. And, and what they were doing was they trained exactly a linear SVM, we talked about those before, using a bag of words on vectors and there were a couple of different classes. Faces, airplanes, cars, motorbikes, et cetera. And what's being reported here is that for each of the actual faces, you know, out of, you know, the average percentage of time. That it was the best rank was 94, 96, 90 so you can see that it's pretty good. And then the other thing that they report is the mean rank. So when I try to find, you know, whether or not, you know, when I show a plane and I say well, do I think it's a plane, do I think its a face, do I think its a motorbike, et cetera? You know, how what is the average ranking of the correct answer? And you can see the value is 1.0 something. Meaning that it almost always got the answer right. But it was just an example of this bag of words approach that says, I don't know anything about parts, I don't know anything about structure. What I know about is images and I take locations and images and I find interesting points, I build some description. Give me a word-like collection of those descriptors. And then for any new image, try to find the descriptors that are present. And that's what's referred to as bag of words.

10 - End

So this concludes our little addendum to object recognition. And what it will do is it will let me later talk about, well actually, it will let me later not talk about activity recognition being done the same way, where instead of doing it over just images, you do it over video. But code words are a very, sort of, standard approach in computer vision these days, and so it's important that you know about it. So I, I went and stole some new slides to make this just for you guys. So I hope you appreciate it.

8D-L1 Introduction to video analysis

1 - Intro

All right, welcome back to computer vision. Today we're going to do just a little bit on video. Today and I guess next one will be our, we're only going to talk two of these lessons on video. One could do almost an entire class on computer vision as applied to video and the things that are different. But we're, we're not going to do a lot of that. A lot of what you've learned can be applied obviously to each of the individual images. And some of what you have plearned, what you've learned where they're essentially two dimensional operators. You can apply those same operators in three dimensions. Or other ways of thinking about video. But we're only going to do a little. In fact talking about the three dimensions. Video, we're going to think of as a three dimensional signal, all right? So video as, as shown here is, I have this image x y and as time moves forward, time could also go that way, it is that I've got this volume of data. And so a pixel inside a video is actually a function of x, y and t. All right so sometimes you'll see me going like this for video and I'm talking about x, y and t. You know that, that's the way I think of video: I think of video as a volume of

data. So, let's talk a little bit about some basic computer vision in video. So, one might be object detection. So, suppose your goal, and we'll talk a little bit about this next time, is activity recognition. Right, you want to recognize what's going on. So you're going to recognize the activity of objects, of people that's there. But, but there are these entities doing something, so you might have to find these entities. That is, what are the things in here that are moving, that are doing something that I need to try to label what their activity is. You know, nominally, or notionally, you can think of as my goal is to find the independently moving things that are in this video. So here I've labeled this thing called a background subtraction, and it's you know, just example. I mean here you have a, a scene where there's some cars moving, and the idea would be well, you know, we'd like to be able to find all of these moving blobs, all right? In this particular case, all these moving blobs are the moving objects. Generically, background subtraction is used a lot. It used to be that people thought that background subtraction was trivial and then they thought it was hard and then they stopped thinking about it at all. It still, this notion of separating out the entities is still a fundamental research area in computer vision. We're going to talk about some simple stuff here. So, the, the most straightforward background subtraction is where you have a static camera. So the camera's not moving, and the idea being that if nothing is changing in the scene then nothing is moving their either and there's nothing to do, so you should go home and get some rest. All right? Background subtraction as sort of simple and overly simplistic as it is, it is used a lot in commercial systems. Traffic monitoring where they're, you know, counting cars, or, or monitoring other things going on. Various human action recognition where you have people coming by and you want to be able to say something about walking, running, whatever. Human-computer interaction. So, people will build systems where humans interact with a big display, and to pull the humans out from the background, they have to remove the background somehow, and sort of object tracking in general. So, sometimes there, what you'll do is, you'll compensate for the motion of the camera, so you'll move the whole image back. As if you had the camera sta, stationary and then you do background subtraction on that.

2 - Background Subtraction

How do you do background subtraction? Not so complicated. You have to estimate the background. So at some time t, you remember doing x, image of x, y, and t. I have to estimate the background, and then I'm going to subtract that background from whatever the current frame is. And wherever the difference is, big enough I'm going to call that my foreground mask, okay, and it's an absolute difference because the background could be brighter than the thing that's moving or it could be darker. Here's an example, sort of notionally, so here again same image, right. And this is my estimated background, we'll talk a little bit later about how you might get such a beautiful background. And the idea would be to take your current image, subtract off the background, take the absolute value, and anywhere that it's bigger than the threshold, just output that. All right, fine. But of course the question is, what's a good estimate of the background? There's a form of background subtraction, which isn't really background subtraction at all, and it's referred to as frame differencing. Frame differencing is exactly what you might think. I'm going to assume that the background is whatever was in the picture just before. Okay? I'm just going to compare the current image to the last image. All right? And, background subtraction becomes, then, I of x,y, and t minus I of x,y and t minus 1. And if that value is greater than some threshold, I'll output that as my foreground mask. Okay? So here again, two examples. So here's I of x, y, t. Here's this van here. And van has moved a little bit. Let's see, which one's easiest to see that something has actually changed? Oh, here. So you see that this car is moving forward that way, and it's now, it's now moved past this line. But at time t minus 1, it was even with that line. Same thing that this van is moving forward. In fact, pay attention to this white van there, okay? I'm going to subtract from this image that background image, and then I'm going to take a look at different thresholds, right? Even before we look at the results, you can just think a little bit about how well will this work. Well, clearly this is going to depend upon sort of what the objects look like, how fast they're moving, what the frame rate is the threshold, so, you know, sometimes this is good. And then the slide that took, you know, [LAUGH] it says usually not. Well, and this can be shown here, all right? So if I

take different thresholds, so here's threshold of 25, of 50, of 100, and you can see, you know, for this illumination, I don't know, somewhere, a threshold somewhere between 50 and 100 seems to be going pretty well. You remember that white van that I told you about? Okay, well that's right there. Okay, so you can see the front of the van, you can see the back of the van, but there's a big hole eaten out the middle of the van. What happened there? Well, duh the top of the van is just white, all right it's just plain white. So the van tops here and then in the next frame it's moved there. So the part where the van was and now you can see the road behind it, that's a difference. And the part that was road, or the windshield, and now the top of the van there, you can see that, that's a difference. But, the middle of the top of that van is white, and then if I move it, what does it stay? White, so I don't get much of a difference. Okay, so this is a problem with, with frame differencing, is that large areas even as they move. They can move relatively quickly, but if you have large, constant, intensity areas, you know that's just not going to work very well. So, what's a better idea?

3 - Mean Filtering

So an obvious better idea is just what's referred to as Mean filtering. Mean is my background now, instead of just being the last frame, which is really silly is just going to be the average of the last end frames. That's what it says here. And my background at time t, is just the average of the sum of n frames in the past. Okay? In which case then my background my, my, my mask is now I take my current image subtract off that mean and that is greater than the threshold. Notice we now have a new parameter, right? We now have this parameter n. We have to worry about how much do we average? So here's an example on the same scene of using n as n equal 10. All right, so it's average. Now you'll see, it's, it's kind of a weird thing, right. So this, this van that has you're kind of blended in there, right? As it averages over different places and my Foreground mask looks kind of like this, all right? And so I'm getting this extension of the cars in, in a variety of ways and it's not so great. And the reason that it's not so great is that the average isn't really the right thing, right? So averaging over all those vans is getting me this kind of weird thing, right. Now of course if I averaged over 50, you know, but then I'd have all these other cars coming through, the average is very sensitive to this. What you can think of is that, the background is just sort of a constant value, with maybe a little variation and occasionally, I get some random perturbation, right? Some random noise comes along. We've seen this before. Remember when we were doing noise filtering of images? And we were using Gaussians, etc., and then we put in that like salt and pepper noise. And we said salt and pepper noise, the problem with it is every now and then you get this big spike in there. And in average it'll get tweaked around quite a bit by that. So what was the solution to that? The solution to that was to use a median.

4 - Median Filtering

So median filtering for background subtraction does exactly what you might think. All right, now we're going to assume that the background will be modeled by the last end framed but I'm not going to take the average. I'm going to take the median value. What's kind of nice about that is now n can be kind of nice and long. Let's suppose the average, let's suppose when I look at the, the road. Let's suppose its value is 57. Plus or minus a little, 57, 58, 56, 52, 57, etc,. Then comes along a white car. [NOISE] And it's like got 112, 119, 194, fine. Then comes along a black Porsche. Slowly because he's being cool, you know it's three and not. But if you take a look at all those values, the one that's going to dominate is going to be the 57 to 56 etc, so when I do a median filter, that's the one that's going to get pulled out. So now your foreground mask is we just take the current image and we subtract that the median of the last end frames, okay? And again, threshold. Your n by the way now is a little less sensitive. I mean it has to be big enough to capture it. There is, as you know, those of you know about in your computing, a running average is easy to compute. You just have to know how many frames you've, you've added into the average so far and the previous average and then it's 1 over n of the new 1 plus n minus 1 over n of the old 1. A median, you have to actually keep them. And then resort them, and that can be expensive. So, unless you'd play some interesting tricks, median filtering can be expensive, but it, but it's, it's not

too much of a problem. All right. When you apply the median, well, it turns out if you use short n, it might not be quite so great. So, for example, here we have the estimated background and you can see a lot of stuff has gotten cleaned up, but because of the small n I still get this problem in terms of where the van is and you're doing a better job getting the van but you still get a little bit of the stuff

5 - Median Image Computation

So let me show you a better example of doing it. So here's some input frames taken from, I don't remember which campus that is, okay. And this is the background model, and that's a pretty clean background model. You'll notice most of the people walking around have disappeared. These people, of course, are still here, because they've been here the whole time. Nothing is here, because these people have, have walked through. When I subtract the two images and just take the sort of magnitude of the value, it's a little hard to see here. I get sort of the general value here. All right. What I can do is I I can threshold those. I say, which points are above a particular value? And those are the points that light up. I can then run, what's called a morphological operator. We're going to do that in just a couple of lessons. It's just a way of sort of cleaning up binary, and what I mean by binary images, zeroes and ones. Right? I can then grow those regions a little bit and I can apply that and look at that. This has pulled out all the people moving around in this area nice and cleanly. With the exception, of course, of these people because they've been there, sitting there, so long that they just look like they're part of the scenery. All right? And that's what happens with median filtering. Anything that stays there for a long time just becomes part of the scenery. That's actually considered more of a feature than a bug. Right? So somebody parks their car and pulls in and stays there for a while. At some point maybe you don't want to keep tracking it as an, as an independent object.

6 - Pros and Cons

So the, the pros and cons of, background subtraction. It's very easy to implement. It's easy to do in real time, fast, and the background models don't have to be constant. They can change, so we could sort of track that change. The disadvantage, of course, is, you know, frame differencing is a real problem, depending on how things move. The median background can be a high memory requirement, we talked about keeping all that. And of course, you have to build a set these thresholds. When does this basic approach fail? Well it fails a lot of times when we can't really touch that, but one, one idea that has come up that has really gotten a lot of play is that, suppose your background actually has a couple of different things that might be in it, right? So, the example that, that I'll show you in a minute you're looking at, water and sometimes it's plain water, and sometimes it's a specularity. Or you might be looking at some areas where there's some foliage, and you might get, you know, one color from the leaf, and one color from the thing that's not so, so these pix, pixels are essentially in the background. They're twinkling between a couple of different values, all right? And that analysis was the basis of a paper that's now very well known by Chris Stauer and Eric Grimson on doing adaptive background modeling. And here, they're showing you a couple different examples. So we'll just do them, one is the water here. And the idea is that the water, most of the time, it's kind of this dark color and that's shown as this distribution as sort of a low value. This is you can think of this as an RG space. But when they get bright specularities, you get this high value. Likewise, this area where the door is, well when the door is opened, it's bright because it's light, it's lit by the, the, the light in the hall, in the office. But when the door is open, it's much darker and it's over there. So the idea is that there's multiple values in the background. So what they did was they said okay, we're going to model the background as a mixture of Gaussians. That is, we're going to say that the background could be made up of a couple of different distributions of intensities, and we're going to learn those distributions. So if a pixel comes in, if it comes from one of those Gaussians, I'll say it's background. And, if it isn't, if it's new, then I'm going to say I think that's a new object, I might hold on to it to see if it becomes background, but that's how you do the detection. One of the cool things about that, I'll tell you that early on when papers like this were being done like I said, people thought background subtraction was sort of

trivial. I think this paper is one of the paper that won the Longuet-Higgins prize for being most cited ten years after it was published, because it just became a really important piece of work.

7 - Background Subtraction with Depth

One last thing about background subtraction. These days there are special kinds of cameras, that instead of just giving us an RGB value, they give us, RGB and D, depth. So you're familiar with the, the Microsoft Kinect, because unless you're living under a rock, you know about the Microsoft Kinect. That's not a rock as in a Iran, it's a rock as in granite. So anyway RGB-D cameras, you can then do background in depth, right? So you take an image, and the depth is fixed. That is, the lights might turn on and off, the sun might cause a shadow to move by, but the background is fixed. So, so in, in terms of its, it's geometry. So you can actually do background subtraction in terms of geometry. So you subtract on the d channel, and that's pretty cool and, and very powerful.

8 - Epipolar Plane Images

So, that's just a little bit on some initial video processing where you have a static camera moving object. I'll show you something else that's kind of cool. What if you have a static scene? So, the right? But your video is generated by a moving camera. Right? What sort of interesting analysis might you do there? Remember, we talked about when we talked about stereo, we talked about motion parallax and disparity. The disparity is inversely proportional to depth. So, the farther something is, the less disparity it has, the closer it is, the more it moves as I move. So, and you probably thought of that as motion parallax in other contexts. So, if I have cameras that are moving, looking at a scene, I could pull out the separate objects by just looking at the relative motion. So, faster relative motion means objects are closer to me. All right? This was first done in what was called epipolar plane, epipolar plane image analysis, or just EPI image analysis, that was done by actually Bolles, Baker and Marimont. Bob Bolles who his, his friend and some, by the way, it's the same Bolles from RANSAC. So, he's got his name against couple of pretty cool algorithms. So, here's a cute image that was taken. This was back in the days when you used to draw images because we couldn't actually get computers to make them. So, the idea was, what if you have a camera that's looking sideways and you can extend this to looking arbitrary ways but it's easy to think about looking sideways. And here is some point on what it looks to me to be, I, I don't know if it's a taxi cab or a police car. And the idea is that this point, of course, as this camera moves this way, this point will move across the image. The rate at which that point will move across the image is, depends upon how far away it is, right? So, a point that's here will move faster, and a point that's out here in infinity, will move slower. So, if you go to their original paper, you'll see pictures that look like this. This is an indoor scene, and what you can see is that the camera is moving just a little bit this way. Right? because here, you can see this thing is occluding that thing there. But then, as the camera moves, I can now see through there. All right? So as the camera's moving to the left, this object is essentially moving to the right. So, you can take all of those images and you can stack them up and here they stack them. This is x. This is y. So, this is the first image that we saw. And this is time. Okay? Now, this little edge here. You know what that is? That's the right most column at each of the times. Now, this might just look like another picture to you, and it is, because if you look over here, this right most column is just as the camera moves, it's just sweeping across here. So, we just pull out each of those columns, you'll get another picture. And so, that's what you're seeing here, that's what this side is, it's the right most column. But what's really cool about this volume, remember, we took videos of volume of data? Is I can slice that volume of data, right? And so, suppose, we just take a look at a cut through there. So, that would be, that's what this would be here. And you see that image there, okay? And you'll notice, and we're going to do this more specifically in a minute, that I get these streaks and these streaks are at different slopes.

9 - EPI Gait

And to show you exactly what's going on here, I have a little scene. I've taken the images, tried to animate them so you can see what's happening, okay? So here's a very sort of ugly scene. It's sort of color coded by certain regions. That doesn't really matter. What matters is the camera now is going to move to the right. All right, it's a posterized co, color scene so you, so you can see these things stand up a little better. Right, so that's image one, that's image two, that's image three, that's image four. So going backwards, so the camera's moving to the right and of course, the, it's not a Rubik's cube, whatever kind of horrible evil, diabolical puzzle that is moving to the left faster cause it's closer. So if I were to take all those images and build a cube of them, this is that same cube, okay? So here we have X and Y, and we stack them in time this way. And here's just the top of them. And I'm just going to take a slice, one of the slices through there, and that looks like that. So now it's at a fixed y value. This is X, and this is time, and here are my streaks again. Okay. And notice that some of these streaks, okay, move at different slopes. Those slopes correspond to the depth of the scene. And what's kind of cool about this is it's really easy to find these kinds of lines, okay? And you can find lines that intersect that, that aren't parallel to the other lines, and so you can use that to pull out the object, all right. This is called EPI, EPI, image analysis. And you can use that in some other ways also, so I think one of the coolest ones I ever saw was done by in the same some we spoke about before. Again, now almost 20 years ago, they had a scene where you had people just walking past you. Okay. And imagine you've now got a volume of video. Okay, and you stack that up. And now I'm just going to show you some chunks taken from, sort of, the top, the, the head area, and then by, down by the feet. At the area that is really at the very top here, nothing changes in the camera. So, as time goes this way, the pixels stay constant, okay? But at the slice where their heads are, okay. You see these lines there, the, these regions here? That's the movement of their heads through spacetime. That's what this volume is, it's spacetime, right. And so each one of these diagonal lines is the movement of their heads. And these straight lines here, what are those? Those are just the background just like up here. Those, when things don't move, right? And the camera's moving, these, the, the pixel value in time stays the same so it's just a straight, ray and spacetime. But if they're moving then you see them moving across in time. Now the really cool thing is what happens if you look at their feet. Right? So if I look down at this level, okay? Here's what the spacetime image looks like. And you see this braided pattern? That braided pattern is the movement of their feet in time. In fact, they even published a paper, here's just some images taken of it showing the same thing, the slice through the head and the slice through the, the feet. And what they were able to show, and here it's just sort of lined up, is that you could actually recognize who the person was with a small sample set by just tracking that braid, all right. So look, if not how you would do gate recognition in a really good discriminative way and things like that. But it's an example of showing you the data structures that you can get from processing video.

10 - End

That ends our sort of little introduction to video analysis. And what we'll do for the next lecture is, we'll take a little bit of a look at how people do activity recognition in video. One of it will be some slightly more modern method, which is really just like a discriminative classification. We'll just do that real short. And then there's one that I really enjoy. Why? because my students invented it. But it, it's become, useful, about a way of describing human motion, which you could then apply the discrimination methods to, so, the idea is that it's a representation of the action.

8D-L2 Activity recognition

1 - Intro

All right. Welcome back to Computer Vision. Today what we're going to do is having just started talking about Video. We're going to start talking about, Action Recognition or Activity Recognition

in Video. That's the processing of video. For generating, generating some sort of label or, or descriptor that tells you what's going on. So, there's a bit of a challenge in activity recognition in terms of terminology. And there's no sort of universally accepted terminology, and I'll say some of this goes back to the 70s, Hans Nagel, first started talking about events and actions in history and things like that. So, loosely, we're going to use the definitions that I have here. So, we talk about Events. And the one thing about an event is it's a single moment in time. Right? So the moment a door closes or something changes right, right in the imagery. A typically we refer to that as an event. A little bit higher up perhaps thinking is this basic idea of movements or sometimes referred to as actions. And these are I'm going to describe a, sort of atomic movement patterns. So you know, sitting down. Or you know, waving or something like that. We've got a gesture-like, it's a, you could describe it as a trajectory in some feature space. Whether it's the feature space of the joints of the body or some appearance. It is it's a, it's a single thing. And then at the higher level there's this notion called Activity. By the way the only reason it says adopted from Venu and, and myself not together I did the original paper on movement activity and action and it was wrong. I mean wrong in the sense of in, in hindsight had activity followed by actions because I was thinking of actions being a big scale thing. And the field which like the paper for a little while has sort of moved on and we have this notion of, of movements. And then actions and then relay activity. And activity is being sort of composed of a series of actions or a bunch of actions happening at the same time. Typically if you're looking at some interactions among the group of people that we thought of as an activity. So one example. Here's a thing taken from some surveillance, work. Some of this was shown at the PETS conference lets see. Performance Evaluation of Tracking and Surveillance, I think is what PETS stands for. There's been a lot of work lately on just you know finding the person in the image here and have extracted them there. And maybe looking at some interactions between people in a train station. The idea of basically being able to watch an environment sometimes to be able to inform authorities if there's possible problem going on, sometimes to do things forensically. Right? So something happened and I'd like to look back over the last eight hours of video and find different kinds of things.

2 - Human Activity in Video

So, there are a couple of basic activities of doing activity recognition. One, I'm thinking of as model based action recognition. So, you have some way of getting the pose, or the appearance of a human body. And, you're going to relate these descriptions of the change, or the pose, or appearance to some action labels. And, probably build a classifier, or something like that. And one of the main challenges in doing that work is, are your data that you use from training taken from a different context than the data where you're doing the actual recognition? These problems tend to be pretty easy. If I have training data from the place that I actually, and the camera, in which I want to do the recognition. And, are much more difficult if the training data come from a different source. Model based activity recognition basically assumes I've got some low level process for either detecting actions, maybe probabilistically with some noise. And then, what I need to do is recognize the overall activity by comparing it to some, I put structural representation of the activity, something that defines how the different actions relate, one to the other. One of the reasons I sort of like talking about it this way is it's sometimes reasonable to get lots of examples of low level actions. So, you can do our traditional building a classifier to do that. But, harder to get lots of examples of, the overall activity. There's some change in that lately in terms of using large data sets. I'll talk about that in a little bit, but in some sense those are a some what artificial problem. When you do this, you have this problem of handling uncertainty. That is, your low level detections are uncertain, and your descriptions vary. And, you have to be able to recognize the activity in the presence of this uncertainty. And, some of the major challenges are you know, just, can you keep track of which part is which? Who is who? If you've got multiple things going on in a train station, there's lots of ambiguity and occlusion, and being able to say what the activity is is sometimes difficult when you're subject to those those variations. Recently, there's been some work where you actually don't do as much about that, you just view activity as sort of a big space-time pattern of some sort. So you're not actually tracking bodies, or, you don't even know there are people there. You're basically

just describing the overall pattern of motion, and you typically train a classifier. We'll mention that just a little bit here, but, not spend a lot of time on that. Also recently, and this was kind of, I, I found it kind of funny, there's been doing activity recognition from a single image. And, you might ask, you know, how is that possible? How is that possible? Okay, thank you Meagan. I didn't really want her saying anything. Yes, how is that possible? Well look, here is a picture. Okay? So you see that, that says, imagine a picture of a person holding a flute. Well, you don't have to imagine it, there it is. You could say, what are they doing? Right? And, you could say, they are playing the flute, right? Playing a flute is an activity. So, even though I'm only recognizing a particular pose, in fact, when I go like this, you could almost hallucinate the flute being there. Right? Even though it's not really there, and you can think about whether that's a vision problem,or an AI problem, or a hallucinogenic problem. But whatever it is, the idea is that you recognize the activity from a single pose. And, I don't really think of that as activity recognition. What I think of that is sort of representative image recognition, right? This is, this is sort of the canonical image that you have in your mind of the pose of a person doing that activity. So, you're recognizing the pose in the context. You're not, you might even be recognizing the fruit, the fruit, the flute, as is here. I guess you could play fruits too, I don't know. You can recognize the flute, and that helps you see that they're doing flute playing. But to me, it's, it's kind of far cry to to call that actually activity recognition in terms of anything like video.

3 - What Were Not Going to Cover

So let me tell you what we're not going to cover. In exactly the same way we talked about bag of words for recognition of objects, where you cre, you, you have some way of finding interest points in an image, and then you cluster all those interest points into code words. Remember that? And then you describe each image by its histogram of the code words, and then you build a classifier to recognize these histograms? Well you can do the same thing in video, right? So you can, you can pull out these little interest points. This guy is moving, and his hand is swinging, or over here, or maybe this car is moving. And that would build you, you could extract some features. But now, remember the volume in video, your feature points, your interest points are in space time, instead of just in space. And you could pull out little spacetime patches, sometimes called cuboids, in, in some of the work, where they're just these little chunks of spacetime imagery, and that builds you your descriptions. You can then do feature quantization to learn codewords. You do this over all your videos. Then for an incoming activity, you can look for the interest points, you can build a histogram of these codewords, and then you can train a classifier. And here they talk about using an SVM with a chii squared kernel. The reason we're not going to talk about that, although it's good work and interesting work is, it's in some sense, only the three dimensional, from 2dxy to 3dxyt. It's just the three dimensional extension of the previous work done on bag of words, or things like bag of words. Instead, we're going to talk a little bit today, and then also, next time, on some representations that fundamentally are about how things move, or about time series.

4 - Motion History Images

So, I'm pretty sure we showed this before when we were talking about motion and perceptual organization. And as it says here, even an impoverished motion data can evoke a percept, and presumably, you all see these moving dots as a person walking, right? So, you're not recognizing this as a person from the appearance of the dots. You're recognizing this as motion and then you see that it's a person. And in fact, I can prove that to you looking at some work that was done by myself and Jim Davis now professor at in Ohio. And so I'm going to show you here. So when you take a look at this picture, unless you've seen this demo before, you probably don't know what that is. When I put this thing in motion, what do you see? What you all probably saw, was somebody sitting down. And what's interesting is, now when the video is done, you can sort of see that, oh, this is a person standing up right here, right? In fact, there's some sort of a blackboard behind them, and then once they sit down, they're head is here and the body is here, because there's actually a chair over here. But until that motion happens, you can't actually recognize that. So we did a little

bit of work on some technology that would let you recognize this, and we created this thing called motion energy images. So here you see a picture of a person, this is a clear video, all right, where the person is sitting down. That's, by the way, Claudio Pinhanez, who also got a PhD in Computer Vision, now doing great stuff down in Brazil. And what these are representing are cumulative images of how, of where stuff has moved, okay? So the idea is, when you're all done, this is area in which stuff has moved. And, you know, the question is, that's somehow indicative, that something about that pattern and how it moved in there is indicative of the actual motion pattern itself. So, can we recognize that explicitly? So we created this thing called motion history images which becomes the motion energy images also. Which are a function of the temporal volume, and what I mean by temporal volume is that same thing, right, x, y, and t. So I've got a collection of video. And it's really a very simple recursive computation. This is an MHI image, motion history image, and the way it works is quite simple. As each new frame comes in, in this particular case you run some algorithm to tell me, which pixels are moving in that frame. Now, of course, this is under the assumption of a static camera. And if you're not using a static camera, you'd have to compensate for the motion of the camera, threshold out the motion of the person, all the usual caveats. But then when you do that, you get this very simple operation. If a pixel is moving, set that location at time t, to some maximum value tau, and you're going to pick tau. Maybe tau is 15. I'll tell you why 15 in a minute, right? And if it's not moving, that's the otherwise, I'm just going to decrement, whoops, over here, decrement the value, okay, from the previous time, but I don't go any less than 0. So the image on the right is an example. The brighter pixels like over here, they've moved more recently. So this is time t, maybe they've moved at time t minus 1, they just moved before. So they started out as tau. So maybe tau is 15x, let's make it 30, that example will work better. It was 30, so now it would be 29. But the pixels up here, they move, the last time they moved is 15 frames ago. So its value would be now, tau minus 15, so if tau were 15, it would be 0, so that would be black, that's why it didn't work so well. But if tau were 30t minus 15, it would be now down to 15, right? It would have counted down to 15. So these areas are nice and bright white because they just moved. These areas are darker gray because, it was a longer time ago that they moved. And so the way you can think about this motion history image, is that it both shows you sort of where stuff is moved. And it also shows you sort of how stuff is moved, because you can tell that the stuff up here moved a long time ago, and stuff down here is moving more recently. It's a recursive filter, that is, I only have to keep track of i of tau of x, y, and t minus, of t. And I can create the new one when the next frame comes in. So I don't have to remember my old frames. The other thing is, where did tau come from? Well, the pixels go from tau down to 0. So, let's suppose I'm bringing in 30 frames a second. So if I set tau to 30, then from 30 to 0 is the range over a second. And what that would mean it that anything that happened longer than a second ago, I'm not worrying about. So I'm recognizing action that's length of a second. Suppose you wanted to check whether something could happen slowly, right? Like, he sits down slowly, it takes him a full second to sit down because he's really slow. Or somebody sits down quickly, right, because they go like this and they, it's not Megan playing with the video. Anyway so they can do it in a tenth of second, so sorry, in a, in, in a, a third of a second, so that'd be tau of ten, right, 10 frames. Well one of the kind of cool things about motion history images is if you give me the motion history of image of tau, right? So everything goes from let's say 30 to 0. I can just, if I wanted the motion history image for tau equal to 20, I could just subtract 10 from it, right, so 30 becomes 20. And anything that's less than 10 just becomes 0. So it only looks back 20 frames. So it's trivial to compute I I of tau 20 from motion history tau 20 from motion history tau 30. So I only have to keep one around, whichever one's the longest one, and I can check all of those time windows at the same time. And then also as should be clear, if this is the motion history image that goes from white to dark. If I just wanted the motion energy image where anything has changed, all I can do is just threshold that picture, right? Anything greater than 0, make a 1, otherwise it's a 0.

5 - Temporal Templates

We can take these two images together, these motion energy images and this motion history image, and together we refer to them as what we call the temporal template. It was just the model of how

stuff moved. All right? And then we tried it on a dataset that we created. We had this person who was way more fit than we were do a whole bunch of different aerobics, different moves. And we just said, okay, can we now recognize if somebody is actually doing a new routine, which aerobics move she's doing. All right? And one of the interesting things that you can see here, what I'm showing you here are the MEIs, all right. I'm just showing you the binary version. And you might say, that this shape, looks an awful lot like that shape. But if I show you the motion history image of the one on the top, you see that, and you can see, that it's dark down here, and light up there, so that's an example that the arms are moving upward. When you take a look at this one, you can see that it's high up here, but high down there, so it's actually going like this. So the motion energy image would be ambiguous. It shows you where the stuff is. But the motion history image resolves that ambiguity. The motion energy image captures the whole image motioned area and captures it nicely. So that's why it's not in some sense redundant with the motion history image, which gives a lot more weight to the more recently moving pixels.

6 - Image Moments

So here's an example of 18 different moves, and it's showing you some of the motion history, motion energy images of it. So we now have this problem of how do we recognize motion energy, motion history images. Well, these are kind of gray-scale blobby-like images. And the good news is even in 1999, there have been about 150 years, or maybe 20 years, pick your favorite number, of computer vision describing gray blobs. And trying to say, okay, I recognize this gray blob as being different than that gray blob. And that's basically all we had to do. So, basically, 1999, we did some sort of, what I think of as old style computer vision we're going to compute some, summarization statistic of that blob. We're going to compute some features of that blob that have something to do with the distribution of the, where the pixels are and their intensities in the MHI, and we're going to build the generative model. Okay. There are a couple reasons we were doing generative models. One is that discriminative models hadn't become as popular as they had been and so Jim Davis' advisor was not smart enough to tell him to go make sure you're using the discriminative model. I will tell you that people have subsequently used it, and of course, it makes your recognition better. Generative models also have the nice properties as we mentioned before, that when I come in, if I come up with a new, aerobics move, right, so I want to add one, I just have to build a model of that category. I don't have to, to retrain my entire system. Okay, whether you use generative or discriminative, the idea is that we're going to compute these statistics and then do the recognition. So, how are we going to describe this sentence? And this allows me to spend a little bit of time telling you about some very old computer vision work that becomes, that's still important, and fundamental. And that's the notion of the moments of a shape. So, so some of you from physics, you know about second order moments, right? Axis of inertia. The moment is the distribution of matter about a point or an axis or something like that. So in computer vision, the moments are defined in a very simple way, right? So the I J moment, okay, is just the sum over the X Y of the whole image of X raised to the I power, Y raised to the I power. But you're only counting points where the image is non-zero. Okay? So, if I is a binary image, then this is a one or zero, so you're only counting it at points that are turned on. If it's a grayscale image, then you're weighting the point by essentially how heavy it is, which is a little bit like when you do moments of inertia. If you've got little points that are heavier than others, they add more to the inertia without the axis. So these are the moments, and so you can talk about the second order moments, or the first order moments, or even the zeroth order moments, right? What would this be for the zeroth order moment? Well, if I is zero and J is zero then these are one, this would just be counting up the number of non-zero pixels. Which means that M zero zero is the area. [NOISE] Right? because the area of the picture that's non-zero, that's just the sums, the counting of the non-zero pixels. All right? The problem with regular moments, of course, is that they're sensitive to what the X and Y position actually is. Right? So, you kind of want the moments of a shape to be independent of where the shape is. So there's something called the central moments that just take X, but they subtract off the mean of X, and Y, and subtract off the mean of Y. And it is, otherwise just the same. So, that's this Mu P Q, and it's X minus X bar to the P, Y minus Y bar to the Q. All right? And, you

know, what is X bar? Well, one way of thinking, it's the average X, so the average X would just be the sum of all the Xs divided by however many Xs there are. Well, that's what this is, right? X bar is M one zero, the first moment in X divided by the overall area, remember the M zero zero, same thing in Y. So that's regular moments, but the problem with moments, of course, is that they're very sensitive to a variety of changes, so if I scale the image up, right, obviously, the moment's going to change, the area will get bigger, the distances get, get larger. If I rotate it things might change. All sorts of problems.

7 - Hu Moments

So there was this guy named Hu, and he created a set of moments. And what was really cool about the moments that he created, is that they were rotation, translation, and scale invariant. And you'll see I, I put scale in green here, because for what we're about to do the translation part is taking account by just using central moments. We don't really need rotation and variant that much, because people don't tend to do their aerobics at 45 degree angle. But we do get scale differences, right? If things are closer to the image, or, or closer to camera or farther from the camera, things get bigger or smaller. So we want to create some descriptors that are invariant, and Hu moments are, are of that type. And we're going to use 7 of them, and that's what these 7 Hu moments are. And I'm going to show you the equations for them. You would never write these down. You would go get an implementation of them because otherwise, you'll make a mistake et cetera. But, but here they are, okay? And in fact, they're so ugly that they don't even all fit on one page, so you see it says this is h1, and that's mu20, remember mu are the central moments, okay, plus mu02. And then they continue, h6, and then h7 is even more ugly. There they are, they're just that sort of that complicated ugly thing. All you need to know is we now have a translation, and rotation, and scale invariant system. So the details of what the, the features do are not that important except that they behave the way we want them to behave under particular types of transformations. What matters is, I'm going to compute these features on both the MEI and MHI, so the binary version and the grayscale version, and that's going to give me a feature vector. And given a feature vector, I can build classifiers.

8 - Build a Classifier

So you remember generative versus discriminative? So generative, I build a model of each class, and then I compare it against all of them. Discriminative, I build the model of the boundary between the classes. So we've talked a bunch about building some discriminative ones, let's talk again about building generative. How much you build a reasonable generative model of each class of action? Okay. Remember one of the reasons we might want generative models we might not have a whole lot of examples. So a simple thing you might do is, you've got a Hu-moment features space and we've got seven things applied to two images so we get 14 numbers. So you could use some sort of Gaussian or maybe it's a Gaussian with diagonal covariance so that you don't have to worry about having too many points. And you might use a small number of multiple mixture of Gaussians, but the idea is you would build a model, a Gaussian model, for each of those classes. And then what you'll do is you'll compare the likelihoods. You remember the likelihood was probability of the data given the model. Right so, I build all the models then when the new data comes in I evaluate the probability of the data. Then if I have a prior, then what I'm going to do is I'm going to use Bayes rule. That's what it says here. That basically says I take my likelihood, I multiply that by the prior of my model and that's proportional to my, to my total posterior. If I don't have a prior, I might just select whichever one has the highest likelihood. And by the way, if I don't have enough data to even build some reasonable model of the densities. I might just use something like nearest neighbor. So, the nice thing that we have is that mo, the motion energy images, motion history images, they're sort of little global descriptors. And the way you do this is you're going to collect these statistics on these global descriptors in order to do the recognition. One of the things that might be a little less obvious is you can't actually recognize the action till it's done. Right, so, I have this description that goes back in time so if I'm watching a person sit down, when their done

sitting down, I now have that chunk of data that represents them sitting down. Most activity recognition these days, there's not a lot of focus on doing online recognition that as as you're watching something happen. The idea of, is that they have the whole sequence to begin with. But if you're actually are worried about recognizing it as it happens, this idea that I can't recognize something till it's totally done can be somewhat concerning and you have to think about how you might deal with that.

9 - KidsRoom

A couple of cool properties as we said, that this does, it's very easy to construct the MHI for one TAL from the other. So you can go from TAL 30 to TAL 15 and that's basically a linear scaling, right. So if I have an action that can go fast, or go slow, that's okay, it's not okay for an action that goes the first half of it is fast and then the second half is slow, where I have to do something more serious, some more serious dynamic time-warping or something like that. The whole idea is that we have these atomic primitives, that can be performed quickly or slowly, but they don't have lots of parts to them. It's very, it's, recursive so it's very fast. What's interesting is, and I've had conversations with some, people who do work in vision from the neural side, or from, computational vision from biology. You could imagine building something like these mhi detectors from very simple neural hardware. Basically filters with slow responses. And, the fact that these can work from very low resolution images, which I haven't shown, but because it's just a global pattern you don't need a lot of pixels. It means that out here in your peripheral vision where you have hardly any resolution at all, when we do the human vision, lesson. We'll, we'll talk about how, you think you can see out here, but not really. You're just making it up. All right, but, what it means is a very low resolution. You can recognize things. So I'm not going to claim anything in terms of biological reality. I'll just say that there's some biological plausibility to, to this kind of work. And there's a reference in PAMI, which sort of talks more about this. We actually use this system to build a couple things. Here's Jim Davis, when we were both much younger building a aerobics system, where you would be taken through aerobics by the system and it would watch you and it would know which, which aerobic you're doing, I mean which one you're supposed to be doing. It tells you, so it can recognize it was real easy. We did have this cool little thing that the music in Jim is very clever making this happen, the music was actually MIDI controlled so we can control the tempo. Right, it wasn't just prerecorded. So that means we could watch you, and we could always make the music go just a little faster then you were going. So we could sort of, just play with your head and speed you up and make you sweat more, but it was cool that the thing would just recognize the system. A more compelling use of this technology, we built this thing called the KidsRoom, and it was an interactive play space for children. The original intent was to just illustrate how computer vision could do some interesting cool stuff. And, what was interesting is, nice article about it in Presence, but, some folks came by, I don't remember exactly how it was that they were at the media lab. But they really liked it and they ported it to the Millennium Dome, so they built this big thing called the Millennium Dome, in London, at the turn of the century, at the millennium. And they had all these pavilions, and they were looking for interesting technologies, and this is one of them that they put in there. So, I, I have a little video here that shows you first of all, temporal templates being used. So the monsters teach the kids these moves, and then, which is good, because if we wanted to we could actually get training data from the actual kids that were there, but the system is robust enough that we could use training data from just a couple of people that we trained on and it would work for everybody. We also used temporal templates plus, this little extra stuff that we used. It's all in the paper if you wanted to read about it. The, the Presence paper. And then there was a little interesting, cool thing in, interac. Well here, I'll just get this started. Okay, so here we have the last bit where the monsters are teaching, the last two of the moves, there are only four of them. First he teaches the kid to do a crouch, and then she does a crouch in response, and, he's praising how well she's doing these crouches, and that's great. We go to the point where he's teaching her to do these flapping motions, and you can see he's pleased with her. And he's giving her lots of good feedback. And then I think the last one he's going to teach her how to do is how to spin. Say something like surely love do a spin, and ready spin dude. There he

goes, he's spinning, and now she starts spinning, and compliment her on that. And when the music changes, basically they're supposed to do their own moves and the monsters will follow them but, it takes them a little while to catch on. And from now on he's following her. Now one of the thing's that we did in order to make this room look a lot smarter. The monster would say what you're doing, right? And when it would recognize it but it only did that, when it was sure it was right. If it was close maybe you're spinning, maybe you're flapping it would just do whichever one it thought was right. But it would only say it if it were sure. And the reason is, if the monster goes and does something that is not what you are doing, what do you do? You say, oh, he wanted to do something else. You just describe intentionality to the character, but, if he says, oh really cool spin, and you're flapping? He just looks stupid. So the way you avoid this interactive system from looking stupid, is it only speaks when it's sure, and it acts all the time, because you'll ascribe intentionality. We, we wrote that up in the paper. It was kind of a cool insight in terms of how to build interactive characters

10 - End

That was back in the late 90's, and what was really cool was this holistic notion of, of the human motion. We also didn't have a large database. You have a small number of, of examples, so it's not sort of the large discriminative database learning kind of stuff that goes on in action recognition now. What we're going to do next time is, we're going to take a look at a different approach that takes some, look at, looks at videos as a time series, time series analysis. And just some basic approach to doing that. And again, this will be something where much like the SVM's that we just talked about, we said we can only give you a sm, small taste of it, and if you want to know about it, you should take the course in machine learning. We're going to show you a very simple notion of a form of a graphical model approach to doing action recognition, and if you want to do more that that, again, you'll take machine learning. But that's what we're going to do next time.

8D-L3 Hidden Markov Models

1 - Intro

All right, welcome back to, Computer Vision. Today we're going to talk, a little bit more about some activity recognition, methodology, technology. In particular, we're going to focus on Time Series, and, mostly, we're going to, this part's going to be, on something called the, Markov Models and it's, and its way of representing Time Series. And then we'll talk a little bit about how that is applied to vision.

2 - Questions One Could Ask

So here is a, what's referred to as an audio spectrum. And at the top up here that's a, a waveform. That's the actual signal coming in, that's, a voltage is a function of time going into a speaker. And what this is, this is what's referred to as a audio spectrum, short term for a decomposition. Basically, you've got low frequency stuff down here, high frequency stuff down here with time going that way. And the question is, that's a sound, what sound is that? Obviously, it's the song spectrum. It's the spectrum of the Prothonotary Warbler, or, or War, Warbler. Obviously, I'm not a bird person, okay? There's a picture of one. But the idea is, if you listen to this bird chirp, and then you process it over time, or actually, you listen to it chirp over time and you process it, you would see this audio spectrum. And the spectra of different birds, of course, are different. Here's two different birds, the Prothonotary Warbler, and the Chestnut-sided Warbler. But what you can see is that the spectra are really quite different, all right? And, the goal is to think about these things as what's referred to as time series, and how would you distinguish between them. So, for a given time series, there are questions you can ask, like which bird is this, okay? So like, you've, you've got, you've got an audio signal, which bird is it? How long is this thing going to keep going? Is this bird sick? Right?

That is, you know, somehow not normal. And are there different sort of elements or different phases that the book, the, the bird has. And when you think of these in a time series context which bird it is, that's a classification problem. Time series classification, how long will it continue, might have something to do with prediction. Is this bird sick? Would be an outlier detection. And finally, you get the notion of segmentation. So that would be for like bird data. Well, how about other time series data? Here's some. So, you know, here's stock data of different types of indices over different amounts of time. And you might have questions that you would ask like, will this stock go up or down? Something we all want to know. What type of stock is this? Right? If you had a set of categories of stocks. And, is this stock behaving the way it normally does? And that becomes just like we saw before, prediction classification, or outlier detection. Here's another example. Supposed you were listening to the audio of this music, right? So not looking at the imagery of the score of, of the, of the music itself as shown here, but actually listen to somebody playing it. You might ask things like, you know, is this Bach or Beethoven, which would be classification. Can we make more of that? That's now a generative model kind of question. And computer music people are always interested in listening to some music. Could I have a machine produce more music of that sort of type? And can we sort of divide this up into, into segments? So this is what's referred to as time series segmentation. By the way, this bottom one is, segmentations receive a lot of attention lately. Something called conditional random field is being applied to it. We're not going to talk about that, we're going to talk about mark-up models. But I just wanted to point out that you might not use what we're going to learn today so much for segmentation as much as you would for some of the classification things.

3 - The Real Question

All right, so you might ask, get ready Megan, you might ask what does this have to do with vision? What does this have to do with vision? Well you might have some video of something happening such as a person moving their hands around to wave, or point, or control something. And you'd like to be able to say, you know, which gesture is this? Or you know, what action are they doing? And the stuff I'm going to talk about today is sort of some of this time series work, in particular Markov models, apply to the decoding of visual kinds of information. So the real question is, how do we model these problem? How do we, you know, how are we going to build, I showed you these time series. Well, we need a model of these. And in particular, we're going to have to give it a bunch of them to be able to construct that model. So it's basically going to be an inference or a learning problem. The learning is, learning the model and the inferences while given a set of models, which one do I think is most likely to have generated the data that I'm seeing?

4 - Markov Models

So now let's talk about Markov Models. And my apologies if you already know what Markov Models are, because we're going to go through this a little bit, then move into HMMs and maybe that'll be new for you. And if it's not, you can just write in a postcard telling me that you're unhappy. Send it to Megan. All right. We're going to pretend for the moment that weather, as in the weather outside is a Markov Model. And if it were, I'll also pretend that there's only three types of weather you could have. You could have a sunny day, a rainy day, or a snowy day. It's kind of like a little kid's book. And the Markov Model says we have to specify, for any given one of these, what's the likely next day going to be? So for example, suppose I tell you that if it's sunny today there's an 80% chance that it'll be sunny tomorrow, 15 that it'll be rainy, 5% that it'll be snowy. Likewise if it's rainy, I may tell you well there's a 60% chance that it'll rain tomorrow, if it's raining today there's only a 2% chance that it'll be snowy. So it's a place that gets very cold so, so once it's rainy, when it's rainy it's not going to snow and vice versa maybe. And 38% of it will be sunny, and likewise, if it's snowing there's a 20% chance that it'll keep snowing, 75% chance that it'll be sunny, and for the same reason before, very unlikely that it'll rain. This is referred to as a Markovian system. And in particular, if the only thing you need to know to make a prediction about what the weather tomorrow will be, is what the weather today is, that's referred to as a 1st Order Markovian

System. So I don't look at today, and yesterday, and the day before. I'm just looking at today in order to make a prediction about tomorrow. If I was looking at today and yesterday, that would be a 2nd order Markovian. All right? So the probability of moving, of being in a particular state of getting to a state, depends only on the state I'm currently in. In order to specify our Markovian Model we need a couple of things, right? So, we need a set of states and these are labeled here as S1, 2, through SN. And those, the S's, those are, there's state one, state two, state, that's not a particular time, or particular day, there's three different states. So in this example, N would be three. We also need transition probabilities, and that's written as AIJ. So that is the probability that if you're in some state I, that the next time you would be in state J. And then finally to kick things off, we need a initial distribution. That's just the probability that Q1, and I'm going to talk about Q a little bit more in a minute, that Q1 is equal to Si. So, Q sub t Is whatever state I'm at, at time t. So q sub 1 is the time, is the state at the first day, or time one. So for this particular example, as I showed you here, you've got three states, sunny, rainy, and snowy. We've got the state transition probability. So, this matrix A, that's the encoding of all of these links, of all of these transition values. And then, I have an initial distribution which just says well to start off with, maybe, 70% chance that it's sunny, 25% chance that it's rainy and, if my math is right 5% chance that it starts out as being snowy. So, given this Markovian Model, you can ask certain questions. For example, you could say if I give you this series so that's sunny, rainy, rainy, rainy, snowy, snowy, you could say what's the probability of getting that series? Okay. Well you have all the information you need. In fact it's written down here below, I've got A and pi, right. It's just, the probability that I start out sunny, that's here, times the probability that it becomes rainy given that it's sunny, so that would be here, times the probability of rainy given rainy which is there. All the way through, I just have to chain it through, right? And when I multiply all those number together, 0.7 times 0.125, times [INAUDIBLE] I get some number, 0.0001512. Okay. They tend to be small numbers, by the way, probabilities and HMMs, or in, in market models, because whenever I multiply a lot of numbers that are probabilities together, since all numbers that are probabilities are less than or equal to one, when I multiply a lot of them, I get small numbers. And when you implement these things, you sometimes have to be careful and do things like take the log but that's all I'm going to say about that. All right, so that's a Markov Model.

5 - Hidden Markov Models

But much more interesting than a Markov model for what we're going to do is what's referred to as a hidden Markov model. So here's the basic idea. Suppose you actually can't observe the state, kay? So there's some state doing its thing, but you can't observe it. But what you can observe, or what are referred to as observables, or the evidence. So let's suppose for the same weather example, okay? Let's suppose the things that we can observe are essentially what women of the day happen to be wearing or carrying. So maybe I see this very sheik bathing suit on a person, or I see just a nice coat that can be worn any time that it was sunny, or an umbrella. So these are the evidence of what the state is. So before I can go further, I have to tell you one more thing about our Markovian system, and that's what's referred to as the emission probabilities. Emission, if you think of it as symbols being spit out, and the emission probabilities are written like this. B, sub j of k. And what that means is, the probability that you'll see symbol k, that you'll see a bathing suit, given that at some time you're in some particular state, okay? So what's the probability you'll see a bathing suit, given that it's sunny? What's the probability you'd see a bathing suit, given that it's snowy, all right? Those are called the emission probabilities. And what that means, our entire system when it's running, looks like this. In this new system, where we can't observe the states, we don't ask questions about the likelihood of having seen a particular sequence of states, because we can't see the states. Instead, we're going to ask about the likelihood, or the probability of seeing a particular sequence of observations. So that's shown here. So you could say, given this sequence of observation, coat, coat, umbrella, umbrella, bathing suit, umbrella, umbrella, what's the probability of that series of that sequence of observations? Well, we could evaluate that if you give us everything, right? So here, I've just written it out as P of O is the sequence of all the observations, and I know the way to, to compute that and we'll get more of the details later is, I can say well, if

you told me the sequence of states, then I can tell you the probability of given observation. And if I multiply that by the probability of that sequence, I can get the whole thing. So one simple example is, well, one possible sequence of states, just one, is that it's all sun. All right, and that can be determined from the yellow stuff here, all right? All right, so it's the probability that I start with sun, 0.7, right there. And then I follow that with a whole bunch of sun, 0.8, point, right? All e, sunny days. So, this is the probability of all sunny days, and this would be the probability of seeing this sequence, if every day was sunny, right? So, the coat, it's a 0.3 probability t, there's 4, umbrellas, and there's only a 10% chance of seeing an umbrella when it's sunny. Notice by the way, I wrote 0.1 to the 4 here. The order doesn't matter, right? because I'm assuming that if you tell me the state, that I could tell you the probability of being, of, of having an umbrella. It doesn't matter what the day before or the day after was. You tell me that it's, that it's raining, I can tell you the probability that they're carrying an umbrella. But this was only for one particular sequence of all sun. You have to worry about all possible sequences of weather, over those seven days to talk about the probability of getting this one series. So to make that a little clearer and a little easier, let's just do the math.

6 - Specification of an HMM

So to make this a little bit more formal, we're going to specify sort of all the elements in HMM. You've already sort of seen them. N is going to be the number of states, so S1 through SN is our set of states. So that's like sunny, rainy, snowy. And then there is a notion of a state sequence written as Q. And the idea is that Q sub i is whatever state I'm at at time i, okay? So when I say Q sub 3 equals S2, what that means is at time step 3, I'm in the second state, all right? Given the state I can talk about the full specification of an HMM. And for whatever reason, HMMs are referred to as lambda. It's a triplet made up of three components that you've all ready seen. There's a state transition matrix, those are the A's, the probability of going from one state at one time, given that you were at another state. There's the observation matrix, which before, I was showing you as a discreet, and was actually represented as a matrix. There's also a continuous version where your output is you have a distribution over some future vector for each state. So maybe it's you know, you're out putting a, some sort of a measurement. And what you say is I've got one Gaussian in state 1, and a different Gaussian in state 2, and a different, different Gaussian in state 3. The idea is that for any given output, you can talk about the likelihood or the probability of given that output, given that you were in a particular state. And then the last thing you need is that initial distribution pi, right? because it says, what's the probability of starting at each, in a particular state?

7 - What does this have to do with Vision

This is about the time when you might say, what does this have to do with computer vision? What does this have to do with vision? I'm so glad you asked, all right? Well, another way of phrasing what we've been doing is, given some sequence of observations, which model was most likely to have generated those? So using our previous clothing and weather example. So, suppose you're seeing some sequence of clothing, right? So you, you check the clothing that people are wearing each day, and you see this, coat, coat, whatever. And then the question you want to ask is, is this Philadelphia, Boston, or Newark? The idea is that presumably, what you've been doing is, you've been collecting sequences of these observations from Philadelphia, a bunch of them from Boston, a bunch of them from Newark. You have a trained a model, and we'll talk about that later. You've created a model that will tend to generate sequences, like you see in Philadelphia, or in Boston, or Newark, with the same statistics that they have. And then given some new observation sequence, you'd like to say, you know, which one is this? Now, notice that if we were checking Boston versus Arizona, you basically wouldn't need the sequence. It doesn't rain in Arizona, for the most part. And you could say, well do I see umbrellas, okay? And since it's always lousy weather in Boston, I live there, if you see bathing suits, you know it's actually Cape Cod. So the idea is, you know, just from the individual elements themselves, you could determine whether it was Boston or Arizona. The whole idea of representing the time series is that there's something about the sequentiality that

matters. It's the statistics of one thing following another, that is indicative of the actual class. And what you're doing is, you're building some models of those time series, so you can then given some observation sequence, you can say which one's most likely. And what we're going to to do is, we're going to apply that to computer vision.

8 - The 3 Great Problems in HMM Modeling

All right. Welcome back. I hope you took a nice break there. As you can see, I'm, well, I'm totally, actually, it's, it's another decade and I just kind of look the same. So what we've done is we've laid out sort of the general idea of a Markov process, a Markov system, and also a hidden Markov model. And what I want to do now is talk about sort of three computational problems of hidden Markov models. And then we'll talk just about a simple application of this to, computer vision. Also, just as a word of note, I'm not going to go into the detail of different parts of, of lots of the math, just one part of it and then the rest I'll, as they say, I will leave it to the reader or the viewer, as an exercise to go get, schooled up in it. Because, the details were less important than knowing that it could be done and then it was applied to computer vision. All right so, we have a hidden Markov model and we say that our hidden, hidden Markov model is a lambda, is a triplet of A, B, and pi where A is the transition probability, B your emission probabilities, and pi was just the distribution where things started. Well given that, a HMM, given a machine, there are three computational problems we can look at. And the first one is this. Basically, given some observation sequence, so remember o1 through o cap T, are the observed outputs, from 1 to cap T. So that's a particular sequence. You could ask what is the probability, P of O, given that machine? That is, what's the likelihood that that machine generating that particular sequence? And this is really the classification, recognition, it says problem but actually how you use it for that, right? Assume I've got a couple different h m m's, maybe three of them, and they each represent an activity. And let's assume that the prior of which activity is going on are equal. All right? So then, once I've, if I have the priors are equal, all I care about is the likelihood of the data given the model. And the HMM would allow me to evaluate the likelihood of the observed sequence for each of the given models. Whichever one was highest, that would be the one I picked. That's a generative model like we talked about before. So the evaluation problem is, substance the key problem at run time. It's one we'll actually talk about. Another problem is what's called decoding the sequence. So, remember you have a sequence of observations, and the question is, what is the single most likely sequence of states to have generated that sequence of observations? So, it's not all possible ways these observations could generate, it's just the single most likely sequence, sometimes called the decoding problem. And sometimes you care about that, sometimes you don't. When you do it basically has to do with saying, well if I, if the states have some particular meaning to me. Right? Remember we talk about sunny, rainy, cloudy, so you know they might have some specific meaning. I might want to say, okay here's what I think the underlying state sequence, is, actually sometimes in, genome work, etcetera. They actually want to know what the underlying state sequence would be. But if you are actually just doing it for recognition, then that's, in some sense less important. And then the third problem, is in some sense the most complicated problem. And that is, okay, I give you a bunch of training examples, a bunch of these O sequences. Train model. Learn the Lambda that and the way you, that you learn is you pick the one that maximizes the likelihood of getting that observation. It's a maximum likelihood, method. It's actually called the EM, Expectation Maximization, and we'll talk about that just a little bit. We won't actually, do the math of the, of that part.

9 - Naive Solution

So, let's take a look at problem one. That is, what's the probability of getting the observation sequence, given lambda, given an HMM. It's the first I want to look at a naive solution, and then, you'll see how the cool solution is so much much better than the naive solution. So, let's start off by pretending, assume for the moment, we actually know the sequence of states. Okay? So, that's q sub 1 through q sub cap t. Right, q sub 1 through q sub cap t. That's the actual state at time one

through time, to each one of them is an s sub j, right? So, maybe it goes from state three, to two, to four or whatever. One through t is the time, okay? So, we're going to assume that. And the other thing we're going to assume is that the observations only depend upon the state that you're in. So, when we say, independent observations, what we mean is that if you knew this state, then the probability of observation is just dependent on the state and not dependent upon anything else. Well, if that's the case, then the probability of the particular observation, observation sequence, given the state sequence. Well, because of independence, it's just the probability multiplied times each other of the output of that, of the particular symbol seen at time t, given that you're in the particular state at time t. And that taken the whole product. And that's just these b variables, right? So, b of q1 that of o1, means the probability given that I'm in q1, whatever q1 happens to be, of outputting o1, and et cetera. Okay? So, if we knew the state sequence, then we could just multiply this to get the observation of the sequence, of the likelihood of the observation sequence given the state sequence. But for any given state sequence, I can tell you the probability of getting that state sequence. Right. It's just a very simple product shown here, right? So, if I know it's q1 through q cap t, then it's the probability, here's pi of q1, of starting in that state. And then, a of q1q2, is the probability of going from that state, q1, to whatever the next state, q2 is, times a q2q3, all the way up to a qt minus one, to qt. All right? So, assuming I knew the sequence, I could tell you the probability. Well, you say, in that case, I can do the following. Given that I can compute the probability of a sequence, of an observation sequence, given a state sequence. And given that I can compute the probability of any given state sequence. Well, then, obviously, we can compute the probability of getting that output sequence by just summing over the all the different state sequences. Right? Taking the probability of the state sequence times the probability of the observation given the state sequence, summed, as I said, over all the state sequences. Okay? You could just do that. Of course the problem with doing that is it would take you a long time. How long? A really, really, really long time. And the reason is, this is summed over all possible paths. All possible state sequences. All right? Well, their end states and their T times steps, that's N to the T. Okay? Exponential in time into the T. And each one costs t calculations because I have to figure out the probability of oh, so the complexity of that is T times N to the T. That, as we say, is a really long time. So, it's not such a good solution. So, you think we can do something better? Well, of course, we could or else I wouldn't be standing here wasting your time.

10 - Efficient Solution

Here's the efficient solution. The trick is that we're going to use recursion. And I'm not going to show you something called the trellis diagram, which is a, a way of thinking about a graph model of this. But what we're going to take advantage of, remember the Markovian property? The Markovian property says that the probability of going to some state only depends upon where I am now. It does not depend upon how I got here. Right? So if today is sunny, the probability of tomorrow being sunny is not affected by whether yesterday was rainy, snowy, or sunny. That's my assumption. That's my first-order Markovian assumption. So to take advantage of that, we're going to define some auxiliary variables and define a recursive algorithm. So the first thing we're going to define is what's called the forward variable alpha. Okay, an alpha sub t of i is defined to be the probability of seeing all the observations up until time t, and at time t being in state i. Okay? That's exactly what it says in this book. In well, now there's a box inside the box there. So it's the probability of seeing everything from 1 until t and being at in type t being in state i. The really cool thing about that is, I can define a recursive definition for this very easily. All right, so you start off a1 of i is what? Well it's the probability of having seen the observation at time 1 and the probability of being in state i at time 1. Okay, well that's easy. Pi sub i is the probability of starting in state i, and b i of o1, is the probability of seeing that first symbol if you were in state i. So that's how you initialize it. Now comes the really re, cool thing, and before we write it out, think about it this way. At some time t, the probability of landing in state j is going to be, well, either I was in state 1 before and I ended up at state J or I was in 2 before or I was in 3 before or I was in 4 before. Right? And these are what are referred to as mutually exclusive, so the probability is just the sum of them. And that's how this is written here, right? What this says is that alpha of t plus 1 of j. So that's the

probability at time t plus 1 of being in state j and having seen all the 1 to t plus 1 observations. Is just, well, just as I said before, it's the sum over all the different possible states, right? There's n possible states of possibly being in those states at time t. That's what alpha t is, right? And alpha t also includes all of the observations, up through t, times the transition probability of going from i to j. Remember, we're trying to compute alpha t plus 1 of j. Going from i to j. So that's the probability that I land in this state, j. And now, I also have to talk about the output symbol, at time t plus 1. Well, that's b sub j of ot plus 1. All right? And there's no way of thinking that I can reach that j state from any of the preceding states. So I just define that recursively. That means I can do one more really cool thing. Okay? So I could obviously run this all the way to t. So I could tell you the probability of seeing all of the observations, 1 through cap t, and landing in state 1. And all the observations and landing in state 2, and landing is, suppose all I want is the probability of the observations, right. I don't actually care what state I end up in. Well that means that the probability of the observation given the machine and that's what we're trying to compute, is just the sum of all the alpha cap T of I's, right? Alpha cap t is the probability of seeing all of the sequence, of i meaning and landing in state i. I sum that up over all the i's, and that's the probability of the entire observation sequence. So you recursively solve that problem. So remember before it was n to the t, which is a really, really awful number, the complexity of this is just n squared t. Okay? Through the, the recursion. Each time step, you have n squared things you have to look at, you do it t times and so it's n squared t. And for any reasonable numbers, n squared t is way, way, way, way less than n to the t. All right? So, that's that efficient solution of the forward, using the forward algorithm and allows us to recover, solve that first problem.

11 - Rest of HMMs in Words

The main point of what I just showed you is that instead of looking at all possible paths, by using recursion, we can look at n squared t paths. And remember, the reason we can do this recursion is that, that, that recursive form, formula only looks back at the previous state. We're taking advantage of that Markovian property. And with just the forward algorithm, we can evaluate the probability of seeing that entire observation sequence, given the machine. So for example, and I haven't yet told you how to do this, if we had trained up a bunch of HMMs, and some new observation sequence comes in, I could have told you which one was most likely using just the recursive, just the forward recursive algorithm. Right? It requires the As and the Bs, but that's what it means to have trained up the HMM. So I want to tell you about the rest of the problems, but I'm just going to tell it you in words. I'm not going to give you the math and you can go look, there are A bunch of tutorials on HMMs out there if you want to do the math. They're pretty straightforward, all right? So, the forward recursive algorithm that I just showed you, can compute the likelihood of being in state i at time t. And having seen all the observations up to time t, given that you have the HMM lambda. You can similarly define a backward algorithm. All right, just sort of starts from the back, goes backwards instead of the front. And it's a recursive algorithm that computes the likelihood of being in state time t and seeing the remainder of the observations. Again, given the, the HMM.

12 - Hmmmm

So, or hm, that's a joke. Did you get that joke, Megan? You got the joke. She's quiet today, which is an unusual thing for Meg. Here's how it works, if we have the HMM, so if we know the HMM, we know a and b and pi. We can use the forward algorithm and the backward algorithm, and this is, part I'm not going to show you. To estimate, that's what's cap E. The distribution of what state I'm likely to be in a time t, okay? because one has to do with the sequence is forwards, one has to do with the sequence is backward. I can evaluate the probability of all, of the whole sequence. That allows me to estimate, what's typically called gamma i of t, the probability that I'm in state one at time t, state two, state three, all right? I estimate that distribution. Well, if you give me that distribution, and I've actually seen at every time t what was observed, then given the likelihood that I'm in each of the states, probability that I'm in each of the states, and the observed data, I can figure out what the most likely b sub j of k. That is, what do I think the emission probabilities are? And I

picked the ones that would maximize the total observation probability. It's no different than saying maximum likelihood, you know, if, if I, if I'm drawing something, and I pull, you know, seven, I pull ten things out, and seven of them are black, and three are red, I say there's a 70% chance of being black. That value is the one that would've maximized the likelihood of pulling out seven of them. It's a maximization procedure. So I can choose the b sub k's that maximize the total observation, given those probabil, the idea which state I'm in, okay? Also, given that distribution, I can also say well, if I think there's a 75% chance that I'm in state one at this time, and a 90% chance that I'm in state two at the next time, that tells me that well, you know, the likelihood of going from one to two must be kind of high, right? And basically by looking over all of time and looking at my guess about the states, I can come up with the best possible transition probabilities. And when I say best, is when, again, the ones that maximize the probability. That gives me my new a i j's, and I've got new b k's. I can also guess new pi's, but those never matter. That's my new machine, right? So after I do the end step, called the maximization step, of getting the new A's and B's, I now have a new machine, I can go back and do the E step again. I can estimate the probabilities of being at a particular state of time, T, and I redo that process. And that iterative process, it's called expectation maximization. The particular algorithm for HMMs was called the Baum-Welch algorithm. Like I said, there's plenty of tutorials out there for you to look at it. The fundamental idea is that by using those recursive algorithms, and then estimating the probability of being at a particular state at a particular time, I can re-estimate the machine.

13 - HMMs General

Talk about HMMs just in general, and then we'll move onto vision. In general, HMMs are generative models of time series, with some notion of a hidden state, that is they can talk about the probability of generating a particular output. The forward-backward algorithms, those recursive ones allow you for computing over the states and you can also use that computing to train up your models. And I will say that HMMs were their big win was in originally in speech recognition dealing with low level auditory features and using that to recover phonemes and then phonemes into parts of words and things like that. And then in Computer Vision, although I will say there's a whole new area these days called conditional random fields, which is really better as I mentioned. If you've got a long sequence and you want to do segmentation, conditional random fields are a slightly more complicated version. Actually, they're significantly more complicated graphical models of representing time series. And it tends to do better than HMMs for the actual segmentation. But from the notion of a generative model of saying, was it an A, or a B, or a C HMMs are still used extensively.

14 - Some Thoughts About Gesture

So I want to look at HMMs in the context of what's referred to as Gesture Recognition. It's sort of an old approach, an old problem, especially using HMMs but it's, it's, it's a good paradigm. There's a conference called Face and Gesture, in, Computer Vision. So, obviously gesture must be important because there's a whole conference called Face and Gesture. And it's a long story going back to college of mine about why face and gesture are put together. I'll tell it to you some other time over drinks of some flavor. [COUGH] the typical scenario was you do some examples of each gesture. The system learns or is trained to have some sort of model for each. And like said lots of things used HMMs. And at run time one would compare the output of each of the different model, you would evaluate what you saw in terms of the likelihood sequence. And if that probability exceeded some threshold, you'd say, uh-huh, a particular gesture happened. So this was done, as I said awhile ago, sort of the mid-90s, then it started to taper off. Because partially this whole idea of having sort of individual trained gestures that people would do to, to make something happen, that didn't feel like just the recognition of individual vocabularies. So it's sort of tapered off. But then all of a sudden, new life was injected into the field of gesture recognition and it was because of human robotics interaction. Okay? People really love the idea of being able to gesture to a robot. Which if you've ever been to my lab we sometimes gesture to the robot in a not so friendly way. But this idea

that you could do things and control a robot, first of all, speech is very difficult in a noisy environment unless you're wearing a microphone. Gesture, you might be able to do something more remotely. And it was just this general idea that it was kind of cool, to be able to make some gestures and have a robot respond appropriately. So and here I just pulled out a paper, Reed 2014. This was at some conference that just happened, and since I'm recording this in 2014, it's a, you know, I'm not making this up. This is, you know, what's currently going on. I'm not advocating one way or the other the importance or whatever of this particular piece of work.

15 - Generic Gesture Recognition

I actually pulled out an older piece of work focusing just on the gesture. This is by Nam and Wohn of recognizing hand gestures using some HMMs. Just because it was very paradigmatic of how people would do things. Well, what's kind of cool about their system was, they had a bunch of different gestures that they wanted to indicate. Like nouns. You know, chairs, vase, lamp, bulb because of course that's a natural vocabulary. I have no idea. And then you want to do things like put it down or bring it or put or discard or jump. I don't, don't even know why that was a particular verb that they wanted to use. But the idea was that there was a fixed vocabulary. The inputs that they were going to use and they were going to control it with the hand, was a couple of different kinds. Right? They had something to do about hand configuration or posture, something to do with how your fingers were shaped. Okay. They also had how you're holding your palm and then they had a sequence of hand position, a motion, okay? And they wanted to re, use all of this information to make a decision about what gesture you were indicating. So the first thing you should ask is how are they getting this information? How are they getting this information? Boy, Meghan woke up. Well they were using something called a data glove. Here's a picture. So a data glove is this thing that you can wear on your hand that had both accelerometer and gyro information about how it was being globally manipulated and also knew something about how you were moving your fingers. So they built this kind of interesting system. And the reason I liked it is, they actually are using these three sensors the angles, the orientation, the position, right? And what's interesting is they use the HMMs just to get information about that motion, trajectory. Right? They had other methods of putting into their system how your orientation was changing and what the angles were. Right? So the idea is that you might have some basic overall probabilistic integration system, and in order to make use of that, you needed things that would output the probability or the likelihood of getting a particular movement for a given gesture, and HMMs were exactly the kind of thing to use for that. In fact, discriminative methods might have been harder to use for that because they need to combine these probabilities. So here was something very simple that they did. They took the, I shouldn't say very soon. Here's what they did, it required some engineering. You take your raw data, which was a trajectory that you would move in xyz, and the first thing they did was, they said, well you know, your gesture is pretty much planar. Right? The plane could be this way, could be this way, but the idea is that a single gesture didn't move in more than a plane. I might even go like this, but you'll notice even this figure 8 is in this plane. All right? So they find the plane that the gesture fit best to, projected down. And then they did what's called a chain code. We didn't actually talk about chain codes, if you did some machine vision, you would learn about chain code. It was a way of encoding a contour as being you know, maybe you can only take one of eight steps. Right? So you go up, you go you take a direction north. You go northeast. You go east, east, east., south, southeast, southwest and then eventually you come around. And you could think of the entire contour as a sequence of discrete steps. Right. So north, east, south, right? So sequence of discrete symbols, ah-hah. HMMs. So they trained a very simple left-to-right HMM, let-to-right meaning that you go through every state and you, you don't skip states you have these loopback things because you may stay in a state for a while where, essentially you are going kind of, lets suppose you are going north, northeast. But you didn't have that as a direction. Well, north northeast, that state you might sometimes put out north. Sometimes put out northeast, etc, whatever. But you'd stay in that state for awhile. It would output, there would be two symbols it could output with high probability. But that state for example would never output south. Okay, so that would be, you'd stay in the state for a while. So they trained up these HMMs and they did one HMM for sort of each gesture, okay?

They also did some stuff that had to do with what they call junctures doing transitions between gestures so that it, it could parse them. I don't think I'll, I'll talk about that here. I just will say that people hook up multiple HMM's together, to form sort of an HMM that can deal with a sequence of gestures, and it's, it's not too much more complicated than the original HMM. So they do that and then they test it. And because they're testing it on a very controlled vocabulary, using features defined to work, because they want their paper to be accepted, okay? You see results that look like this, okay? So here are their hits. This is the percentage of, hits are the percentage of the actual targets that were correctly labelled, misses are when you don't label a thing correctly. And as you can see, they're doing tests of over a 100 of each of these examples and almost all of their gestures are recognized with a relatively high accuracy. You can see there are one or two that was that were not so high and what and showing a table like this is one thing, a slightly more interesting table might be a confusion matrix. This is like we showed for the SVMs for the bag of words where you go back and take a look at the slide you'll see a table. Then it says how often is an airplane confused as a face, or that kind of thing. Normally if you're going to report results of that doing classification. Don't just provide hits and misses, also provide a confusion matrix because sometimes the actual source of the performance is that some of the categories were much more confusable than the others. You have no way of knowing that if you just report sort of performance rate overall, but you need to be able to see the the, the confusion. What, which, which element, or which categories were confused with which categories.

16 - Pluses and Minuses of HMMs in Gesture

That's all we're going to talk about in terms of HMMs in gesture. I'll say the good things about it were, it was a learning paradigm, right. So, you define some features, by the way you have to define the right features, you collect some well-annotated data, which also takes some work. But given that, you're able to train your HMMs by just giving it the data. So it's a learning paradigm and that's good. The training is a little bit slow, it's not very slow. But recognition can be very fast because of that recursive those recursive algorithms we talked about. Running those is very quick. So, so that works really well. You know, of course, not everything is perfect. They're not as good for segmentation labeling as some newer methods. I mentioned conditional random fields earlier. If you actually have something where your training data has particular underlying states and you know what those states are. So it's not even really a hidden marker model. It is a, It is a marker model, but your training data you know the underlying states. And you're seeing an observation and you want that state sequence for, for that segmentation, there are now better methods and I mentioned in conditioned random fields. You know, in general, this stuff requires a reasonable amount of data to train one of the reasons it was used for speech is we have lots and lots and lots of data. Although I will say that HMMs may require less data than some of these more discriminative methods. And then I put down something that I think is tonalogical, but also important to know. I said it works well when the problem is easy. We did a lot of work on HMMs, gesture, etc., whatever. And it's very powerful for capturing regularities, when the regularities are quite clear. You have to do a lot more work on your feature selection and a bunch of other things when those differences are a little bit harder to, to see number of states and those kind of things. So you know that, and that's probably true of most classification methods. When the problem is easy, they all work great. It's when the problem is not so easy that, that the challenge comes in. And that's when people start reporting on small differences. That means the real issue in def, in choosing your classifiers to understand where the complexity and where the difficulty of your problem is. But in general, people still use HMMs a lot, and they use it for describing activity or, or, or time series. And you know, it's, in some sense, your first line of defense against activity recognition.

17 - End

That ends our discussion on activity recognition, and you know, there's more and more of it available these days. The other thing is, if you wanted to learn more about HMMs and the graphical

models that they represent, like I said, that all comes from machine learning. So that's yet another class, because you know what? There's always more to know.

9A-L1 Color spaces

1 - Intro

Welcome back to Computer Vision. Megan's laughing already, because you could tell it's Friday. Today, we're going to start on just a couple of lectures that are, could be labeled other stuff you should know. It's just things that come along in Computer Vision that are, are relevant. Some of them are very old, some of them are new. Some of them seem trivial so we don't even teach about them normally. And yet if you don't know about them, you're totally mystified when somebody talks to you about them. And what we're going to talk about today is a little bit about color. Now, I had to wrestle a little bit with exactly how to do this because my own background color really came through psychophysics psychologies. My, my degree was in brain and cognitive sciences. So and I think of color as a psychophysical or perceptual phenomena. That is when you see this thing and you say, oh, that's a light blue shirt. That's actually not a physics statement, it's a psychophysics statement. So I'm actually going to start there. We're going to talk about just a little bit about, actually an, anatomy. And then we'll move on to a little bit of computational elements having to do with color. So and also in a few more lessons, you're going to see some, just a few of these slides and discussions again, because we're going to talk about the humans vision system. But I need to just talk about a little bit of it now, and in particular, I have to talk about your retina. Well, and my retina, and everybody's retina.

2 - Cones

What we're showing here is a picture of high magnification, probably a electron micrograph, then pseudo colored, of your retina. And you can see that retina is made up of two kinds of receptors called rods and cones, and we'll talk more about the differences between them when we do the human anatomy conversation. But today, all we're going to talk about is cones. And in particular, there are several million cones in the retina. They tend to be focused more or grouped more in the, near the fovea, the middle. They're responsible for the high-resolution imagery. That is your ability to see very sort of with a good resolution at the center of your visual field. And most importantly, for today's conversation, it's your cones that discriminate color. That is that they're sensitive to different wavelengths, and your, your visual system uses the outputs of the cones, processed through a variety of channels to see color. And particularly there are three types of cones that are called red, green, and blue. That's not really quite correct, you'll see why in a minute. They really should be called long, medium, and short. And you'll also see it's written here that the vast majority of them are red and green and a very small percentage of them are blue. All right. So, this is a graph of the sensitivity of each of these three types of cones, roughly red, green and blue, in terms of how responsive they are as a function of a single band of wavelength of light. So here you see the wavelength of the light expressed in nanometers, and you can see that there's one channel right here that starts at the longer range and peaks somewhere around there. And then there's another one that's called green even though it's really just offset a little bit from that. And then finally, what's called the blue, which is really the short wavelength cone. If you were to make a picture of sort of the retinal mosaic and you colored the cones, what you'd see would be a picture that looks kind of like this. You'll notice that in the middle, there's hardly any blue cones at all and there's some more of them out here, all right. But basically almost all of the cones that are used are in the red and green. And the the blue is really just there for a little bit of coverage. And yet we see color very, very well.

3 - Tristimulus Color Theory

In fact the this, this notion that there are three receptors that are absorbing light, refer, results in what's referred to as the tristimulus color theory. The idea is that it's three inputs. And you know, here it says the spectral-response of each of the three types of cones. Unlike the previous one where we're showing you the relative sensitivity of each of them, what this graph is showing you, just as it says, this is the percentage of light absorbed by each cone, actually by each cone type. The reason that even though blue is so much more sensitive down here, because there's so few blue cones. The amount of light down here absorbed by the blue cones compare to the amount absorbed by red cones and green cones is approximately equal because there's so many more green cones and red cones than there are blue cones. So the question then is, is can we, how, it says can, but how would we use the outputs of, of these three types of receptors in order to see color, in particular like spectral color. And what I mean by spectral color is, you're all familiar with this red, orange, yellow, green, blue, indigo, violet. You probably learned that somewhere along the way as the colors in the rainbow. Well what it really is, is these are different wavelengths of light. And those are the colors that you see when those wavelengths are put out, okay? And I'll, I'll, I'll, we'll, we'll see a little bit more about what we mean by that, all right?

4 - Color Matching Function Based on RGB

In order to think about starting to understand color in a standard way, experiments were done and they started quite a long time ago, where you'd do the following, right? You would, you would make some light coming into the observer, and that would be some light along that spectrum. So, the red, the green, the blue, etc., you'd have some pure spectral light. All right. And the observers job and here the observers one great big eyeball which is kind of creepy, but there's actually a person sitting there, would be to just adjust the amount of red, green and blue light. And they were given a particular type of red, green, and blue, particular wavelength of light, and they could put more or less of it, of each of them to try to get it to match the test light that was being produced. And most of the spectral colors could be done this way. You could just add a certain amount of, of that light. But, it turned out that some of the colors couldn't be matched, unless you actually added, a little bit of red to the original test light, okay. That is, in some sense, in the mixture, you had to sort of like have negative red light and of course we couldn't make negative red light. So what you would do is you would sort of add some positive red light to the testing. So that's the first thing to realize is that certain color, certain wavelengths in the spectrum. You would look at it and, you know, for those particular three lights they gave you, there was no blending of them at all that would equal this color. And the way that gets expressed is the following, right? So here you were giving a certain type of blue, green, and red light and then depending upon where you are in the wavelength, where you are in the spectrum, depending upon what single light I put out, all right? You would have, this is how much red, green, and blue you would need to add, and you'll notice this area here is the negative. These were the colors, okay? These were the spectra that for those particular red, green, and blue lights that they gave you, you couldn't make them match. So they represent that as being sort of negative red.

5 - Color is a Psychological Phenomenon

So color is a psychological phenomenon, all right. You know what In fact let me go back to, to something here. If I turn on a red light and a green light and I blend them together what color will I see. Yellow. Yellow, right. So why? When I tell people this at Georgia Tech, some of them tell me, oh well, you know, red is here in the spectrum. And green is here in the spectrum. And when you turn on both of them you get the average spectrum. No. [LAUGH] The light doesn't know anything about averaging. So here's a green light, and here is a red light. What happens is this light over here in the green, triggers the green cones a little bit more than the red cones. And the red light makes the red cones go a little bit more than the green cones. Kay, so let's pretend for a moment they were sort of balanced. So in other words, if I had a yellow light, which makes both of the

cones go about the same amount. Your system can't tell the difference between a green light and a red light turned on versus a yellow light. Forget this little blue stuff here for now, all right? So the reason that red plus green make yellow with lights has nothing to do with physics and everything to do with how your retina is put together. Right? The red and green cones go on at the same time. So, color is psychological. That is you can think of color as representative. Some combination, linear combination otherwise of red, green and blue and it's related to cones not to physics. By the way almost all of us have the same cones, but there are some people who don't. Right, so the sky might not really look blue to them although actually to them it looks blue, but it's actually not blue because they actually can't see blue. So these people are referred to as color blind, all right, a bunch of people are missing one type of cone. Almost all of them, by the way, are men. It, it's, it's some reasonable percentage of men are colorblind. Some tiny, tiny percentage of women are colorblind. So it was kind of funny back when I was a grad student, psychology class, people have to sometimes do a, so I was the TA, people would have to do an experiment and write it up and hand it in. And you can tell, every now and then, when somebody, shall we say, ran out of time to actually do a real experiment. because they'd say they worked with 20 color-blind patients, ten men, ten women. For them to find ten color-blind women, they probably would have had to sample more than Massachusetts. So there are very few color-blind women.

6 - Luminance vs Color

So, now the second thing to know about humans, all right? Human vision. Humans are much more sensitive to changes in what's referred to as luminance, how bright something is than they are sensitive to what color it is. Now here's an example. Now this example may work better or worse depending upon how your monitor is set. So if you're not seeing the illusion I'm about to show you, play with your monitor. Here's two images with text on them, and in the top, the background and the text are now, now I'm going to have to change the color of the pens, they're the same color, but they're the different brightness. Okay, different, the technical term is luminance. All right? And probably all of you have no problem seeing that it says text in color without intensity differences, blah, blah, blah, fine. On the bottom, actually on the Mac monitor, I can hardly read it. On this monitor, I can read it just fine because it depends upon how each monitor, and we're going to talk about monitors and gamuts in a little bit, how they show it. But you could have adjusted the color of the text in there so that the color became almost, what's called isoilluminant. All right? So the same amount of luminance, okay, but different color so you just basically take the, the cyan there and just make it brighter or darker. Don't change its, what we're going to learn is called the hue, the color, and you can make it go away, right? So the bottom line is that the human system is much better at dealing with differences in luminance. So these phenomena, trike, tri-stimulus color theory, difference in luminescence, this has been luminescence, has been known for a very long time, since the very early 1900s. So starting in the 1920s, people started doing some what are called psycho-physics experiments by psycho-physicists, because who else would do psycho-physics besides psycho-physicists? Of basically showing people different kinds of stimuli and asking them what they see.

7 - CIE RGB Color Space

So some of the first sort of systematic experiments were done by these folks, Wright and Guild in the 20s. Although the experiments people know about are a little bit later in the 30s. They were trying to map this notion of wavelength to perceived color, right. So this is what I was telling you about before, where as I change the wavelength. How much do you have to change the RGB in order for you to say that they match? And they were allowed you to be able to say two colors are more similar to each other through this process, and they define what is called the CIE color space. CIE actually stands for commission on illumination or something in French, and that's why its CIE. So the diagram they came up with here this is called their xy chromaticity diagram. Couple things to notice and we'll talk about what x and y are in a minute. These are the colors at, as the wavelengths go around, right? So you're getting longer and longer wavelengths, right. And you get

back to here. So the numbers around the, these blue numbers around the border, those are the wavelengths. And the middle is what you get by blending different colors. Now what do we mean by blending? Well, what they figured out when they were looking at people computing playing with these RGB lights, and other things was they came up with a new color space that wasn't just RGB, but that it was going to be a linear transform of RGB, but that the prominent value was going to be called Y. What does Y stand for? It stands for luminance. Don't ask me why it is? Because it's actually from the XYZ space, and the idea is that Y is the luminance. And so then X and Z were two other quantities with which you be, bet, between X, Z, and Y you could figure out the color. And you could represent a wide range of colors, and in fact in this diagram that I was showing you before. You notice down here it says x and y well, it was just this x normalized by the sum and y normalized by the sum. And so in some sense it's for sort of a fixed luminance, it's taking the luminance out of it. And you can see as you move into the middle oh, you can't see that at all, because it's red drawing on red. As you come to the middle you're blending all these and that's going to be white and we'll talk, or gray depending on the luminance and we'll talk about that in the minute.

8 - CIE Lab Color Space

There is something called the CIE Lab space. Sometime called Lab space but L a b actually, where L is luminance, and then, as you change the a and the b, you end up with different colors. I'm going to use the work color now, instead of the more technical words later which, which we'll use. So here, you have a luminance of 25%, so it's kind of darker. And you see the greenish, reddish and blueish things as you move around. And then the middle is kind of gray, right? And as this, as luminance gets brighter, the middle gets brighter, and you get these lighter variations in color. And so that's called the Lab color space. And the idea is that you've got luminance, and you've got sort of how saturated the color is. And actually, what the, what we call the color, but what is ty, typically referred to as the hue. And that sometimes defined you can think of the color as a three-dimensional volume of one sort or another, and you'll see there's a whole bunch of ways of representing it. But the idea is that one way of thinking about it is that down here is dark, up at the top is brighter. And as you move around this circle, so it goes through 360 and around again, is what's referred to as the hue, okay? And as you move outside, things become purer in color, all right? So two systems that people use a lot to talk about these things are called sometimes hue saturation and lightness, or sometimes hue saturation and value. There are all different kinds of color spaces. These two can be drawn as sort of these thick cylinders. So down here again at the bottom, things are dark. Up at the top, things are light. In hue saturation and lightness, the whole top is white, which I think by the way is an awful representation, I'll tell you why in a second. Down the whole thing in the bottom is all black. And as you go around, you'd get different colors. That's what you can see around here, different hues. And as you go from the middle to the edge, you get more and more saturated. So in the middle, everything is gray. It goes from black to white through gray

9 - Hue Difference Quiz

So if you say that hue goes around between 0 and 360, then the question is what's the difference in hue between two, two of the following pairs of hues. 225 and 75. And 45 and 315. So what falls in between those?

10 - Hue Difference Solution

So what's the difference between them? Well you take a look and you say, you know, to get from 75 to 225 that's a difference of 150 and yep, there it is, it's 150 okay? But then you take a look between 45 and 315 and you say what's the difference you say, oh my gosh, to go from 45 to 315 requires a, a, a hundred and oh, that's a lot. So what's the difference? What's, what's 315 minus 45? [SOUND] 270. It says there's 270 between them, that's a really big difference. Well, of course, the

answer is no. The difference between them is actually 90. And the reason is that, if we go from let's say 0 all the way around to 360 again, 45 is over here 315 is over here, right? So the answer is it's not 270 between them, it's only 90 between them. So it's just a way of remembering that things have to wrap around.

11 - Other Color Spaces

As I said, there are lots of different color spaces. Here's a nice picture of all these different ways of representing these things. By the way, notice the one here at the bottom left, red, blue, green, red, green, blue. We'll talk about that one in a minute. All right, but let me show you what my favorite. In fact I'll put up two of them. So which one's my favorite? Well, I think I already gave it away. I told you that the problem with this one here, is that the entire top surface is white. And so, here's the question. How much can I change the color of something that's white? Answer, not at all. White is white is white. So that's why I like this idea of having this peak on the top, so that is you can't get far away from it. Likewise, same as, as on the black. And you spin around, the double cone gives you the, the hue. And as you get further out, they call it chroma. I actually like calling it saturation better. It, it doesn't really matter too much. But the idea is, I tend to think of it as a cone. And what's important is when you realize that it's a cone, you realize that if you're comparing, and we'll get to this in a minute, if you're comparing the hue of two different patches, if they're very dark, the hue doesn't mean anything. If they're very bright, the hue doesn't mean anything. And that's very hard to see on this cylinder representation, but it's very easy to see, see on the, on the cone. Now of course, what is the color space that we all know and love, or at least know? All right? Which is the one we know best? Well, it's the RGB space, right, where basically, you've got a cube, where in sort of a back corner where things are 0, 0, or everything's black. If you move down the red axis, it turns red. If you move to the green axis, it turns green. Add them together, what's red and green make? Yellow. Why? Cones. And if you bring the whole thing up, it looks white. And what's cool is if you take that cube, and you sort of pull the white corner to the top and the black corner to the bottom, right? So in fact I have a sort of picture, just think about that, it would be almost like a double cone. Not quite, it would be kind of a lumpy double cone with sharp points. But the idea is, white would be at a point on the top, black would be at a point on the bottom, right, and then you would have chroma as you went around. And that's why my preference is the double cone.

12 - Color Gamuts

One more thing before we get back to computational computer vision. When we talk about the color space of devices actually when we talk about the color space of devices, what we're usually talking about is what's referred to as the color gamut, okay? And color gamuts are just subsets of possible colors. And here I'm showing you three different color gamuts that have been measured or that are just defined in a particular way. So you see this first one? Alright, this is the 1931, that's that Chromaticity Diagram I was telling you about, and the, and this big that's all the colors you can see. Notice that on the inside, is this triangle that is a tremendous subset of that possible set and you see it says HDTV. Turns out these are all the colors that HDTV can make. So if you wanted to make this really nice lush deep green, HDTV can't do it because of the way it defines its color space. All right? Any photographers out there? I hope so. If you use Photoshop or any other photographic manipulations systems that understand color spaces, they'll tell you that you can have it in a color space of sRGB or Adobe RGB or ProPhoto RGB. These are different sized color spaces, and you know one of them is really small it's this sRGB. Okay, why do people use that one so much? That's the color space that most color monitors used to be able to support. Okay. And so, if you were displaying something, say on the web, you don't want to have these colors out here because they're not going to show up the right way on a monitor. So when you try to do color balancing sometimes you have to map colors that are distributed out throughout all of color space onto a smaller color space. That's particularly true when you print things. Here is a, some gamut comparisons of so again the, the big one is all of human vision, the white one is this Adobe RGB we were talking about. Notice the green one. The green one it says Epson 2200 Premium Luster. That's the color

gamut of one particular printer on one particular paper, okay. That's the set of colors that that printer can make. So if you have a color out here it won't work very well. Likewise down there. I learned this the hard way when I first got into digital photography. My daughter at the time was about 11 years old was wearing this really kind of dark blue shiny, royal blue dress and I printed it on a HP printer, not a great printer but not a terrible printer and it was purple. And what was annoying to me to begin with is the color of a bunch of other things were perfect. So why, you know, because I understood color balance, you know, maybe I had a wrong color, no, you know, my, my son's gray thing was nice and gray. It didn't, it didn't shift blue or red. But her pur, her blue thing was looking purple. The reason was is that is was outside the color gamut of the printer. All right? And I've subsequently paid a lot of money to get a much better printer

13 - Revisiting Pixels

Let's go back and talk about pixels. By the way everybody know what pixels stands for? Pixels stands for picture element. If you talked to a television engineer thirty years ago, forty years ago, he talked about pels. Computer people call them pixels. Television people call them pels. Little bit of information. So we know that we have pixels at every sort of location, x y in image it has some we'll just use the word color for now. So let's zoom in on a little, area, here and we take a look at some particular pixel, and we can say, that pixel's at X location, here it says 243, Y location of 84, and what's the value in that pixel? Well the value actually was three of them, right? And here they are. A red, a 150, a 173, and 81 written as R G and B values. Those are the values, the intensities of each of the three channels. Red, green and blue. All right? So, the color, here, are vectors. Remember, we talked about them as being a vector image? Right? So, at every location, we get a different vector. By the way, you see, over here, on this white pixel, it says 254, 255, 251. This is obviously an eight bit image that goes from zero to 255. If this was a floating point number that we let go from zero to one or floating point image, then that would be you know 0.98, 0.999 or 1.0 and 0.97 or something like. All right. So those three values together, give you the perceived color. When you're displaying an image, what you're really doing is you're plotting a different locations on the screen. A vector by essentially turning on a red light or green light and a blue light. You're essentially doing a plot of those colors, but when you look at it, it looks like a, picture, alright? What would happen if instead of plotting them in x y, you plotted them in RGB space? Okay? So you're going to make them be the same color as before, but you're going to put them down in a, in a three dimensional space not according to x, y, but according to what their RGB value is. Well, so here I have that same little picture, and I pull out the first, pixel. And you can see here that red is 152, green 50, blue 40. So that's what down here and this is in this three dimensional space. All right? All right? Here's another point closer to white, so it's all the way up, in the corner, and finally, here's a, third point, all right? So these are plotting these pixels in the RGB Space. So we go from Image Space to Color Space. So, let's take those fruit pixels, and plot them all in RGB Space. What would that look like? Well, it would look like this, and this is sort of, kind of an elegant, three dimensional, plot of that. And what you can notice right away is that the pixels are sort of grouped, right? These kind of reddish pixels presumably have something to do with the apples. You know, what are these whitish pixels, we'll they're going to be the background. And here are these yellowish pixels, probably have to do with the oranges. And you start to think about, you know, we might be able to use your location in color space, to try to segment out the different parts of the image.

14 - Plotting Pixels in a Color Space Quiz

So here's another quiz. What does that view, I'll go back to that view, that color space view, let us do. Okay, and here, so one is, think about clusters of pixels that are similar in color. Understand the shape and size of objects present. Identify pixels that are different, and separate them. Count how many pixels of each color there are, all right. So what do you think it is?

15 - Plotting Pixels in a Color Space Solution

Let's just do a few of these things, all right. Clearly, this lets us think about clusters of pixels that are similar in color because we get clusters of pixels that are similar in color. Yeah, but why is that useful? Well, one might believe that pixels that are similar in color are ones that are somehow supposed be treated as a unit and pull them apart. It certainly doesn't let us understand the shape of objects. Size, yeah, not so much because you can think of sort of the number of pixels but that would only work if an object was constant color. Identify pixels that are different, well maybe kind of etc. and separate them so we'll give that a partial credit. Count how many pixels of each color there are, well, what do you mean each color? Color's not like, you know, are these all red? I don't know, it's been, remember, the idea is that color's going to be a categorical thing that's operating in psychology, psychological space, right? All this lets us do is think about clusters where it says similar in color, so we're going to have some notion of a distance.

16 - Red Filter Example

Suppose our goal was to pull out the reddish pixels, so here's one example. All right? So let's suppose we filter this, and we say, we're going to pull out all the pixels where the red value can be anything at all, the green has to be low and the, the blue has to be low. What do you get? All right? Well, you'll see that you get bunch of stuff where like, look. Here's a green value, that's dark green, which is perfectly reasonable because it might not have very much red at all, but it has got, you know, a certain amount of green. Suppose well, I want it to be even more red. Okay? So you might say well, let me require that the green and the blue be even more, me be lower. You know, you start to see that you've lost, first of all you've got these edges here, these pixels on the thing, and you've lost some of the red pixels that you wanted in the, in the apple. And basically, it's kind of hard if you're building, these are essentially box filters, right there. There's certain range in red, certain in green, certain range in blue. And it would be hard to build a box filter that pulled out all your red, your red apples. And the reason for this, and we've seen this before, is that, here we're plotting three different shades of the, three different intensities of sort of the same green. In the RGB space, right, so in, in a nice sort of green in the middle, we get this nice separation between these values. But as things get darker, they get closer, and as they get brighter, they get closer. Why is it as they get darker they get closer and if they get brighter they get closer? Remember that cube, right? So as I get closer to white, right? The I, I have no choice, but to sort of get closer in RGB space, and as I get closer to black I have no choice, but to get closer. So to try to do this separation in RGB space is kind of difficult.

17 - Separate Intensity and Color

So how might we think about creating some channels that make it easier to separate these things? Well we already saw this from our previous conversation. Let's separate out the intensity from the color, okay? So the first thing we're going to do, it we're going to define intensity Y as some combination of R, G, and B, okay? And so that gives you this nice picture here. And those of you that are old enough to remember black and white television, you start to be thinking, oh, wait a minute, this is like an image that just reflects the overall intensity of what's there. And that would be correct, all right? And in fact, a luminance function, which is what this called, is just a linear weight of the red, green and blue pixels. Here are weights of red, green and blue. And by combining them together, I get a single value Y, for luminance. Remember, Y stands for luminance for reasons that are, because it's the middle of X, Y, Z. That's why. All right, so now I need the two other values, right? I mean I could just keep the R, G and then try to reconstruct them. But instead what they do is they compute a different color space. Then this is called the YUV colorspace. And here's a, a computation that takes the B values, the Y values, and creates a new value called U and V. And a way of thinking about this is one of them has to do with sort of the difference between the luminance in blue. And the other one is the difference between luminance in red and it matches and it creates this thing called UV. And the thing that you need, that you need to know is that U and V

contain the color information. So they tell you that they're giving you an image in YUV space, you know that UV contain the, the chromatic the color information. And by the way in this YUV mapping, there's a maximum and minimum that U and V can get maxed to. And that's why it's called Umax and Vmax. And we assume the R, G, B range from 0 to 1. Okay, if it goes from zero to 255 you just scale the whole thing. All right, when you create that you can actually make three images, right? You can make a Y image, a U image and a V image and that would look like this, all right? So the Y image is ju, exactly what you're used to seeing in your black and white television. But your U image and your V image, those contain the color elements. Now you might ask, well why aren't they in color? Why aren't they in color? Because each one of them is a single channel, okay? Right, so that's the U. So the brighter it is, the higher U, the lower it is, the, the less U. Same thing with V.

18 - YUV Color Distribution

We now know how to get from RGB to YUV, so you can actually do some cool things with that, all right? Let's plot the pixels that we're looking at before instead of RGB space into YUV space, okay? Well here, you see a YUV distribution, all right? And looking at it just like this, it's, it's not clear yet that we've bought a lot of improvement. But supposed we ignored the y value, right? We just projected everything down onto the UV plane. What would this distribution look like? Well, it would look like this, okay? And suddenly, it's a lot easier to see how you would separate out the different pixels of different colors. In fact, we can try filtering on just this UV dimension

19 - UV Filter Quiz

So, let's suppose we wanted to pull out the red regions of the image and leave in, remember we've projected down all of the luminance so Y can be anywhere from 0 to 255. What red values, what UV values would we look for?

20 - UV Filter Solution

Well, let's see, I don't know, we probably want, you know, something that's like here, and maybe something that's kind of like that. So I did like, for U from 130 to 200, and for V, I said about 100 to about 130.

21 - UV and YUV Filters

We can just filter the image, right? So we filter the image, tada. Yuck. All right. You know, the first thing that happens is you realize you've got all of these white pixels. Why is that? Well, because they didn't have a huge amount of V but they, they still had, a reasonable amount of that Y, all right? So how would filter out these bright pixels? Well duh, you would just reduce the requirement on the luminance, we're not going to allow all of the Y pixels anymore. Instead, we'll filter, we might say, well, we're going to allow the luminance of just, I don't know, 0 to 150, let's say. And so that would eliminate, so here we are in the same, right? That, so that would capture all of these pixels. All right? So now if we use that, right? We in, we include the Y value of 0 to 150, we get a much better selection of the red pixels. And this is just to show you comparison that it's just easier to do this on the, in the YUV space than the RGB space. The only reason this is easier is because we're using boxes. Right. Because there's a linear combination of RGB that will give you almost prelim, it'll give you YUV. If you could orient your boxes they would end being the same filter, but if your trying to limit the boxes they would be straightforward.

22 - Intuition and Other Luma Chroma Color Spaces

So why would you use YUV? Well, okay, it's easier to cluster pixels, but, you know what? That's, that's not that important, because I could just rotate those filter boxes. There's a much deeper reason, and that actually comes from the part that I showed you earlier. You remember that human vision is much more sensitive to chr, to changes in luminance, than changes in color? What that means is that you could actually subsample the color, have fewer pixels if you will, than luminance. And the, for the human system, it would be about the same amount of resolution, okay? So in order to compress your imagery, in order to use fewer bits to describe something, one of the things you can do is, you can use more bits for the wide channel than you do for the U channel and the V channel. There's an example, it's called YUV422. I think the original 422 actually had to do with the amount of bandwidth of the luminance versus the two chroma channels. These days when you think of 422, the idea is that if you were encoding two pixels, you would use two bytes of it and assume that you were just doing pixels 0 to, to 255. You would use two pixels for the two bytes for the luminance, but one byte for U and one byte for V. So you had twice as many bytes dedicated to the luminance signal than you do the chroma signal. And it turns out that humans, it's pretty indistinguishable. Something else that you can do, there are other chroma spaces, some of them come from video and television. There's one called YCbCr, change difference between luminance in blue, luminance in red. There's something called Pb and Pr. They're, there are all these sort of different linear combinations. There's the L*a*b* color space that was based upon the human measurements that we talked about before. There's something called L*u*v*, which is just a little bit easier to compute than L*a*b*. The bottom line is that all of these are different ways of representing color. If I were to plot this now in say, HSV space, right, and I just showed you H and S, right, hue and saturation. Now, it's pretty easy to find the red pixels, right? Here they are, there's some over here some over there. All right, and the, the hue gives you the color spectrum. Why is this coming around back over here? It's because hue is actually going around in a circle and it crosses zero there, so these two are actually very close together. One last thing just if you ever want to play around. If you ever have a digital video camera and you bring that video into your computer and it's kind of like, low light, you'll see it does a reasonable job. If you can pull out the R, G, and B signals and play them as a separate video, the R one will look okay, the green one okay. The blue one will have all of this noise in it. You play it and it's awful, and that's because they dedicate hardly any bits at all to encoding the blue signal, because it doesn't matter to you, all right? And again, this isn't physics, this is psychophysics, but it's how computer vision and computing of color relates to color.

23 - End

So that's just a little exposure to color spaces. I'm sorry if it was sort of obvious, but, but if you've never thought of color spaces, then this notion of where does RGB come from? What is a luminance signal? How might I to do background subtraction skin detection, etc in color? You sometimes have to manipulate these kinds of things. So, now you know.

9A-L2 Segmentation

1 - Intro

All right. Welcome back to Computer Vision. If I'm a little less energetic than usual I just flew back from Taiwan and boy are my arms tired. That's like a really, really, really, really old joke but I really did just fly back from Taiwan. After spending just four days there and back. So if I start talking, if I start talking quietly, Megan'll remind me to wake up a little. All right, today and for the next little bit, we're going to be talking about segmentation. And it's going to follow our color conversation because it's important to think about spaces in which you're describing your pixels and doing that and the fact that you guys now know about color spaces will make this easier.

2 - Segmentation

So why are we interested in segmentation? And what do we mean by segmentation? Well by segmentation obviously, we're going to segment the image into a variety of regions. And the regions that you want to create depend a little bit on what you're actually trying to do. So, there's this general intuition, and here we point to this work up by the folks out at Berkeley and Jitendra Malik, and his group. They really pushed hard on this. There's this general view that if I show you a picture, like this picture here on the left here, which has these, bison. That's what we call them in the States, or buffaloes, and we still have them wandering around, not in Atlanta, but in general. The idea is that I should be able to, you know, if I handed that picture off to a, to anybody and I said, draw a border around the different objects, you would get something that looks kind of like that, all right? People would segment and, and the, this segmentation wouldn't all agree, right? Some people might separate out this little guy here from the grass but you can see that this one doesn't. But in general, there'd be a good agreement on where the different elements are. And this database that's pointed to, they recorded a whole bunch of segmentations done by people, and that can be used to train systems. Here's just another example taken from a typical American scene here. In fact, it even says that it's in Idaho. And you can see that the, the regions that were cut out seemed to correspond intuitively, what you might think of as the different object, or something more along the lines of different parts. Another reason you might want to do this, is you've got some objects, that are sort of the ones you want to pay attention to against the background. And you want to focus on the object, and not the background. So, here's some examples referred to as figure ground segmentation, which is the pulling out of the, the figure from the background. And these typically are done in things that are moving around. Here we've got some animals playing. Here's a strange creature referred to as a young child. And the idea is that you'd like to be able to pull those objects out. So if you want to analyze, like we were doing activity recognition, you want to describe just the movement of the object. Another way of thinking about segmentation, again some work that was introduced by Malex Group is this notion of super pixels. So these two pictures that you're looking at here, you can see that everything is sort of cut up into these little fragments. And the idea of these fragments, especially take a look at along around here, right, is that within those fragments, everything is sort of the same, and it's within the same region. And what that means is, when I think about further cutting up this picture, or processing it, I don't have to think beyond these little regions, which we'll refer to as, well later in the literature, as superpixels. Right, the idea is that I go from having tens of thousands, or hundreds of thousands of pixels, to maybe having hundreds of superpixels. So that makes the combinatorix much, easier. And again, getting these superpixels is a form of segmentation. Segmentation extends beyond just single images of course. Here we're taking a look at video. This is actually some work that comes from here at Georgia Tech done by both Professor Erfanisa, Jim Raise and Professor Jim Raise group here. And there's this ice skater going along. And the ice skater is being segmented and it's not always so easy because here, it's showing you against just ice but obviously it pulls it out. And you want to be able to extract it carefully.

3 - Noisy Image

What we're going to do is, we're going to talk about segmentation. I want to motivate this using a toy problem, a very simple problem, and then we'll move to slightly more sophisticated examples. So here's our toy example problem. So we have an image on the left, and you can see that there not that many gray levels present and in fact there's a histogram of them on the right. And on the histogram, you can see that there's this big collection of black pixels, another collection of gray pixels, even a little bit bigger and what's this thing on the right? That's this huge collection of white pixels. And they're exact. They're black, gray and white which leads to thinking that okay in this image there are three regions. And I would label them 1 for the black. 2 for the gray and 3 for the white. So the idea is that we went from the groups in this histogram to these segments in the picture. But of course that's on this trivial picture. Here we have that, that clean image. What if you have a noisier image, right? So here I've just added a little bit of noise. And now our histogram is

not just these straight lines anymore. But we get these distributions of intensity. So the question of how do we break things up. And the question is you know, how do we carve this histogram into these different groups? In particular, we refer to this as clustering. So there's a fundamental relationship between segmentation which is carving an image up and this notion of clustering which pixels belong together.

4 - Clustering

Here's a simple way of thinking about it. So our goal, let's suppose we're just worrying about the intensity of a pixel where its intensity is just single value. Right? So it's a grayscale image, it goes from 0 to 1, 0 to 255, minus 27 to plus 18. Whatever your range happens to be, the idea is that you've got this linear that, dimension along which the values are distributed. In this case we, we're going. So there's our distribution of our pixels, right? So there's a little circle down here let's say for every pixel that's in the image. All right? And the question is, how do we find these centers, these clusters that we say okay, those are our three cluster centers. And once we found them, we could say all right, I've got these three different areas of the image, one, two, and three again. What do we mean by good cluster? Well, generically here we're going to talk about that our best clusters are clusters whose centers minimize the SSD, the sum of squared differences, between, the, all their points and their centers. Okay, so the idea here, here is our SSD and it says summing over all the clusters, given a point within the cluster these. Actually we'll make that square just so it's clear, the distance between the point and the cluster center. All right? And the challenge in finding these clusters is that it's a little bit of a chicken and the egg problem. All right? So here's, right? Suppose, that's what Q. Suppose we knew where the cluster centers were, right? So they're here, here, and here. Well if we knew those where the cluster centers were, which points would we assign to each cluster? Well that would be pretty easy. Just as it says in the answer. We would assign the points to the cluster depending upon which cluster center they were closest to. So given the center, it's obvious to assign the points. On the other hand, what if we, instead of knowing the cluster centers, all we knew was the cluster memberships? That's what these ovals are signif, showing, right. We know which points belong to which clusters. Right, this is the chicken problem or the egg prob, I dont know, it's the other problem. The question is, if i knew which clusters the points belonged to, where would i put the centers? Well, i would put the centers to be the mean, the average of each of the points of their cluster. So everybody know why? Do you know why if I gave you a bunch of scores that represented how people were doing on a test and I asked you sort of, you know, what's, what, you know, how, how well do you think people do on this test you would take the average? You know why you take the average? Turns out the average is the value that minimizes the sum of squared differences. It's trivial to show. And the reason you do that is you believe somewhere in your head that the actual thing you're looking at is a distribution that's centered about a point with a Gaussian fall off around it, right? Some, some value whose probability falls off as the square of the distance. So the value that would generate the maximum likelihood of the data is the mean value, cause it minimized the sum of squared differences. I thought every masters student in computer science would have known that. And I just gave this lecture, just a little while ago, in class here, and it turned out, well there were undergraduates in there, too, so maybe they were the ones who didn't know that, but not everybody knew that. So, The reason we take the averages of things, the mean values, is it minimizes the sum of squared distance. And if you have a Gaussian model or any other distribution, where the likelihood of probability falls off as a square, then if you minimize the sum of the squares you generate the, the value for which the data you saw would have the maximum likelihood. Just tuck that away so you'll remember it somewhere.

5 - K-means Clustering

So we have this chicken and the egg problem and those of you, I'm assuming, most of you are familiar, this is actually referred to as k-means clustering. It solves the chicken and the egg problem by smashing them together, which could be a pretty nasty picture, but essentially, it's just an iterative algorithm. What you do is, just as it is says here. And we'll take an illustration in a minute,

you randomly, yes just randomly, pick some cluster centers. You might try to be a little smart about how you do that or, or not. I find it easier to be not smart. And then, once you have the centers, we know which points should be assigned to those clusters. Once we've assigned to the clusters, that's step two, given the points in clusters, step three, we can re-assign the center, and you set that, you set the new cluster center to be that. And step four is, well if any of the cluster centers move, and the only reason they would have moved is because you changed the assignments, you go back to step two. And this is classical k-means and I assume most of you have seen that. But, just showing a few in the contest of, context of segmentation here's a nice little example from Andrew Moore, recently appointed the Dean at Carnegie Melon. Congratulations Andrew. So here we have k-means, right? So we've got a bunch of points. Now we know where we think the cluster should be, but that's because we're smart. A computer's not so smart. We picked some random centers, so there they are. Not a particularly good choice of initial collection. We then partitioned the space up, right? So those of you from computer science recognize this diagram. This is the, the fracturing of the, of the space such that this line says that everything here is closest to that point, more than any other point. And so once we have those assignments, what we can do is, we move the centers to be at the center of the points that were assigned to that cluster, and then we iterate. All right? And we repeat that until it terminates.

6 - Number of Clusters

To apply this to the segmentation problem of vision, we have to think a little bit about what feature space we do this in, all right? And it turns out for an awful lot of what we do in clustering and segmentation, and frankly just about everything in computer vision, the feature space you use is what matters. So here, I'm showing you that single line, with the idea being that we're just a grouping along a single, intensity space. So if I were to cluster, using just this simple intensity space what would it do to this panda picture, and Professor Grauman in Texas, she's got these, this thing for pandas. Actually I think she got these pictures somewhere. Well, so let's see. So here's our panda. If I tell it that there's going to be two centers, okay, then I would get a division that looks like this, right? So you can see that basically, the lighter pixels have been turned into one region, and the darker pixels have been turned into another region, all right? Now, if I told it instead there were three clusters, you might look at something that looks like this, all right? And here you can see there's the bright, the dark, and the grey, all right? kind of like the good, the bad and the ugly. So what you're seeing here right away, is one of the problems in doing K means clustering. We'll talk more about this, is that you have to tell it how many clusters there are, and there are methods for then trying to figure that out beyond that, but fundamental to K means is you already know the number of clusters. When you do this type of clustering, you can think of this as quantizing the feature space, right? So those of you familiar with vector quantization, if you do it in multi-dimensional space, the idea is you essentially carve a continuous space up into a set of buckets. So this segment, this clustering is a quantization of our feature space.

7 - Segmentation as Clustering

Even pandas aren't really black and white, they're actually colored. Okay, usually it's like disgusting, dirty brown, because they're, they're really sloppy eaters. Anyway so instead of clustering in just the intensity space, we could cluster in RGB space. So this is notionally drawn here. Right, where you can see that each point now belongs to a different color cluster and here they are. And so you're grouping these pixels based upon their color. So now you're doing a quantization in three dimensional space. Here's a nice example of using color instead of intensity this original image here. And you can see if you take a look at sort of the, the, the clustering based upon intensity over here, it doesn't do a particularly interesting job of segmenting the cabbage from the broccoli which is something you should always do because you like one and not the other. All right, but over here you can see that based upon the different colors you get the different segments. And it's not rocket science, but it's just this idea that you have some things that have this similar intensity but are very different in their chroma. Right? Their, their colors. And we talked about

color space, last time, all right. Notice, however, this pepper here and this pepper there. This system does not know at all that those two peppers are distinct. Right, because the only thing it uses to create these clusters is the distance in color space. You might ask can I do something smarter. Can I do something smarter? Thank you very much, yes you can include not only clustering in say intensity. But you can also cluster, so here we have, we have this little part of the black ear and this dark part of the eye, and you might want them to be in separate clusters. Well, how would you do that? Very simple. Instead of just clustering on intensity, you would add the position as well. So now, we have a three dimensional space of intensity, no. Intensity x and y or, as shown here, intensity x and y. Here we have the black pixels. Together they're low in intensity but separated in different areas in x and y, and so you would get a different, two different, clusters. And of course. If you've got intensity and you can do position you have, can have color and you can do position. And the point here is that your feature vector is getting longer, looking for features that you'd all like or, as many as possible, to be similar. In order to say that everything is in the same cluster. So, notice that if I include the x and y here, so, for example, these two pixels, this green up in the top left and the green in the bottom left. They can't be clustered in the same segments because they're different along some dimension, namely the y dimension. All right? So that's, that's the utility of adding these dimensions.

8 - Pros and Cons

K-means or something like k-means for segmentation. It has some strengths. It's very simple to do. By the way, it's guaranteed to converge. You can sort of show that. But it has some distinct challenges. First of all, it tends to be very memory-intensive, because you're keeping all of your points and you're shuffling them around in, in in a, in a membership way. Perhaps one of the biggest ones, well actually there's two big ones. One is here is, you need to tell it K, you need to tell it how many points there are how many clusters there are, a priori. You can try to discover, that but K-Means doesn't know anything about how to do that. It, it's sensitive to initialization and outliers. And in particular, what it's looking for are nice, essentially spherical clusters, where they're spherical in feature space. That is, it doesn't want any point to be too far away. And the problem with that can be illustrated in, in a figure that looks like this. So here is a figure with two clusters in it this one here and this one there. Now i've drawn them, or that I should say they've been drawn in red and blue to help you think that they should be separate. But if you, if you were to look at that pattern, in just black and white, you probably would say, well there's this bent cluster here, and then there's this bent cluster there. But of course, if you were to run K-Means, you would get what's shown here. This would be one cluster, and that would be another cluster because K-Means is looking for these spherical clusters.

9 - End

So we've seen using k-means that clustering is a for, is that segmentation is a form of clustering. And so far we've just clustered in intensity space. And things get more interesting as we either add more interesting dimensions, that is other descriptors. And also using a clustering algorithm or a way of finding these clusters that are be, that's better than just k-means. And that's what we're going to talk about next.

9A-L3 Mean shift segmentation

1 - Intro

Welcome back to Computer Vision. We're going to continue our conversation on segmentation. Last time we, we introduced it as segmentation as a clustering in feature space. And feature space could include things like position, so you might get nice segments that are localized, within an image. This time we're going to look at, some better algorithms than say, k-means. And then also

some better, or I should say, some more extended feature spaces which allow us to do better segmentation.

2 - Mean Shift Algorithm

So we said one of the biggest problems with K means is that it was really sensitive to, sort of needing to know how many clusters there were, what K was, and also possibly to the initial conditions, that you gave it. So the assumption of K means is that there are essentially these spherical clumps within feature space. So if you're thinking of these as a probability distribution, it is that there would be a couple of modes, peaks, in our features space that, and sort of falling, nearby those peaks is a sort of fall off, and that our pixel intensities are distributed according to a probability density function that has these modes. So if you could find the modes, you might be able to segment the image that way. So, the approach that we are going to talk about for doing that is called the mean shift algorithm, and we actually introduced this when we were talking about tracking. It's a little bit easier to talk about that when we do it as a cluster. So the mean shift algorithm was designed as a way of finding these modes, or these local maxima of density, in some feature spaces. So here, by the way, this, this is a beautiful, pastel image of, these houses, as the input image, and what we have here on the right is the plotting of each of these pixels in the L u v color space. You remember L u v color? This is why we're doing this whole conversation after, we talked about color. Because if I told you L u v and you hadn't heard it and think I was doing RGB, no it's L u v, right? And you can imagine right away, that looking at a picture like this, the color, the chroma, is what matters, and the absolute intensity might matter less, right? Because the, so let's see, using this yellow here. So this is a brighter yellow, and this is a darker yellow, but I would like to have them both be in the same color segment. So L u v makes that easier.

3 - Mean Shift in Space

We're going to use the mean shift algorithm to find the modes within that distribution. By the way, the slides I'm going to show you, I've showed you before, and again these are all over the internet. It says originally from Ukraine. It's in Sarel, and if that's who did it, and I thank you for doing these animations, because everybody uses them. So the idea of mean shift is the following. Here, I have the distribution of points in some future space. I plop down some region of interest. And that region is drawn here as a circle, but think of it as a Gaussian blob, right? So, it actually is more weighted in the middle, than it is to the outside. Mean-shift tends to work a little bit better if you have a center waiting. The reason for that is, if I've got a hard circle on the end, then if I move it a little bit, I might run into a bunch of new points, and it'll change my value quickly. And I'd like to things to always be smooth when I move things around, so by having a fall-off in the galaxy, and, as I get there sl, as I get there, those points have a, a little bit of influence, and then they gain more influence. So what happens is I put down this region of interest. I compute the center of mass, that's the weighted center of mass, and the difference between them, that's called my mean shift vector, and I just shift my mean there. Isn't that a beautiful animation? And what do I do? I do it again. I compute the weighted center of mass. There it is. New Mean Shift vector shift it. [SOUND] Marvelous. And what do I do? I do it one more time, and I keep doing this until I've converged. And there are some interesting proofs of convergence under some reasonable assumptions and, that finds you this mode.

4 - Mean Shift Clustering

So how do we apply this to clustering, okay? Well, basically the idea is you, you're going to clust together all the data points that are in an attraction basin of one mode. So, what's an attraction basin? An attraction basin is the region for which all the trajectories lead to the same mode, or all the points go there, all right? So here's a very simple example. So here is a set of points, drawn from a distribution. And you can tell, obviously that, the idea is that there probably is a mode here, and theres a mode there. Great, the question is, which points belong to which mode? And that's

done very simply by, if you were to start your mean shift in each of these locations, which mode does it end up at? All right, and that's illustrated in this figure here, right? So, here we're showing you the area of which all of the, the mean shifts get to this mode, the mode on the right. Likewise, everywhere over here, they get to the mode on the left. So it clusters your feature space using these basins of attraction to the two modes, all right? So, when we're going to do this for segmenting a color image, what we have to do is define some feature. So maybe we're using just u and v or maybe u, v, and x, y, if we want a color and location. We have to initialize the window, the windows at every pixel, that is every possible color point in that LUV space. Now, would you do every one? No, you would take a sampling of them. You do mean shift until it converges. And then you merge all the pixels, all those initial points into a single cluster that all end up at the same peak, all right? And so, that looks a little bit like this, all right? So here, L and U, this is L and V, okay? These are the actual pixel colors, distributed. And if you run your clustering, and here, it's being done in this, LU space, all right? If you run your mean shift here, okay, you will see, see all these points going up to that peak right there? And then there's a peak here, and a peak here, and a peak here? Those, let's see, one, two, three, four, five, six, seven, so there's seven modes here, and that means there seven basins of attraction, which means you would end up with seven regions in the image.

5 - Mean Shift Segmentation Results

So let's take a look at a result. This comes from this nice little database that was published, based upon this Pami paper. Comanici was one of the original developers of mean shift as applied to Computer Vision. And what you can see here is, so here's our original image, all right, and you apply mean shift, and, it does a really fine job, all right? I mean, take a look at this brick structure here. And you see all of those pixels are put in the same region. Likewise, you get this front of this building here. The whole sidewalk is on one cluster with the exception of over this area, right? Well, that's in a dark shadow from the tree. You'll notice the whole shadow has been put into one cluster, because the system can't, at this point know that it's a shadow. All right? But I think you would agree that if you take a look at this picture here, and you were to draw boundaries around where the different clusters were, okay, you would probably end up with an image that looked something like that. Here are some more examples of a mean shift segmentation result. And you can see it does a pretty good job. You know, take a look at this picture down here. All the water is put into one cluster there. This foliage is broken out nicely, the islands, etc. So if you've got the right feature space, it does a pretty good job of finding the clusters. Finally, I want to show you an example that also comes from that same original database. And again, it has a nice, original image here of a, of the mountain. And on the right hand side are the segmentation results. So, the good news is, you can see it does a pretty reasonable job. I mean, it pulls out this section, all of the section up here that didn't have the snow on it. And maybe this, section here as well. But, notice all the texture that's going on in here, okay? Well it's not that that's wrong, it's just less clear [LAUGH] than it's right. What I mean by that is, it's not totally clear what you're trying to do with this picture, that you're carving it up into segments. That was original idea behind the super pixels. The idea was, let me carve this thing up into little atomic units that I know don't have to be broken up again. But, whether I recombine them, or use them together in some way, depends on upon the future processing that you have to do. So often, what you'll do is a segmentation step to create these broken up, these regions, and then figure out how to process those regions going forward. So the segmentation then, in some sense, is a preprocessing step to give you a much smaller set of units to work with.

6 - Pros and Cons

So that's a description of mean shift as applied to segmentation. It's a, it's a, I will tell you it's a very strong technique. The cool thing is it automatically finds these basins of attraction. So, you don't have to tell it how many modes there are. You just tell it, in fact there's one parameter choice essentially, which is that window size, how big that Gaussian is. There's also, when you talk about

that, because you're operating in a feature space, you actually have a distance function, right? because when my Gaussian is over some distance, so if I want to scale color more than texture, oh, texture, we haven't talked about that yet. All right, we'll do that next. Or color more than luminance you make that decision ahead of time. So, there's this parameter choice of window size, but there's also a, a general distance metric function choice. But you have that in every sort of clustering that you do. It doesn't make any assumptions about the shape of the, the segment in the image, or the cluster in the image, it just does this in the, the probability of the distribution of the pixels. And it's a very generic technique for finding multiple modes, and people use mean shift for a variety of things like tracking, when we actually talked about doing that previously. You know, as in anything else there are some down sides. It does depend on this window size, and sometimes figuring out that window size is a bit of an art, with respect to the nature of the set of the images that you're looking at. And, the other problem is, because you are sort of moving in feature space, oh, and, and, you want to make sure that you get a reasonable number of points within your, your region of interest in order to shift the mean. Well, suppose you've got a big feature space. All right, and what I mean by that is you've got fifty dimensions. Well, almost by definition, your future space will be sparse, right? Because the, sort of, the number of little, if you, if you let's suppose you carved up every dimension into, like, ten buckets, well, if there's fifty dimensions right? That's ten to the 50th buckets. So by definition, there's going to be lots and lots of buckets that have hardly any points in them. So mean shift needs you to have a reasonable number of points within that region, so it doesn't work as well in high dimensions.

7 - Segmentation as Clustering

So, so far we've been doing segmentationous clustering, and we've been using some feature space based upon color, and then we have these different algorithms. One of the challenges is, is that things like just color or the brightness, or, or position, which is something else we looked at, alone are not enough to do segmentation. And here are some examples. So you can see that, you know, that generally in this area here the colors are about the same as this area there. Likewise, the brightness here, the average brightness is the same as the brightness over there. Now you have no problem seeing the differences between these, but that's because you just don't use brightness positioned color. You also use texture, right? Obviously, the texture here is different than the texture there. Likewise on the fawn. The texture there is quite different from the texture here. So the question is, how would we introduce some, some measure that was sensitive to texture into our segmentation? One way of thinking about texture is this. So here I have this picture, 'kay, and I've got clearly different texture here than I do here, than I do there, all right? So here's one approach. Let's compute a series of filter outputs. So here's a 1, 2, 3, 4, 5, 6 oh, 24. It says 24 right there. Here's 24 different filters, okay? And they may have different orientations, so here we get different orientations. And they have different scales, so they go from a courser scale to a finer scale, and here's one's that are just sort of little center surround. It almost doesn't matter. Well, it does matter, but it almost doesn't matter which filters you use. The idea is that you're creating a new set of dimensions that you'll have for every point, all right? And when you do that, the idea is that hopefully along these different dimensions, and there's 24 or however many you have, you would get some clusters, depending upon the types of textures that you have in your image. And that's what's being shown here, right? Is you have these clusters in this image space, all right? And so here's an example of using let's just suppose we take a look at, this is the derivative in the X direction, so how much gradient there is going in the X direction at one particular scale. This is one of the dimensions. And this is in the Y direction, so it's your, measure your gradients in the Y direction. And the idea is that, well, in this case, there are little areas that have in this case, windows that primarily have vertical edges, right? Because they've got strong x-gradient and no y-gradient, likewise, here's horizontal. This has a little window that would have a texture in both of them and these would be areas that have hardly any gradients at all.

8 - Texture Features

And what we're going to do, actually, is do clustering twice. The first thing we do is, we're going to cluster in that big feature space I was just showing you. And those clusters are going to be referred to as textons. Now this is a word, I think it's a word that was coined by Bela Julesz who was a psychophysicist, guy who did random autostereograms quite a while ago. And they were referring to things like T-junctions and little end-stops and other textural descriptors that ex, existed in, in images as sort of he saw them. And the idea is that these clusters in the feature space would be different little textural elements that were found in these images, right? And so you would describe each little window, okay, by what's referred to as its texton histogram. That's what this thing is going to be showing here. We'll, we'll take a little bit more of that. But the idea is that, here there is just a single value at every pixel, which is the single texton that's tends to be present at that pixel. That is, which cluster it got mapped to. And you can see that, you know, if you were to just cluster based upon this, it would be kind of weird because, yeah, this is not one nice thing because it's a single texture. But the idea here is that, even though this is a single texture, there are all these different little textons that are present. So instead of segmenting the image based upon which single cluster you're mapped to, instead what we're going to do is we're going to take little windows. So here's a little circular window and we compute its texton histogram. Okay, so now we've got a whole bunch of different textons and this is the count of the number of those textons. So I get a histogram at every window. So here's another window, and it has kind of a similar histogram. And here's a third window. Well, its histogram looks nothing like that. And the idea is that you're now going to cluster in the histogram space of these textons in order to be able to get your segmentation. All right, and how well does that work? Well, let me show you an example. So here is a picture of a rhinoceros, no, a zebra, I guess. I don't know, a thing with stripes, okay. So suppose I cluster it based upon color. What would that look like? Well, it would be a mess. Okay? And there's your mess, color-based regions, and you'd say, well, that's a terrible idea. That's a terrible idea So what would be a better idea? Well, how about we cluster based upon texture? And you can see that you get these broad clusters that seem to do, you know, a reasonable job. I mean, it, it does make the entire, you know, this region here, which is sort of this big region there. You'll notice that it's not, you know, because of the scales that got used in this case, it didn't find these legs here, or, or this one there, but that's a function of sort of how you go about describing it. And I will say that in general, this is kind of an under-segmented result that probably should be more segments, but clearly this is way better than this color-based approach, because it allows us to describe the texture.

9 - End

We've seen two methods for expanding our ability to do segmentation. One was the new method, the new algorithm of finding modes within some feature space and that, and it was you know, didn't require you to know the number of modes to begin with, and all those other properties. And then in some sense, more importantly, no not more importantly, in addition, we found a way of expanding our feature space to include something that describes texture. And what you did was you created a big t, feature vector, texture feature vector at every point, and you would cluster that to get these keywords or these textons. And then you would, for every region, you would get a histogram. And then you would cluster based upon those histograms, and that's what gave you your segments. Combined, these gives us a pretty powerful method.

9A-L4 Segmentation by graph partitioning

1 - Intro

Welcome back to Computer Vision. We're going to continue with our conversation about segmentation. And this time we're going to up a little bit, the sophistication of some of the methods that we're talking about. So far, all of our segmentation work has really talked about sort of

clustering pixels within some feature space. That is sort of describing each of the pixels so i might build the clusters in position, and and intensity or position in color, or even in a texture space. What we didn't really do was talk about the connection between one pixel and another. Right, I mean after all if you're doing a segmentation the idea is that you're going to be drawing a boundary and you're going to say nope you, you go to the left you, you go to the right. So I, want to talk about the connection between pixels.

2 - Measuring Affinity

So to do that, we have to think about pictures as of not just a collection of pixels, but actually as a graph. And what I mean by a graph is shown here. So, in the graph we have a, a node for every pixel, okay? And between every pair of pixels is a link, okay? Just as in any other graph, those are the edges of our graph, right? And on each edge, there's what's referred to as an affinity weight, or just a weight, and here that's written as wpq, all right? And the idea is that I'm going to need an edge for any weight that is nonzero, okay? And the weight is going to measure the similarity between two pixels. And the two, and the similarity will be essentially inversely proportional to the difference. So that just means that the, the more different they are, the less similar they are. And another way of saying it is if the difference is too big, I'm going to say the similarity is zero. So those edges go away. So I'm going to build a, a weight matrix, using this affinity. So I, a standard affinity, for example, might be that I'm just going to use a Gaussian, right? Where I say the, the affinity between two pixels, i and j, is just this exponentiation, as a function, just as exponentiation as a function of the distance. So of course, there is a parameter here. And here, you can see the change in affinity depending upon what that parameter is you get these different curves. And essentially if you use a very small sigma this thing goes, grows quickly. So in which case, only nearby pixels get any reasonable affinity at all. Whereas if you have a large sigma, you can connect, you can have strong infinities in between points that are somewhat far away, far away in your distance space. So if your distance includes just color, then it would be that. But typically, our distance is also going to include actual distance.

3 - Segmentation by Graph Partitioning

So if I have this graph, segmenting the image is just breaking the graph into segments. So here I'm showing you that I, I took all those points I had before, and I broke this thing into three segments. So that would mean that there were three regions in this image. And the idea is that what I want to do is I want to delete links that are between the segments, right? So maybe there used to be a link that connected this point to that point, and it's been broken, all right? So I'm going to delete links that go between segments. And so, which links should be easiest to break? Well, obviously those with the lowest affinity, right? because if I break them they're saying, you know, the blue pixel and the red pixel, which maybe have great distance in my feature space. So, they have a very low affinity between them. By the way, here's a, an actual example of segmenting this tiger against the background, and you can see that it's got spatially contiguous regions by, by, doing that. So, what I need to do is I need to construct a distance function, such that similar pixels will end up being in the same segments, if, if they're near by. And dissimilar pixels, ones that are very different, will end up in different segments.

4 - Graph Cut

Now, many of you, because this is a graduate course intended for computer science, I'm going to assume you've seen some form of an algorithms class, and you may have heard that there's something called a graph cut algorithm. And a graph cut algorithm does exactly what you think. It cuts a graph. So here I've got a graph and the idea was originally it had lots of connections in it, and when I'm all done, I've decided to break this graph on those four links to create two s, different graphs, a and b. And they're totally d, disconnected, so those would be two segments in our image. And the idea is that the cost of a cut is quite simply just the cost of all of the segments that you

broke. So that's what this thing shows you, here. I take, for all of the, if there's a p in a, and q in b, and I've separated a and b. What's the cost between all of those? And a graph cut would give you a segmentation of the image. So the question becomes, what is a good graph cut, and how do you find one? So there are algorithms called min-cut algorithms, sometimes called max-flow because they're they're duels of each other, they're the same, that allow you to find that segmentation in a relatively not too complex a matter. Fast min-cut algorithms exist. So the original min-cut Fulkem, Fulk, Fulk and Fulkerson, you may have learned there are subsequent more rapid ones. We're not going to go into those here. All you need to know is that you can go to your local algorithm store and buy yourself a good graph cut algorithm and the more you can spend the faster a graph cut algorithm you can get. And the idea is that, that would partition your graph. A problem though, if we just use min-cut for doing segmentation, is that min-cut by definition is going to try to pull out little segments that have as few connections as possible. So what it tends to do, is it tends to pull off very small little segments. because by definition a little segment will only connect to a small number of other nodes. So, really what you want to do is you want to something a little bit smarter than that. In an intent, in fact, what you want to do is what's referred to as normalized cuts.

5 - Normalized Cut

So normalized cuts fixes this bias that Ncut has of, wanting to pull off these little elements. And the way it does it is it's going to propose a cut, and it has the traditional measure of cutting A and B. But it's normalized, by the association within A and the association within B. And a simple way of thinking about is the association is just sort of how well connected, summed over all of the nodes within a, in a segment how well connected those points are. So the problem is, is if you have a two segments, and one is very small, it's going to have a very small value. And if that denominator becomes a small value, that whole element becomes very large. So, even though the, the other element might be a small size, the fact that this one became small will penalize that a lot. So, by doing a this normalization you can end up with a much better cut of the graph. If you just wanted to do, the best end cut normalized cut that would be computationally-prohibitive. But there's an approximate solution that exists using what is referred to as the generalized, or solving the generalized eigenvalue problem. I'll just show you a very simple, not simple, I'll show you a very high level description of, of what goes on. And if you want more details, you can go and take a look at papers or Wikipedia or anything like that. And so here, here is how it works. Let's suppose we've got the matrix W, which is the adjacency matrix. So W,i,j is just the adjacency between pixel i and pixel j. And then, there's a special matrix, this diagonal matrix, that is just what's called the degree of the node. For every vertex, how many other vertexes is, are connected to it, right? And so in this case, it's the sum, so the, the, it's a diagonal matrix, so the, the degree of pixel i is just the sum of all the affinities i,j summed over all the j. And then what you can show is the normalized cut cost, can be written in using this, simple expression, where this vector y is what's referred to as an indicator vector. So it's a vector that's got positive ones and negative ones as to whether, remember cutting this into two parts A and B, so it's a positive one if that point belongs to segment A. And its a negative, constant otherwise, all right? So just to solve even that for y is very difficult, but, what you can do instead is, this is just written out here. We solve this eigenvector equation, and what that'll do is it'll solve for you for these y's directly. But these y's, they're not going to be binary, you know, minus one or plus one. They're going to have, continuous values, and what you then do is you use that vector in order to solve for, the partitioning. And this is related to, spectral partitioning, for those of you who are familiar with those algorithms, and it's just a, an ei, an eigenvalue way of approximating the solution to this partitioning, because the full solution would computationally too difficult.

6 - Results

So let me just show you some of the results that you can get using that. All right? So here are again, this is taken from the the Berkeley database. And because Shi and Malik, Jianbo Shi and Jitendra Malik did normalized cuts. And you can see that there are some pretty good results here. You know,

you like, I especially like this one here where it breaks up the picture of the people in, in a particular way. Now, of course, these, segmentation is tremendously influenced by that affinity matrix, and the affinity matrix is defined by that distance function. So, we're back to feature spaces again, right? We have to know what features we should be computing a space over. We may have to scale the features because all of that determines the distances. But, once you do that and you play with that, you could end up with a really powerful, segmentation method. By the way, you'll notice that there are more than, two segments in these pictures. Well, so how do you do that? Well, the simplest is you partition your graph, and then what do you do? You partition again, partition again, and you need a stopping criteria based upon, sort of the affinities of the, the links that you're cutting. Here are some more examples also taken from the Berkeley database for segmentation.

7 - Pros and Cons

So, that's the normalized cut approach, and it's one of the more effective methods within the whole graph-based way of doing segmentation. The pros are because it's a graph structure, we've got lots of algorithms for dealing with craft structure. So, it's a very generic way of doing it, because you separate out this graph structure question from the affinity matrix computation. So the affinity matrix is where you get to play with things, like your distance functions, and your feature spaces. And then, your graph algorithms just run given that that, that matrix. So again, it doesn't require a model of the data distribution. It just requires a distance function that respects that. There are some challenges. In general, the complexity of dealing with this can be pretty high, because you now have this, these matrices that are essentially n by n, where n is the number of pixels. So you want to be careful exactly how you manage that complexity. If you have densely connected graphs, because you want to allow for large regions, et cetera the system has to work even harder. And the, the spectral decomposition approximation to the normalized cut is a little bit trickier to do. And, and then there's this question about this balancing of the partition. So you remember, we introduced the normalization to the min-cut, in order to not get small pieces, because there was a bias towards small pieces in the algorithm? Well, another way of thinking about this is, we've introduced a preference for balanced partitions, for balanced regions. So if you really wanted to have small, little spots being pulled off of something that was bigger, normalized cut finds that kind of painful, because it wants to have these balanced partitions. Having said that, I will say that it is one of the more effective segmentation methods that people currently use. And I think it's because it has this nice dissociation between the affinity matrix computation and the graph segmentation method.

8 - End

So that ends our conversation about segmentation. You can go from these, sort of, very simple methods that just, sort of, cluster in some feature space, to ones that cluster the feature space and then do histograms and cluster around those. To the more gra, the more complicated graph-based methods which use fo, some form of spectral partitioning in order to div to split the graph into multiple sections.

9B-L1 Binary morphology

1 - Intro

Welcome back to Computer Vision. This is another lecture of mechanisms or methods that are, they're not sort of current computer vision research, or really the state of the art in Computer Vision, but they are very important methods that we make use of all the time for a variety of operations. And what we're going to talk about today is really some binary image analysis. There are lots of times that we produce images that are essentially zeros and ones. One talks at zero is a background and one is a foreground. So, for example, I might be looking in fact we talked about background subtraction when we were doing, detection of moving objects. Right, and so basically the ones

where the stuff was supposed to be moving and the zeroes were supposed to be where it wasn't. And in fact, there was this example that I showed of people sitting on the steps of the University, and we did a median filter, and then we subtracted them off. We did a thresholding, then I did, I said, we did something called morphological operations to clean it up. Today, we're doing the morphological operations, and it's often buried deep within any functioning vision system.

2 - Binary Image Analysis

So i mean here's an example of just showing a binary image. Okay, where you've got a circuit with some leads that are connecting it and some of those leads have shown up well and some of them show not so well. Or maybe some of these things are noise and you want to get rid of them, and here of course this is a notional version of a binary image where you've got ones and zeros. So binary image analysis is used in all sorts of things. We're just showing you sort of document inspection, but parts inspection. So you've got a part and it's supposed to have particular property to it and you don't want to make sure there are any holes that shouldn't be there or the holes that are supposed to be there, are there. Mostly for two dimensional stuff, also manufacturing. You have some idea of what the images are supposed to look like, but it's basically, where is there stuff and where is there not stuff, is what matters. And so you can think of those as binary images. And of course, document processing, where the where basically, if you think of it as just black ink on a white background, it might be very fine, but there's, there's not really supposed to be grey. Now those of you that know about anti-aliasing, that has to do with limited resolution. We're not going to talk about that. We're just going to talk about sort of binary images.

3 - Kinds of Operations

You know, what kinds of operations do you want to do on binary images? So, you know, often what you want to do is you want to separate objects one from another, that are sort of accidentally touching, and you want to collect together pixels that all belong to the same object. And then another thing you might want to do is compute the features of a binary object. And we actually did this also. Remember we talked about hue moments, and we were talking about moments in general? And moments are a descriptor of the shape of a binary patch. And so, again, those are things that you would do on binary images. So here's an example. This was a thresholded, red blood cell image. And by the way in the old days, not so old, in fact mom, sorry. My mother, I think her first job out of college was as a medical technician. One of the things you had to do was count white blood cells in a, under a microscope and for a certain amount, and you would count the density. Actually, people sometimes still do that, even today. But what you'd really like to be able to do is spread out a bunch of blood cells and have a machine do the counting. How many red blood cells? How many white blood cells? Especially, if instead of just doing a patient at a time, you were doing a massive study, and you had tens of thousands of samples you wanted counted, right? So, here's an example of taking an image, thresholding it. And, you know, if I were to ask you to count how many red blood cells there were, you could probably find all of the, well let's start with one that's easy, right? You say a-ha, here's one that's pretty separated, no problem, here's another one. Oh, but this is a weird one, look at that. It's got a hole in it and here are two, those are probably two, right? But it's one blob connected by a thin thing. So you'd really like to be able to take this image and then fix it up. You know, because, in this particular image, there was 63 separate objects detected and a normal blood cell has an area of about a 50. But you'll notice there are some things here, where obviously there were things that were touching each other that were counted as single cells, when actually, they were gobs. And then there were things that were really small, right? And so here, they were just using the number to determine which kinds of things are cells. And what you'd like to do is you'd like to be able to separate out each one of those cells. So, we're going to do some binary image operations to do that. And what kind of operations are there? Well, first one we're going to talk just a little bit about is thresholding. because in some sense, in order to get a binary image, you have to go from a color or a gray-scale image that has continuous values or, lots of values to something that just has two zero and ones. So determining a threshold is always a

challenge. Then, what you'd like to do is you'd like to find all the pixels that are connected. And that's called connected components analysis, and we'll, we'll talk about that, that one's used a lot also to find independent blobs. And then you want to sometimes improve those blobs, and that's referred to as morphology, and we're going to talk about a variety of morphological operators, including some of the more important ones. Then once you've done all that, then you may compute some features on those blobs, like I was talking about before. Moments, areas, these kinds of things.

4 - Thresholding

Let's talk about thresholding, and this is an example, I think this might be taken also from Linda Shapiro. So the idea is that you're imaging these cherries, okay, and the idea is that the background here is very dark. Okay? And a healthy cherry is kind of bright, all right? But a bruised part is medium dark. And what you'd like to do is be able to say, do I have a bunch of bruised pixels. So in this particular one, they've removed all the black things, but you see these are the grey scale values, right? And you and I look at this, and we say yep, you know what, here's my threshold, and this is the number of pixels that are bruised, and this is the number of, of pixels that are good. And the idea finding a threshold like that automatically is not trivial. Generally what you do when you look for thresholds is you take a histogram of the intensities, right? So, this might be, you know, go from zero to 255, and this is the number of pixels that you have in the image. And the question is, you know, what are your distributions? Unlike this one, which had this nice, clean dis, separation there, this one, it's a lot more complicated. Are there two, or are there three? I think, you know, from my, from my take, I'd say, oh, there's three lobes there, all right? And there's different ways of trying to do that. A simple method, this is again, a long time Otsu's method, it's, it's a binary visionary method. It basically says find, assume that your system is bimodal, and find the cut off that minimizes the within group variance. Okay? So you're only allowed to draw one line, and so, and it's weighted by essentially this, these, these values. And when you do that, what's kind of cool about it is you can do that when it's certainly bi-modal. When it's actually tri-modal, let's say, you'll typically pick one of the modes, all right, and then, you can, look, do it again, and, you'll cut it again, and what's kind of nice is when you slice a mode that shouldn't be sliced, you'll get a reduction in the variance, but not very much. And you can track how much that is. And it, and the method can be adapted to do a pretty good job of thresholding a continuous grayscale into either a binary or a three valued system.

5 - Thresholding Example

This gets done sometimes, in medical imaging. So here is a, an old scan. Probably a, I'm going to guess CT. And the image on the left is the grayscale, and then they've thresholded the image on the right. And what you'd like to do is, you'd like to say, you know, there's this chunk here, and there's this chunk there, and maybe this chunk, and this chunk, and some things there, and that. And maybe I'm going to ignore some of that other stuff. The first thing that we're going to look at is what's referred to as a connected component analysis, all right? So some of these are a little hard to see. But we'll just focus on this one right here. The idea is I'd like to pull out this thing as being distinct say from this thing, okay? These are not connected so, I would like to label this maybe as a 2 and this as a 3, all right because I want to find all of the connected components. Or connected set of pixels. The connected components algorithm or operation takes a binary image and it generates what's referred to as labelled image. So every pixel now it used to be a 0 1, it's now giving it a label that is the number of the object to which it belongs. So right, if there is just one object, everything will be a 0 or 1 and the 1, right? But if there are two objects, there'll be 0s, 1s and 2s. All right, by the way, one of the things that we're not going to talk about is when we talk about a pixel being connected to another pixel, there's what's known as 4-way connected and 8-way connected. 4-way means that you're connected to north, south, east, and west. Did I get those right? Well, it depends upon which way you're looking. 8-way connected, says I'm connected to the north, south, east, west, and I'm connected to northeast, northwest, southeast, southwest, all right? For what I'm going

to show you here, we're going to assume a 4-way connected the, the algorithm is pretty much the same if it's 8-way.

6 - Connected Components

There are a variety of connected component methods. The one that most people use is referred to as row by row and it's sort of, classic algorithm and it's simple to go through. And there are some efficient speed ups that we use so, that you can run it quickly in industrial systems. But, these days running your connecting components is really the thing that's taking your time because the machines have gotten much faster and the algorithm itself is pretty straightforward row by row. So, let's start with a picture that looks like this. Okay. So, we've got 1s and 0s. And if you look carefully. You'll see there's really only one object here. Okay? Those are all the 1s. And what we want to do is we want to eventually determine that all of them should be labeled as a 1. Okay, but this system doesn't know this yet. All it knows is that there are 1s and 0s in the pixels. All right? So, here is the algorithm. We start with a variable at 0 and what happens is we go across, okay? And when we find a new pixel, if it's not connected to anything before it or above it, and of course there's no above and there's no before, we'll increment our counter, all right? So we start by labeling this 1 as a 1, and then we go forward, and then we're not connected, and then we would get to here, and so what would that be labeled, what comes after 1? 2, okay. So we label that as a 2. So, this is going to become a 2, all right? And then, go over here and then what are we going to get when we get to next one? What are we going to label that? 3, okay? So, that works fine. The reason that this one is listed as a 1 even though you've scanned it. It's, it's connected to the one above it. You say, oh I'm connected so, I'm a one. Likewise when I get over here, likewise when I get over here, it's connected to the one above it. So, these are all labeled as 1, 2s and 3s. But you see where it says here, remember conflicts. All right? As I'm going through, when I'm labeling this pixel, okay? I now have a conflict. because when I look behind me, it says that I'm a 1. And when I look above me, it says I'm connected to a 2. So what I do is, I pick one of those, maybe the 1, but I remember that I've got a conflict there. Likewise, I have a conflict here, and what I learn is that 1 is the same as 2 because it's connected there, and 1 is the same as 3, so it's connected there. So, after I'm done labeling all of these, and I've got 1s, 2s,and 3s, I relabel everything as a 1 because all the 2s get relabeled as 1. All the 3s get relabeled as 1, and that way the system knows that there's exactly one object and all the pixels with the 1s are the pixels from that object, and it works just like that. It works great. Here's an example from that CT cross-section I was showing you before, and you can see that after some various other operations that we'll talk about in a minute, it's a, it's able to label these different elements. Okay. And what's nice is those, those elements have semantic meaning to the radiologist who's, who's looking at these things. You know, it's also useful for if you have some clustering algorithms. So here you are seeing, just some clustering pixels, in a, in a, in a beach sce, well, it's no much of a beach, is it. No. Yuck. Okay, it must be the Northwest. No sand, just muck. Sorry, Seattle. And you see they're labeling these different things using connected components.

7 - Dilation and Erosion

I mentioned the idea of doing mathematical morphology. That is, fixing up your picture. Okay? And, fundamentally, there are two basic operations, and they're referred to as dilation and erosion. We're going to go through both of those in detail. Once you have those two basic operations, you can combine those to make more operations. Eh, there are a whole bunch of them, and this used to be a whole industry doing mathematical morphology. The ones that we use most to date, or I should say, the ones that we use most currently, are various forms of closing and opening, and we're going to talk about those. Then there's thinning and thickening, and a few other things, but they all stem from these two basic operations of dilation and erosion. So let's go through those. So dilation is exactly what it sounds like, right, when you dilate something it expands. And so dilation, expands the connected set of ones in the binary image. So, here you see this thing expand, if you get bigger, and really, you know, just grew out, and were going to talk about the mechanisms by

which you do dilation, and something called the structuring elements which control the shape. This one is an interesting one, right? Everything expanded and has two effects, first of all, things got a little straighter. But also, notice that these little cutouts went away. Likewise, when this thing got bigger, okay, the holes went away. Dilation in general can be used for, like it says, filling holes and gaps. The, inverse, converse, reverse? The other, of dilation is called erosion, okay? Erosion, just like when beaches and other things erode, is you're essentially shrinking the shape. Right? So this big bar becomes a narrower bar, this big circle becomes a smaller circle. Here's where you start to see its power, right? These two connected, sort of elliptical things become, like here, it's this case, it's these two circular things, you, you, could actually do it, go from those ellipses and circles. The important thing is that middle part went away. And here you see that these little protuberances went away when the entire thing shrank, and that's referred to as erosion, okay.

8 - Structuring Element

To go over the mathematics of both dilation and erosion, we have to talk about something called the structuring element, okay? The structuring element is a mask, okay? It's a shape that you're going to use to do these operations. And, it can be any shape. For today, the shapes are going to be ones and zeroes. They can also be ones, zeroes, and x's. And the x's are don't cares. But nothing we're going to talk about today will, will involve don't care, so I'm not going to talk about it. What's important about the shape is that they all are defined with respect to sometimes called an anchor, or reference point, or an origin, right? And usually, not always, usually that origin is in the middle of whatever shape you're making use of, okay? So given a structuring element, we can define both dilation and erosion. So dilation is as follows. Basically, I've got some binary image, B, okay? And that's here. I've got a structuring element, S. And the structuring element has to have an origin. Basically, what you do is, you're going to take this dilation structuring element and you're going to run it around here, placing the origin at different places. And wherever you place it, you're going to take the OR of all of the ones in the structuring element with the ones underneath it. And so what that means is if any of the ones land, of the structuring element, land on a one in the binary image, the aura of all that will be one. So, Arpin did this and he's very proud. So, there's my structure element, and he, there he goes, [SOUND] very good. Cool. All right? So you basically run it all over, and what the result of this, okay, is this val, is thing here. And let's just take a lot at that, right? These are our two original ones. Okay. So when I put this structuring element down here, like that, nothing was a one. In fact, as I slide it all along, nothing is a one, so those are all zeroes. Right? But when I put my structuring element here, notice it overlaps there. Okay. So this value which is the origin for that first position, that gets a one. And as I slide it along, that's how I get the, the, the rest of these. So that's dilation by a particular shape. Now I'll tell you sometimes you'll dilate by sort of a funny shape like that. Usually your shape will be symmetric. Sometimes it will be taller than it is wide because you're trying to extend things in one direction as opposed to another. But this cutout kind of thing would be a little bit unusual. In fact, here's a nice example. So here's a binary text and you can see that there's white pixels but they've got lots of, if, you know, if you take a look at that m, you can see it's kind of, bunch of little spots in there that seem to have some holes in it. So if I use a structuring element that looks like this. That, essentially, says that if I touch anybody to my right, left, up or, or below, I want to turn on the pixel. And, when I do that, you get the really much better result, right? I mean, it's much easier to read the one on the left, the one on the right. The one on the right might be uglier, okay, because it doesn't look so crisp, but there's no question about reading it.

9 - Dilation Quiz

Quiz. So here's the quiz, all right? So, what's the result of this dilation?

10 - Dilation Solution

Basically, remember, what you're going to do, is you're going to say, here's our origin. Where when I put this, structuring element down will its 1's touch the 1's in this picture, all right? Well, the

answer is here and you realize actually almost everywhere. In fact, the only places you can put this structuring element down. And not touch a 1, is over here. Right, because this is a 0 and this is a 0. And there's nothing over here, all right. And I can put it over here. Because this is a 0 and this is a 0. Everywhere else. So, for example, if I put that structuring element here, I touch that. If I put structuring element here, I touch that. Likewise, everywhere else in the picture, I touch a 1 and that's why that, it looks like that.

11 - Erosion

Erosion, at fact, in some sense, is easier to even think about. Again, I have a binary image B. And I've got a structuring element, okay? And now, my structuring element is this little tall column here of just ones, and its origin is in the middle. And the way erosion works is, I take my structuring element, and I run it over a picture. I, and, and what I'm going to do is, I'm going to take the AND of the ones i, in my structuring element with the, with the pixels below it in the image. And I take the AND. Well that means that all of them have to be a 1 for it to be a 1. So another way of saying this is, when I put my structuring element down, all of its 1s that are touched 1s in the underlying image, or I'm going to set the value to a 0. So before we take a look at what the result is, let's think about where can you put this thing down in this picture, such that it touches only 1s. And you realize, well, I could put it here with my origin there, right? I could put it here, with my origin there, and the same thing over to this column. And that's the only place, and that's why the result of this is this right there. And by the way, I guess I didn't mention the other one, this circle with a minus on it, that's referred to as erosion and the circle with a plus, that's the dilation nomenclature, okay?

12 - Effects of Disk Size on Erosion

Show you an example of erosion, and here we're eroding with a disk, a circular disk, with the origin in the middle, okay? So this is the original image, we saw this one before, and if we erode with a disk of radius 5, you can see that what's going to happen is, first of all of these lines that are thin, they all go away. Why do they go away? They're thinner, obviously than the size 5. Or actually it's radius 5. So they're thinner than size 10. Alright? And so, none of, there's no place you can put that disk down and have the pixel survive. Obviously, there's one Stretch here that's thick enough, and so that lead survives. So I make the radius bigger, this thing, they all go away. And if I made the radius even bigger yet, what happens is all these leads go away. And also you notice this square is shrinking by that amount because the only place. That you can fit that inside there is at this level, and so, it's going to pull away from the edges. So that's erosion. Now, as you saw, dilation fills holes and stuff, but it grows your picture. Erosion removes little things you might not want there, but it shrinks your picture. I don't really want to grow or shrink my cells. I just want to disconnect them. Not my cells, the cells in the picture. So I don't really want the things to get bigger or smaller. I just want to clean them up. Well, so that should make you think probably I could some eroding followed by some dilating, or some dilating followed by some eroding. And [LAUGH] you'd be exactly right. And those two operations are referred to as opening and closing. And, I say here the two most useful binary morphology operations. That is just one man's opinion. My humble, well okay, my not so humble opinion that that's probably the most important thing. And the only reason I say it's most important, is in many systems that I've worked on, a, down in the, in the guts, somewhere we're doing some opening and closing just to clean up our binary images a little bit.

13 - Opening

So, opening is a compound operation. So erosion and dilation are primitives. And it's a erosion followed by a dilation, okay? So that's what a opening is. And it says here, you can show that an opening of something of A by B is the union of all the translations that fit entirely within A. That's kind of complicated and wordy. It's much easier to see in a picture. So here's our binary thing A, and this is our structuring element B. And the opening is all the places, all the translations within A,

that you can fit B. And when you do that, you would get out this thing here. And you'll notice that it has broken the, the connection between those two triangles. And it's also rounded the edges. So, it's not surprising that it's broken them, because it's too small and the whole thing doesn't fit there. And the rounding is because of the shape with B. It says here, and, and you know, this is a, a slide. The opening is the area that we can paint when the brush has a footprint of B, and we're not allowed to let the brush get outside of the original image A. One of the other cool things is, if you take a look at this, if I now have this shape, I would open with B again. Well, it fits everywhere inside of there, so it wouldn't change, right? That is if I kept going, it would stay the same. And that's referred to as idempotent, or idempotent. I don't know how you pronounce it. I know how to spell it. It's an operation that when you repeat it, it makes no change. So, if you erode and then dilate, you get something. Well if you then erode and dilate again, you're not going to make any changes. Which is not the same as saying if you had erode, erode, erode, and dilate, dilate, dilate, that, that would be the same as erode and dilate. So you get that? So repeated openings, which are erode, dilate, erode, dilate, that doesn't buy you anything more. But sometimes, you want to erode either multiple times or with bigger shaped things, and then, and dilate again. So let me show you an example of using opening for like that cell colony thing we were talking about. It's a different picture, but it's the same idea. So, here we have this Thomas Moeslund, thank you very much, it, in Aalborg. So here's an original image, okay, of some cells that are growing. Here is a binary image, okay? And this is the opening using a 11 pixel disc, all right? And you see that it pulls out, you know, these cells pretty well. And when you come over here, you say okay, so it's got that one, it's got that one, it's got that one, it's got that one. It's got that one, it's got that one, it's got that one, it's got that one. And you know, clearly, maybe it misses this one, all right? Maybe it misses this one. So this tends to be common in sort of these kinds of things. You can find some operation which tend to pull out what you want, but it's not going to be perfect. All right.

14 - Closing

So that was opening. You erode, and then you dilate. Well, some of us come from the other side of the tracks. We like to dilate, and then erode. Same thing was eh, again using the same structuring element. So, using the same words from before, you can show that closing of A by B, all right? Is, it says it's the complement of the union of all translations of B that don't overlap A, well that's much better shown in a picture. Right? So now we take our paintbrush B and we run it on the outside of A, where we're not allowed to go touch, go inside of, of that and what results is the new area that is that, that, that's been painted. And one of the things that's interesting to see is, you see how this notch is smoothed out. But because of the convexity of this and the ve, I can actually move this around very precisely, right around that, so this is, these external corners are preserved. Okay, so it, it, it fills up sort of concave corners, but it, it remains, it keeps convex ones there. So, again, if I run that brush over and over and over again, I'm going to get the same painting that I got before, and again it's idempotent or idempotent, pick your favorite. Okay? The repeated operation has no effect. So, here's an example of doing closing. Take an image, threshold it, close it, and you get, but you might say it's a pretty reasonable outline of the boundary. You notice, it did catch some of the shadow well that's because your threshold pulled up some of the shadow. So again, you know, it's effective in a variety of ways of sort of cleaning things up. It's not like that it give you sort of perfect image processing.

15 - Opening Followed by Closing

So what I show you is, is the opening, and we typically use a disk or a square that can it can, can also smooth contours, but it'll typically break thin connections eliminate small islands, so small little spurious things. Also sometimes eliminate some sharp peaks. The closing can smooth contours where they, at the, concavities. It'll fuse narrow breaks, so things that were supposed to be connected but had a little hole in them. It'll, it'll do that. And it can, you know, sort of fill in these gulfs. So often you'll do one followed by the other, all right? So here's an example. This is the original image, okay? And it's, it's a fake image obviously, just to make the point. So here's our

original image. And you'll notice it has, you know, th, this cut out here, th, these rough edges here, this kind of noise here, these spurious things there. And the opening, all right, does exactly what you think. It'll eat away this stuff, this stuff, that stuff, and that stuff, all right? And also these little ridges up here. A closing, all right, well, what that's going to do is that'll fill in this hole, so that's gone there. Fill in that whole thing, so that's gone there, and fill in these things there. Okay, remember neither opening nor closing are going to grow or shrink things right? They're not erosions or dilations. One is a dilation followed by an erosion. Which one is that? That's a close and, the other one's an erosion, followed by a dilation, that's an open. Well, when you do an opening followed by a closing, okay, so you take this and then you close that, you get rid of this, you get rid of this, you get rid of that, you get the two underlying elements, okay? So, the idea is that was what you really wanted, all right? And these kinds of operations are what we use often. So to show you a real life example of that here's a fingerprint example. Here's the original image. You, here are the ridges that you want to find. There's a whole bunch of spots you don't want to find. There are some gaps that are not real, right there. And so what they do is they apply an opening, followed by a closing. And you get this really nice fingerprint image. Okay, so that's an example of, of doing it for real. Something I'm not going to talk about now, there's something called a hit or miss transform. It's basically how to find a particular shape. By the way Lynn Shapiro, I think it's in Shapiro and Stockman, or another book that Lynn was associated with ha, in their textbook, they have a whole bunch of stuff on morphology. So if you need to know more of the details you can go get that.

16 - Basic Morphological Algorithms

Given opens and closings dilations, erosions, this hit or miss transform that I was talking about that pulls out the shape. You can then do further things, okay? Here's a list of them. Boundary, region, extracting connected components, convex hull, thinning, skeletons, pruning. I'm just going to talk about two, and I'm really just going to talk about one in detail. Not even detail, one slide's worth. And then thinning, I'll just mention, okay? So sometimes, you like to extract the boundary. Well actually, that's very easy to do. What I can do is, I can take an object. If A plus B is the dilation, and A minus B is the erosion, I can get the boundary by just taking the object and subtracting out the eroded version of it. Here, they talk about B being a 3 by 3 square, it could be whatever size square. I'm going to take the binary pixels, and I'm going to remove any of the ones that survive the erosion process, all right? So you want to do that, and voila. This is what you get. And you realize, what's happening here of course, is here's our original binary image. And when I erode the, you know, the contour moves in by a certain amount. And if it's three by three by at least one and a half, two pixels, so what that'll do is, if I subtract what's inside the red from the whole thing, I end up with just this boundary. So, it's just a trivial operation right? I, I take the image, erode it, and then subtract those pixels, and by subtract, I mean remove those pixels from the original. And that's how you do boundaries. There's another operation called thinning. You'll, you'll notice this operator here. This is this hit or miss operator. What matters is, you take your original picture and you've got these different structuring elements. And essentially, you keep eating away a at your picture using that structure, that structuring elements, and you would go from this filled in picture like this, you would end up with, with something that's just like this. Or it's a one Ppxel wide thin band, all right? When you keep doing the thinning, and that can be used for finding skeletons. There used to be a lot of thought about using skeletons for a variety of shape representations when we're doing models, and people still use it and that's why I mention it. But I'm not going to talk, anymore about it. It's just thinning and thickening. And in fact, the thickening is just the inverse of that. So here you have this original thing, and then you're growing it out till you get out towards the, the boundary of some sort. Again, I'm not going to worry too much about the details.

17 - How Powerful is Morphology

So, the question is, how powerful is morphology? And like most questions in life, the answer is what I have to always tell my kids, kids, it depends. If you have nice, clean binary images, or even,

I shouldn't say that. If your binary images behave exactly the way the three or four binary images you looked at when you designed the system were behaving, then things will be fine. So clean might be the wrong word. If it's well, here it says almost clean. Let, let's just say that your binary images are, sort of, well-behaved. So here's an example and, again, this comes from Linda Shapiro, all right? Here is an original binary image, and, if I ask you to look at that quickly and ask you where are the faults in those gears, you would have to look for it at a long, look at it for a long time. But if I show you this picture, and I say, where are the faults in the gears? You would say, oh, I guess there's one here, and there's one there, and now when you look at it you see, yeah, there really is, there's a tooth missing from the gear. So the question is, how did they do that? So again, I'm not going to into too much of the detail, I'm just going to show you this set of images. It's all on one page, that's the sequence of operations, and i'll just describe them briefly, okay? So, here was their original picture that i showed you. Basically over here, they were looking for places that had a ring that would be essentially black like this, white out here. And remember I was telling you about don't care pixels, you know, don't care in there and, and, and of a particular shape, particular size, and the only pixels that lit up were these guys. Okay? All right? Well that's fine, and then you say okay, I'm going to take those and I'm just going to dilate them by something called a hole mask, which is just this basic shape. All right? So you realize what you end up with is just the holes filled up. Well, if I, or this plus this, I get that. So voila, I have filled the holes of my gear. Okay, great. Then what I can do is I can essentially do the same trick again of removing pixels that basically have with a structuring element like that. So I find every place where a structuring element like that fits, and that will rem, that will be able to pull out these, these middle parts. And then I can dilate the edge, and I end up with this thing that looks like that. Great. That's sort of a bound around the edge that's thicker than the edge. If I and these two things, so it's an and, I get just the teeth, okay? That's what's shown there. And, and also, you'll all notice we're starting to see there's that gap right there, because that's where the teeth are missing, all right? Well, what I can do is I can dilate that by something that has the size of the tip. And if I run that around here, I end up filling everything except where that gap is, both here and there. And then given those, I can do, in fact the result is just a, it's got this thing called defect queue. That's just this little round ball, basically the operation's exactly as it is here. It's b 7 right, minus b 9 here, dilated by the defect queue, or back with b 9 and I get this, okay? So, would I expect you to implement something like that? I, I don't know. The point is that with these relatively simple, low level mathematical operations, you can do a relatively sophisticated shape analysis if you have the particular shape that you, if you know the particular shape that you want to deal with, and the particular types of defect sizes, for example, you wanted to find. So, there are other operations that you can do. Here's a, just a list of things that you might computer. These are shape descriptors of binary blobs. And, like I said, this used to be a very important part of computer vision, especially in assembly or machine vision of inspecting 2D parts. And for those particular applications, it still is.

18 - End

Mathematical morphology has this huge science behind it that's a very abstract algebra because in some sense, it's all about set theory. Right? I mean, basically, that's what you're doing. You've got these binary sets, and then you do these operations on sets, and then there's gray level morphology. You won't find much of it in the computer vision literature anymore. When I say computer vision, I mean conferences that use the word computer vision in their title, like computer vision pattern recognition, European conference on computer vision, the international conference on computer vision. You won't find a lot of things, people doing work on morphology. But if you go to certain, pattern analysis kinds of things where they're doing syntactic operations, you'll still find mathematics of that, being done. However, even though it doesn't appear in this computer vision research, we use it all the time. Because we use it to clean up our results. In order to be able to get those great results that we can brag about to our our, our colleagues, or to get paid by industrial folks to build systems that actually work. So I think of think of this as not sexy, but very important. I had a whole bunch of jokes to follow that and I couldn't use any of them. So I'll just say that, it's kind like me. All right. Talk to you later.

9C-L1 3D perception

1 - Intro

Welcome to Computer Vision. Today were going to this might be the last one of the lectures on just sort of this collection of stuff, that I, maybe not be thought of core computer vision, but just stuff that I thought you guys should be exposed to one way or the other. Also want to give a shout out to a Kelsey Hawkins, who helped produce some of the material here. So that's opposed to all the other people that I stole things from, since Kelsey's actually my student and put this talk together, I thought I would, I would do that. But today I want to talk to you a little about, a little bit about depth sensing. I would say that this is tremendously driven by the fact that there's a lot of work recently in Computer Vision dealing for robotics. The re, recovery of depth previously had been a bi, back in the 60s, 70s, and 80s, had been a big focus in Computer Vision. Then things sort of moved away into more certain motion things, labeling stuff, processing a video to extract stuff, and, and of course processing all the images on the web. But in the last decade or so, with the growth of robotics, there's this goal to be able to visually extract information from the scene for a robot. And a lot of that information that's desired is about the geometry of what's out there. Where is there stuff, where is there not stuff? What things are on the table? How would I pick them up off the table? So that sort of given a resurgence to the interest in recovering geometry that's out there. And, of course, recovering geometry is a fundamentally difficult problem in Computer Vision. You know, here is a picture. You look at this, monocularly, and you see a surface with some geometry on there. There's a certain ambiguity, perhaps, as to what's in front and what's behind, but there's a question of, well, how do you get a single image to, to understand this kind of stuff? You, of course, have all these problems that intensity, image intensity changes as a function of the illumination. But here's an image of same scene lit from different colored light. And, of course, it provides a different image to be processed. And yet, the geometry is identical. We've talked about this before, challenges of scale. Here's an interesting picture. When you look at this table, you see a bunch of, white bowls, cups, et cetera, whatever. But if we actually were to sort of take a look at just in this patch right there, [LAUGH] clearly there's not a whole lot of interesting variation going on there for image processing to handle. And, you know, somehow it basically has to be able to, to somehow figure out from the basic extraction of this shape that there's this cylinder on the table. And that's generally a hard thing to do. We also have trouble that, color is not all that discriminative sometimes, of what's there. So there's sort of a, a similarity between the foreground, and the background. What we'd really like is we'd like some representa, something that gave us the shape directly. And so, that we would see something that looks like this. And then it won't surprise you that this is where we're headed. So here is a bunch of objects on a table. And, you know, it'd be nice if the system could just know that this was one surface, and that this was another surface, and these were the objects. And, what we're going to do is we're going to talk about 3D sensing.

2 - Passive 3D Sensing

There are a couple of ways of doing a 3D sensing some of which we've already seen. So Passive 3D sensing, what do we mean by that? That means you don't use any active illumination, that is you take the world as it's given to you and you somehow sense that. And maybe you exploit some known geometry of the properties or something about the scene. So have we actually done something that recovers geometry of the scene? Well of course we have, right? We've done stereo. In fact, those of you who are participating in the class for credit have actually matched stereo images. So here's an example of some some stereo rigs. These are sort of amateur ones. You know, I love this one, right? Build the wooden blocks, screw two cameras on, capture a picture. Here's something a little more sophisticated where you've got some high end video cameras mounted on a rigid bar that you can control the vergence angle. But of course if any of you gone to the movies lately, you've probably paid a lot of extra money, in order to be able to entertain your two eyes separately. Or more precisely, to see movies in 3D, stereo. And here's the types of, rigs that they

use. Some of you notice that's, Spielberg on the right, in the middle I think that's James Cameron. He was behind, the making of Avatar. Avatar was one of the first movies where 3D was fundamental to sort of the experience of the movie. And there have been a lot of 3D movies before, and, obviously one since, but in this non-film critic's view of the universe, it, it is, it was a seamless blending of sort of what you would think of as normal imagery, but done in 3D along with computer graphics. And so, the 3D experience wasn't despite, wasn't just sort of the experience of 3D was actually crafted as part of the movie. At least, this is my part view of the universe. And it's what it really made it, and, and the technology had come around quite a bit with polarized light, to really be able to give you a compelling 3D experience. There are other ways, also, of getting depth from a natural image. One, cool one, and we actually talked a little bit about this when we were, talking about focus pull, right? Is that the way a lens works, focal plane is actually in focus and then some other, and the, when things that are further away or in front are not in focus, so you can go from focus to depth, all right? In fact, here's this cute little animation of this scene, right, where you can see that they're varying the focal plane, and by varying the focal plane you can figure out the depth of different parts of the scene. All right, so that's all passive.

3 - Active 3D Sensing

But the other way of doing 3D sensing is what we refer to as active 3D sensing, where somehow, you're projecting something out into the environment, or sending some energy into the environment and reading that energy back, in order to recover depth. So we actually talked just a little bit about photometric stereo, when we talked about photometry, and the idea that you could recover shape if you had known light sources. So, even though these are thought of as regular images, you're, it's actually an active sensing technique, because you're lighting the system in a controlled way that you know about. But, when most people talk about active sensing, they talk about something that creates, sort of, special signals or pictures. So an obvious one is, just bounce something off a surface, and wait to see how long it takes to get back, all right? So there are different methods of doing that. This is a little guy. He has these little ultrasonic sensors, okay? If you go way back in time, back in the, way back machine, I don't know if it was the 70s or 80s, the Polaroid cameras started doing active sensing. They actually had a little acoustic sonar chip on them, that would send out what's called a chirp. And then, based upon how long the response got, how long the response took, it knew based upon the speed of sound, how far away the thing was, and it would focus the camera. One of the interesting effects of that is, because they made so many of those cameras, it made those sonar things very cheap, so people started putting them all over robots and things like that, and using sonar to detect depth. Much more sophisticated is what's called LIDAR, or a Laser Range finder. There's two of them here. One is on the side over here, one is stuck in here. These are on a PR2 robot. These actually generate a slice and they, they send a little beam out and they wait to see how long it comes back. Now of course, that's a lot faster than with sound, okay, but it, it's measurable, but you only get a slice of depth. So then what you actually do, is you put it on a little device that goes up and down like that, and so this little front area actually sweeps that plane and it produces a 3D depth image that way. So that's these are referred to as time of flight sensors.

4 - Structured Light

A more sort of traditional or i say fundamental computer vision method of doing active sensing is what's referred to as structured light. So here's this stereo figure that you might remember that we used before. The idea was we had two cameras, right. And there was some point out here and that point would project, oops, missed the line there, okay that's better. It would project to this point here, it would project to that point there. And because i knew which two points they were, i could figure out the triangulation, alright. Well, structured light says, instead of having a camera over here, I'm going to put a projector. And what this projector is going to do is, instead of having a sensor here that notices where a point is, and remember we had to go look for the point. What we're going to do is we're going to send the things out this way, right. So we're going to send some particular, think of just one point for a minute, right. And then in this camera over here, we're going

to see where does that point show up on the image plane, and then we can do the triangulation. But instead of doing points, what we actually do is we project stripes of some sort. In fact, sometimes this is called a light striping, right. And here it's shown with colors that makes it real easy. And the system of course, knows where each of the colors lies on it's plane. So then in the right image. In the right camera, the only camera, we see a picture that looks like this. So, what you can see here is like when these things are bent like this, okay. As it comes closer to you, right. So it comes over here, it moves to the left and so the line moves to the left there. In fact, you can see that a little more clearly in this image here, all right? So we have a set of hands and these are the light stripes that are projected on them. And actually this is observed in just the right-hand camera. And here what you're seeing is a 3D reconstruction of this surface based upon that light strike. So basically, by doing this light striping, by projecting these stripes out and then seeing them over here, i can recover a full 3D model. All right. So that's one form of projecting something out.

5 - Infared and the Kinect

Lately there's been a huge growth in using infrared structured light. All right, and infrared structured light, one of the nice things about it is you and I can't see it, so it's doing all this stuff out there but it's not, sort of interfering with our vision. And the main reason, that IR structured light has become so big is because of this puppy right here, the Microsoft Kinect, all right? Kudos to my friends and colleagues in in England and Cambridge who helped take the technology, and it was, the 3-D technology that I'll tell you about was actually developed by PrimeSense which was a Israel company. And then Microsoft leveraged it in a way to go from the depth to recovering skeleton, which is a whole other interesting piece of work that we won't talk about here. But if we did it in machine learning, since they used machine learning methods for doing it, we could talk more about doing that. But the Kinect is really, I mean it was $150, so all of a sudden you could put depth sensors on little experimental robots for very little money. So, how does the Kinect work? Well, it's not public, all right? So there are people who know exactly how the Kinect works, people from PrimeSense, etc., but it's not public. Lots and lots of this stuff is known. But the exact implementations are not known. The PrimeSense patents describe at least two ways. And I will tell you about both of those ways. Conventional wisdom is that one of the ways is actually used in the Kinect and one is not. I don't really care, they're both kind of interesting. So, the two methods, that they use, one is a very, very clever optics trick. In fact, it's so clever I don't totally understand exactly how it works, because I have to get more into detail. But essentially, it projects a pattern with two different kinds of lenses, and we'll talk more about it in a minute, that allows you to go directly from the image to the depth. We'll explain that. And then the much more standard way, which is just like light striping, is that you're essentially using something about the projected image from the projection of a particular pattern, and what it's observed like in the, in the other camera. So, you're about to see some drawings that look kind of funky. They don't look like, you know, the high quality drawings that Megan always insists that I use. But that's because these are actually taken directly from the patent and the patent applications. I think it's cool to just see those. So the key to the Kinect sorry, the key to PrimeSense focus method are these cylindrical lenses. So cylindrical lenses, if you take two of them and you have one that goes one way and I think on the next thing, you have one goes the other way, you end up with this funny situation where you have different focal lengths depending upon which way you move. So you have a, one focal length horizontally and a different focal length vertically, and of course, slightly different focal length as you were to sort of, rotate around. What this allows them to do is to project out speckles, and they, now, what happens when a dot is out of focus? Well, in a normal lens, right, it becomes a blurry circle. When you have one of these, what are called, astigmatic or asym, asymmetric lenses, it becomes an ellipse because the, the lenses are different in one direction than they are in another direction. And what's cool about the way they did this geometry, and this is the part that I'd have to work through the math more to understand, is that depending upon the depth, so here the hand is at one z and here is at another z, again these are from the patent. These ellipses, well they have different thickness, but more importantly, they have different orientations. So you can just take a look at the orientation in a little local area, of these little dots that have been smeared, and the

orientation tells you the depth. This is really, really cool. But I think it's so cool that they probably don't use it. because I think it requires some pretty precise lensing and some other things. I'm not, not really sure. But this is the, the one method of how the Kinect works. And if you go in their patent, you'll see figures that talk about like this, that you capture this test image and then you find the range base upon the pattern orientation. And then you build this 3D map. Although you notice this says based on offsets between test and reference pattern. So that's the method I'm about to tell you. Here's the one that just talks about the, the, orientation.

6 - More Standard Method

So what about the more standard method? The more standard method is actually just like light striping. So you recognize this figure, this is the light stripes where we project out the stripes, look at them from a camera, and then we see that the stripes bend depending upon the depth of the object. So, as we said before, you're still just doing stereo, okay? Except instead of two cameras, you're actually doing projection of one and coming out the other. But the thing that I want to remind you about stereo is if you know the relative geometry of the cameras, you know how they're, they're aligned and their, their offsets et cetera, right, their extrinsics and their intrinsics, right. Then for any given location in one image, there is an epipolar line, remember, in the other image along which to look. Now, this is also true for the projection method, right? So if I project a point out, right, at some location in one image, then depending upon the depth of the scene in my other image, yes, I'm over here, Meg, in the other image, where that will, point will be, will be along an epipolar line. And how far along the line it is is a function of the depth, so I only have to search in along a line if I know the calibration. Well, In my Kinect system, of course I know the calibration between the projector and the camera, but instead of projecting lines, what we're going to do is we're going to project out a speckle pattern. So instead of a regular projector, we're using a Kinect. Instead of line stripes, we're spraying out a pattern of, they call them pseudo-random speckles because it's important that the pattern be a little different everywhere so that I can do a, ready? Normalized correlation, all right. Instead of using a camera, I use my Kinect, because the Kinect actually, if you take a look at it, has a little projector in it and a camera. And the relationship between those are known. And then what the camera sees is a speckle pattern, but where the dots are offset as a function of the depth. So that's shown here. So here's a picture so it's looking flat, this is from one perspective. It's actually sort of faked in how it's doing it, but don't worry about it. If it was just a flat wall, but if I change the depth, so I put a little book on this stand, right, you see that's the difference between the depth, right? So on the left one I'm just projecting it on a flat wall and looking at it, and on the right, that's the offset that comes from the book, right? And, and by the way, what is this little pattern, this little thing here? That's a shadow, right, because the thing is offset a little bit, and I'm projecting here, there's a little area right here that I can see with the camera that can't be seen by the projector. That's what the black pattern is. In this particular case, the projection is coming from this side, and the camera is coming from that side. All right.

7 - Algorithm

So if you actually wanted to, build this, you have to implement an algorithm for doing it. Now, I tell you, I took this algorithm from, I think, their claim slides, they're on the web, that he took from somewhere else, etcetera. So I don't know for sure that this algorithm is the one they used, but it sounds pretty good. All right, it is, it's similar to what you would do. So, first thing you do is you project out these dots, and you just detect them. So you detect these speckles, and you label all of them as unknown. I don't know their depth. Now what you start doing is you randomly select a dot that's, that's labelled as unknown. And then you take this windowed search, all right, doing a normalized correlation, and along a scanline. That is, that really should say along an epipolar line. I know exactly where to look, okay, and I check for the best match. And by the way, I have to make sure that that best match is good enough. If that best match is not so good, I mark that dot as invalid and I start looking for new dots again. But, if my first match is really good, I start doing what's called region growing. I start looking around the, that neighborhood for matching other points

against the known pattern, okay? Oh yeah, by the way, what am I correlating? I know the pattern I'm projecting, so I've got my little local pattern of what's called pseudo random speckle points, and I just correlate them around. And so, pixels nearby are added to a queue. For each pixel, I initialized its offset looking for it from the one I just had. I search around a little bit, see if I find a good enough match. If so, I label it as matched with it's offset, its disparity. And then I look for its neighbors and I add them to the queue. And I stop doing that when that queue is empty, because I've finished using this anchor point. And I go back again and I, I, I, reinitial, I pick another point and I stop when all the points are labeled as with a known depth or marked as invalid. So what makes that work is that I have this, semi, this pseudo random pattern, so any little patch is not, can't be matched everywhere but is matched or just in one place well along that epipolar line. I know those patterns and so I can just drive that search very quickly.

8 - And Now a Video

There's a cool little video on the, on YouTube. Let me just show that now and it explains how this works. I'm actually going to cut out the section when it starts talking about rabbits, but that's because you guys already understand how stereo works. Microsoft's Kinect has a depth censor that can tell how far away things are. In this stream, brighter colors are closer and darker ones are farther away. This video will explain how it does this which is pretty clever and not how many people initially thought it worked. By the way, these things are relatively easy to implement. You can implement these things too. Just hook up your Kinect. You can use, just write a little program, or even Matlab. You can do it directly and display it. And it's kind of cool to experiment with. The thing that I like about this video that you can see well is, he, he sets it up so you can look at this in a dark room using a IR-sensitive camera, which, by the way, most digital cameras are IR sensitive. So if you do this, you probably look at this on the video feed coming off of your digital camera while you've illuminated things with the Kinect and you'll see it. Now if you take a night vision camera and look at the projection from the kinect, you won't see a grid of numbers, but rather a random speckle pattern. This accomplishes the same goal, though, because no group of specks looks like any other, so the Kinect automatically knows the angle of every group and can therefore triangulate distances just like shown before. The downside of this technology is that it can only work indoors because sunlight would wash out the speckle pattern, and similarly, multiple Kinects would confuse each other.

9 - Projected IR vs Natural Light Stereo

Alright, so we've looked at projected IR versus natural light. What are the advantages of this projected light, or projected IR? Well, yes it works in low light. You're not counting on having external light. You're not counting on having texture that you can deal with since I'm projecting this texture, and its okay if I've got repeated things because I'm, I'm it's not actually repeated in a speckle pattern. Remember we had this problem with picket fences in stereo? How all the pickets look alike, to know which picket it is, is a challenge. Well, if you painted all the pickets different colors or different speckles, that's not a problem, and that's what the connect does. And in particular, you can tailor your algorithms to the actual pattern that you're producing and engineer the thing to work really well. What are the advantages of natural light? Well it's natural light. So it works outdoors, it works in sunlight, it works, you know, it just works whenever it's, the lights are turned on. Doesn't work in the dark, that's okay, neither do I, but the idea is that, you know, you're just basically using the scene. The other thing is that you don't have to worry about the resolution of your projection, you've got the whatever at your high-resolution sensors. It's a lot easier to build high-resolution sensors than high-resolution projectors. Both of them have some difficulties, you know, they, they if you have very shiny surfaces, all right, or specular surfaces like a mirror, right, so your projection is going to go. And, you know, clearly, if I'm doing stereo and I'm a robot, and I'm looking at a mirror, I see the thing that's up there and I think oh, I could just reach down here and get that light that's on this, oh no, okay? It actually, the, it doesn't see the table. And my

projector, well, it projects down so it'll also be thinking about the ceiling, all right? So, doesn't work in any of those cases.

10 - Depth Images

Just want to end a little bit talking about, if you've got this way of recovering depth, how do you represent depth? That is, how do you think about the representation of it? We only actually talked about disparity. We talked a little bit about depth images but not too much. But I want to talk about just two quick methods here. One is a depth image and then there's a slightly more, I use the word nuanced, a little more recent notion of representing depth, namely point clouds. So depth images as shown here this was another one of those stereo figures from the Middlebury data stereo dataset and this is a ground truth known depth image, right? So you see these rings here? Right, well, you see that this one's darker, and brighter, and brighter. They're at different depths. Likewise, this picture here is this picture there. It's brighter than this mannequin back there because it's closer, and if we had more grayscale resolution, you would see this would be darker than, than, than this area because the nose is further back from the cheek. So depth images have some nice representations. First of all, I've got depth everywhere, right? I can just go to every pixel and say what's your depth? So I've got a full dense depth representation. Something that's a little more subtle, is that a depth image not only tell you it's two meters from here to that table, it also tells me that between here and that point is free space. So it doesn't just represent the surface, it also represents, by default, if you will, that there is no stuff between here and there, right? So that putting the point down there is also declaring that the ray from my eyeball to that point is free. So it's a representation of both the surface and of free space. People don't make nearly enough use of that fact but it's true and important. The other thing is often we need to think about depth discontinuities. Well, depth discontinuities are just edges in the depth image, where we have lots of processing for dealing with images. There are, of course, disadvantages. One is that it's very viewpoint dependent, right? I've got this scene geometry but my dept image is based upon where my camera is. So if I move over here, I've got a different depth image, and combining them is difficult because they're in the reference frame of my image. They're not in sort of some natural reference frame that talks about the object. So it doesn't capture the geometry quite so much and the other thing is, if I want to think about reasoning about the geometry, I kind of need to know where the camera is, with respect to this, in order for me to talk about where the geometry is.

11 - Point Clouds

So what's a different representation? Well it's nothing more recent newer approach, it's what's referred to as point clouds. And point clouds are essentially, you take all of your depth pixels that you hit. And you say, aha that's a little, we used to use the word voxel, that's a little voxel of stuff. And I'm just going to hang it out there in space and so I would get this constellation of points that's just a cloud. Okay, now I actually can't see inside surfaces, so it's maybe a cloud that I can only see the outside of. But if I had like another camera and I knew how those cameras were related. I could take the points that it saw and stick them out in the same cloud area. And eventually, I would have this big point cloud of depth points and it would be a lot easier to reason about the geometry of those. Now of course, when we do this that point cloud sort of doesn't know how it got created, right? So did I look at it from here or did I look at it from there? So, I don't have that free space information anymore. So point clouds of course, have their advantages and disadvantages. The biggest one is probably that it's viewpoint independent, right? So, because it's viewpoint independent, I can merge different point clouds as long as they're in the same coordinate system. And I can actually reason more about the geometry. The disadvantages include, so for example it's not, it might not be very dense, right? And in depth image, I've got depth everywhere. Point clouds, whatever points I got, I got. And if I want to know what's the depth going this way, I might not know the answer, I just know about these set of points. And so fundamentally, you've also lost this free space stuff, okay? Because you don't know how we, so I have a point right here, I don't know if it was viewed from this way or from this way. If it was viewed this way, I know that all these

pixels along this ray are free from here to over there, I don't have that representation in point clouds. The other thing is as I get further away essentially my cloud, each individual pixel that I see is taking up a bigger and bigger voxel. So the point cloud density, sorry, the point cloud resolution, you know, how big is a point cloud, point. The resolution is determined by how far way from the depth sensor I observe that point, so nearby ones are tight little points, further back, they're bigger voxels. But I didn't tell you yet about the biggest advantage, the biggest advantage about point clouds is the point cloud library. There's all this code out there and it continues to grow, called PCL for the manipulation of point clouds. And so, that's code you don't have to write, all right? So, it allows you to operate on depth, just not worrying about the details of the bookkeeping about these depth representations. So that's, that's really a huge element. And one might even say if it wasn't for the PCL, the Point Cloud Library, point clouds would not be nearly as popular as they are. Eh, here's a, [COUGH] here's a pointer to, what's the website? Pointclouds.org, all right? Which tells you something, right? .org, this is open source. So it continues to, to grow, and continues to improve.

12 - Point Clouds and Surfaces

Couple other little things about point clouds. One is, wi, the point clouds, of course, are sampled from the surface that this depth sensor hits. So you're, what you're getting is, you're getting a sampling of the surfaces, which means there are no volumes. You have to infer a volume, right? So even if there's a nice sphere right here, you're going to get points along its surface, and maybe if you move around, you'll get more points from the surface, but it's up to fut, further processing to decide, oh, that's the spherical volume. So there's no explicit representation of volumes. There is an interesting element to it, and that has to do with surface normals. So, I assume you all know what a surface normal is, you've got Geometry. So the idea is you've got some underlying surface, and at any given point, you can fit the tangent plane, and the normal to that plane, that's referred to as the surface normal. You can think of it as a first order local approximation to the surface. Point clouds actually have surface normals. And you might ask how is that possible. How is that possible? Well, because it actually, the sensor, it does a variety of little processing to try to find little local points nearby and fit that tangent plane in order to estimate the surface normal, and sometimes you, that's useful. I will tell you that the surface normal representation is less robust than the actual location of the point itself. So, this cute little needle diagram, in order to fit them, you have to essentially take a little Gaussian around the, the points that are there, fit a little tangent plane to those Gaussian weighted points, and then that gives you, an estimate of service normal. You know, you have to figure out how big a Gaussian do you want to use, is a little bit like how big a Gaussian do we use when we take an image gradient, remember that? We used to have to smooth out the image to get rid of the noise, but the more you smooth it, the more you've spread out the gradient? Same thing is true here. The more, the more points you consider, the more smoothly the surface normal will vary.

13 - Point Feature Histograms and Software

Suppose, I've got a depth scene here and I'd like to match it to some model or some previously observed depth scene. How might I do that? Well, what I need to do, is I need to find some patches, some locations on my object and match them to some locations on my object. And you know it should be somewhat invariant to how I look at it. All right, so I'm looking over here, looking over there. I want to find the same thing, the object might be rotated. Things might, we've seen this movie before. Remember we we're matching pictures as we had two pictures that were rotated, translated, scaled, and we wanted to align them. Do you remember SIF? SIF was a way of saying, in Harris points, remember they were Harris Stevens points, and nobody ever remembered Stevens. We had ways of finding points, and then generating a descriptor and that descriptor had to be, sort of discriminative and yet, robust. That is, we wanted them to be relatively unique but robust, so that when we found a description here and we matched it with the description there, it would most likely be the right one. And then we improve that, we use ransack in order to find the right sets of matches. Well, we can do exactly the same thing with depth. You know what's drawn here is that

I've got this interest point and I've got this sort of neighborhood of these five points around it or I might look at a, at a volume of these. And what I'm going to do it, I might find the relationship between these points and say the little local normals associated with the point, or something to do with the geometry. The idea is whatever it is that I'm measuring is not a function of how I looked at the points. So in fact, the way we often do these is, we typically will build a, we'll, we'll take these points and we'll build a, a description and we'll put them into a histogram. Okay, so maybe we just quantized these different orientations into, you know, five different levels, so we have, you know, five, level thing, and then we might have distances. So, in fact, one example, they used a five by five by five histogram that gave you 125 different values, okay? Remember in SIF I think we had 128 different values. So here we have 125 different values. And then for every patch, we compute these and we can compare them against the patches in our model, find the putative matches, and then do something like ransack to, to do the matching. There's a version called the fast point feature histogram which is a way of doing this in 3D, and again, I think is probably influenced directly in the Point Cloud Library. So there are a couple of places where the Point Cloud Library, I, I already pointed you at 'PointClouds.org'. For many of you in the robotics universe there's something called ROS, the Robot Operating System originally out of Willow Garage and then it's, it's robotic software open found, it's got all those letters in it, it's an open source ongoing development system for doing robotics and the Point Cloud Library and by the way, Open CV, Computer Vision Library are all part of that universe. And that's really helped make it easy for people to do that development. And there are drivers, by the way, in that for using the Kinect or PrimeSense sensors as well, so you can go get your point clouds that way. So here's a picture that I showed you before from pointclouds.org. And in fact, here it says, ransack cylinder segmentation. So basically, they ransacked to find the different planes and to also find these different cylinders. You know how to do that already, so you could do this and it basically gives you a direct interpretation of the geometry

14 - End

So that ends our, our discussion on direct depth sensing. Probably six, seven years ago, I wouldn't have even mentioned it anymore because light striping was sort of relegated to sort of assembly line machine vision. But with the advent of the Kinect and other types of such sensors, and really, robotics putting geometry back as a first class computer vision problem. Depth sensing has become a big deal again. And since many of the manipulations are similar to what we do with images, it made sense to consider it within computer vision.

10A-L1 The retina

1 - Intro

Good morning, or whatever time it is by you. Welcome back to Computer Vision. This our last, general unit on for Computer Vision. And it's actually we're going to talk about what's probably the most remarkable vision system in the world. Namely the human vision system. And that's just not me being sort of arrogant about us, us as a species. But really, we really do have a remarkably well developed visual system. We'll talk about sort of how much of the brain that makes use of it especially at high level kinds of structures.

2 - Human Brain Overall View

So, the brain is generally what we're talking about and here you have a picture, the overall view of the human brain. A friend of mine, Jeremy Wolf, always said make sure you draw in the nose. So, okay, so your nose is over there, your eyeballs here and then there are all these various structures, the cerebellum in the back here. Behind your head is where a lot of rapid motor control kinds of things take effect, bunch a whole bunch of structures, all of this crinkly stuff up here is the cortex

and we'll talk about that a little bit later. By the way these little things sticking out right here called the olfactory bulbs, those are the things that actually go into your nasal cavity that does sense of smell, that's actually a part of your brain that sticks out into your into your nose. And another part of your brain that sticks out is actually your retina and we'll talk about that in a minute. The parts of the brain I want to talk about today are, its, this lesson and next, of course, are the parts that are involved with the human vision system. And those are poorly drawn here on this schematic. It's just ugly, but it's actually much clearer. So, again, here we your two eyes. Okay, okay now your nose is up here. And, you notice that out, coming out of each eye is what's referred to as the, an optic nerve, and they sort of cross in this funny way at what's referred to as the apt, the optic chiasm. They then make, a, a way stop. They make a stop here, down in here in, lateral geniculate and then the lateral ge, which we'll talk about, and the lateral geniculate has projections up into something called the superior colliculus. And the, most of it, a lot of it goes to what's called the visual cortex, and we're going to talk about that. By the way, we describe these things as if it's a flow of information going up. And that's partially because as computationalists, we like to say, well, first we do some low-level processing at the, at the, the detectors and the retina and then it goes up to these weigh stations where various kinds of things happen, and then up for higher level processing. The reality is somewhere between 80 and 95% of the projections into, for example, the lateral geniculate, are actually from the cortex back down. So the engineers amongst you will say, oh, that's gotta be feedback of some sort. Well indeed it probably is, and it's still stuff that we're just slowly getting a handle on. So take what I'm about to say with a grain of salt. In fact, well, take everything I ever say with a grain of salt. But in particular, I'm only going to give you a relatively simplistic view of what's going on in the vision system. It's also a slightly dated view because while I'm try to be pretty current about certain computer vision things, there's a massive, massive field of neurophysiology and human vision. So I'm just going to give you some of the highlights and it'll give you a general overview and I encourage you to, to read more as we go forward. One thing to say also before we continue forward is sort of the anatomy and physiology of the brain really reflect the computational complexity of vision. So to give you an example, you have auditory nerves that come from your, cochlea that go into your brain and those nerves, there's two of them, one for each ear, and those nerves have somewhere around 30,000 nerve fibers in, in that bundle, okay? The optic nerve, each one coming out of here, that has somewhere around a million fibers, okay, 30,000 for hearing, a million for vision from each. Likewise, and we'll talk more about this later. We are going to talk about the visual cortex, this area back here. So far, and that you can, again, you know, plus or minus, somewhere around 30% of your cortex, right? The cortex being the most recent part of the brain in terms evolutionary structures. We think of it as being sort of the human helmet. Somewhere around 30% of the visual, of the cortex is seen to have something to do with vision, as opposed to, say, 3% for hearing and 8% for all somatic sensory touch. So, again, the complexity of vision is not only showing up when you look at your problem sets and how, and you say, how could this possibly work, but it is also reflected in the detail structure of the human anatomy.

3 - Visual Field

So let's talk a little bit about vision in your visual field. So as you know, you've got two eyes, well you started out with two eyes for most of us. Each eye is about 160 degrees and they overlap at about 120 degrees, which means, if you do the math, it means you have about 200 degrees total field of view. Different animals are different, by the way. Some animals, the ones that are prey, and supposed predator. Their eyes are back here. So they have a wider field of view, and that is they can see more behind, further behind them, which you might want to do if you're worried about a predator. But of course, it means you have less stereo vision. But remember, we were talking about nerves coming out and crossing this area called the optic chiasm? Let's take a look at that a little more carefully. The human vision system does, well you're probably quite familiar with your motor control system, right? That the left part of your brain controls the right arm, and the right part of your brain controls the left. And if you have a stroke on the left side of your of your brain you'll, you'll lose some motor function also some sensory function. Well, you probably didnt stop to think about that the eyes have a tricky problem, right? So the stuff that I see over here is seen in this eye

and in this eye. And the stuff I see over here is seen in this eye and in that eye. How does the brain deal with that? Well, the way it does it is, it actually splits the eyeball in half, in terms of where the projections come from. So here we see the right visual field, which is looked at by this left side of the eye. Which is the temporal side of the left eye and the nasal side of the right eye. Tho, both of those come in through the optic chiasm going to the left side of your brain. So the left side of your brain is looking at this stuff over here on the right in both of the eyes, okay? And likewise, the right side of my right eye and the right side of my left eye, which is looking at the stuff on the left-hand side, is projected to the right-hand side of my brain. And so just the same way that your sensation on your right arm is in your left brain, the sensation in your right visual field is at your left brain. Also in this other view of the sort of the projection of the optic system into the brain, you can see these other structures, right? So we get here to the lateral geniculate, again on both sides, project into the superior colliculus. And here you can see that there are these projections out to this whole area of visual cortex which is in the back. And we'll look at that. One of the implications of this right half, left half, the right part of your brain looking at the left half, vice versa, is if, if you have a some sort of the degeneracy or stroke or something, and let's say you were to lose some function on one side of your brain you'd experience what's referred to as here it's written, hemifield neglect. 'Kay, and what that means is that, so you've had a damage, let's say, to the left side of your brain, the whole right side area would just sort of not be there. And what's interesting, and, and we'll see an example of this a minute when we talk about blind spots and stuff, you don't feel like there's a hole there. You just feel like this is my field of view. I don't actually know what's over there. Okay, so it's not like right now I feel there's a hole back here. Despite what my wife says, there's no hole in the back of my head, I don't think. But I just don't, I just know that I can't see what's back there. Well what happens is if your brain sort of starts not knowing what's over here, it just kind of, you know, it just doesn't worry about it. You know and in fact, what will happen is that it'll fill in some stuff for a little bit, kind of like you do with the rest of your visual field. We'll talk about that in a minute too. But, but I don't want you to get a sense that you would feel like there was a great big black hole sitting over there.

4 - The Human Eye

All right, so let's start sort of climbing up the visual system, starting at the bottom, which of course is the eye. And here is a sort of a schematic picture of the eye. And you know, lights coming in this way, and there are a variety of structures. Let's see, a couple that we'll, we will mention. So you have this lens here. Lens changes, what's in focus. And by the way, you have what might be thought of a spherical retina. And so, the, in some sense, spherical retinas are a lot easier to deal with planar retinas. We make, we put planar retinas in cameras, things like that, spherical means all the rays are sort of the same, which is kind of nice. The other thing to notice on these lenses is, that they're attached by these fibers right here. And those fibers pull on the lens, and they cause the lens to, to deform, which changes the focus, and that's how you focus near and far. Well that's how you focus near and far. The way I focus near and far is I put on reading classes. Why is that? Because I'm old. Well, I'm not that old, but what happens is my eyeballs think I'm old. In fact, sometime, right around my 40th birthday, I looked at my watch, and I was trying to wonder how my watch knew it was my 40th birthday because it decided to go out of focus, just that day. But what happens is, the ability of your, of those fibers to deform that lens, that lens, it hardens up a bit. And, and you lose the ability to deform the lens as much and you can't do near focus. And that's why you see those of us with less hair carrying around reading glasses. So, the lens focuses the light coming in, goes through this center called the pupil. You have a certain amount of adjustment of the size of the pupil to, in terms of the amount of light that's in there. We'll, we'll talk about the dynamic range in a minute. And then what happens is the light passes through a, a clear liquid, which keeps the eye in its shape, and it projects onto this back area called the retina. Okay. And, one of the things you'll also notice in here is that there's this optic nerve, there's also a bunch of blood vessels that come in and feed the retina. And we'll talk, in a little bit, about how come it is that it comes all the way up through here, so that it actually sees the, the middle here that is, it's not just in the back. But when you look in there, what you would see is a funny little area. It's, referred to as the optic disc or the

blind spot. And in fact, if you've ever been to an ophthalmologist and he looks in your eye, what he sees is something like this. All right? That's assuming everything's healthy. So there are all these blood vessels that are nurturing the, the retina, and then there's this area in the middle where the blood vessels and, and nerves are coming together and that's your blind spot.

5 - Blind Spot

So how many people would like to find their blind spot, raise your hand. You can't, it's blind, no. We're going to do that. So it's actually really, really easy. Couple of ways we can do this. So, on my screen right now, I have a picture of a, a plus and a, and a dot. Your blind spot is always on the outside of your visual field. So what I'm going to do is, I'm going to look for the blind spot in my left eye. So it's going to be outside. So what I'm going to do is, I'm going to, whoa! It happened right now. So right now I'm looking at that dot on the right, and to the left of it is a cross. And if I move here, there's the cross. There's the cross, but right here, the cross is gone. I think that's at 12 deg, I can't remember, I think it's 12 degrees out, right? It's just not there. It's just plain black. Okay if I want to do it with my right eye, well what I'll do is I'll look at the cross, and I'll move until I oh! There it goes. The circle is gone. If you are doing this, if you rotate your head like that, the cross comes back because the blind spot is located off to one side and as I rotate, the blind spot is now up here and the, and the, the cross pops back. In class, normally what I do is I make a bunch of pieces of paper and I hand this out right here. Okay, and I ask people to take their left eye, so, so, for those of you not paying attention, so we have a dot, and over here we have a bar with a cross hash pattern in the middle of it. And what I do is, I tell people to move that dot til something seems to happen to that bar. All right, so back here I see the bar with the cross hash, I move in, and then something happens, the cross hash disappears into just a line.

6 - Inverted Retina

All right. So here we are back at our eye again and let's start talking about sort of the most important part. This is the part of your eye, the retina. It's the part of your brain that is in your eye. Now here is a picture schematic of a retina, okay? And first you'll notice there are these receptors called rods and cones, and we'll talk about those in a minute. And then those receptors connect to these things called horizontal bipolar cells, which take the chemical output of the receptors and eventually turn them into stimulation for these ganglion cells, which are the first proper nerves, these are the nerves that actually produce firing spikes that we're used to for nerve cells. But when you look at this, you should look at that curvature, and it should bother you a little bit, right? So here's the picture of my eye, okay. But you see of course my thing is, is bent this way. And this thing is sort of bent that way. So actually what I need to do is, and I'm really proud of this, is we need to rotate that picture. So this curvature here is like this curvature there. At which point you realize what comes up this way, the light. All right? The human system has what's referred to as an inverted retina, okay? And that is the light comes in, and passes through, all this gunk. It passes through, there are blood vessels in here, there's nerves, it passes right through them. Then it hits your photo receptors, okay, and then any stray light keeps going, sort of gets absorbed. This background layer, this is called the pigment epithelium. This is actually a very black pigment in the back of your eye. So, you might ask, why would you build an inverted retina? Why would you build an inverted retina? Nobody asked me, but here's some, some speculation. There are a couple of things. The, the reason that, that I like best is. What happens is, is that the light goes through here and anyone that happens to hit the photoreceptors, right there, gets absorbed. And then after that, it hits this pigment epithelium. Okay, and what that means is the pigment epithelium absorbs any stray light, so it eliminates scatter within the, within the eye. And that gives you a very sharp resolution. It reduces your ability to see at night a little bit. We'll talk about the sensitivity of rods and cones in a little bit. But it gives you this, this idea that you know, first the, the light, whatever light gets absorbed by the detectors goes into processing and everything else is absorbed and removed. It doesn't cause any disturbance. It does mean that there's a funny sort of thing that goes on. These structures produce a very, they're clear because they're made out of you know, gelatinous stuff that

is biological tissue. But of course it causes little shadows, right. So if you go right past the blood vessel, it'd be actually a little bit of a shadow. So you might ask, why don't we see these shadows. Why don't we see these shadows? The human eyeball is remarkable. Anything that's perfectly stabilized on your eye, the brain just removes. Okay? So that shadow stays exactly in the same place on the retina. The brain just removes it. In fact, there are some cool tricks you can do by building little sta, stabilization systems that will keep something projected exactly the same place on the eye and eventually you just don't see it. So, anyway, so that's the inverted retina of the eye

7 - Rods and Cones

Now, if you were to take a look a little more carefully at these photoreceptors, and actually look at them for real, you would see two types. You would see rods and cones. Now, we've, we've looked at this picture before when we're talking about color, but now we want to talk about the anatomy. So, the rods are these little, skinny structures here, and there are something like, it says 120 million rods, plus or minus, within your retina. They are very, very light sensitive, about 1000 times more light sensitive than the cones, if given time to adapt. We'll see that in a minute. And it can sort of help you discriminate between, of different illumination levels at very low overall illumination. The other structures here are the cones, and we've actually seen these before. The cones are the things that allow you to discriminate color. We talked about them when we talked about the short, medium and long wavelength cones. The cones are also the ones that are packed within your, what are called your fovea. That is that middle area that has the very high resolution. We'll talk about the impact of that in a minute. And it's the cones that give you sort of what we tend to think of as the high quality vision, and the rods sort of provide everything else. Now by the way, even if you didn't have an electron microscope, and even if you hadn't sort of peeled apart peoples' retinas. It sounds kind of gross. You would know that there are at least a two kinds of systems going on in the eye. And one of the ways that this was discovered is what's referred to as dark adaptation experiments, okay. So in dark adaptation, here's what we do. We, we put you in a dark room for months at a time, no. You put somebody in a dark room, and you, let's say after one minute, you say, how bright a light do I have to have a light to be before you can tell that it's on. Okay, if you do that? And then later we do it all over again and we do it for two minutes, and then we do it for three minutes. Right, so we track to see how much light you need in order to detect that there's some illumination. Now we've all had this experience before right? You walk in from a bright day light into a dark room and it takes a little while. We say for your eyes to adjust. Well, that adjustment is not your pupil opening and closing. That's a rapid response. What's happening is, is that the pho-, is that the photo receptors in your eye are adjusting to the amount of light. All right? And if I were to track that, I would see this kind of a curve. Okay, I would see a curve that starts going down like this, and starts to look like it's going to flatten out. And then somewhere, this says about eight minutes, varies on person to person, somewhere around there, all of a sudden it starts to drop down again. Okay. What's happened is your rods are now kicking in. Your rods are now coming into play. And you can go way much, much, much lower threshold. All right. And so, you know, when, when you have a single system you never get psychfisicers that have sharp breaks like that, right. And basically what you're seeing here is there are two curves. There's a curve here for cones. And a curve here for rods. And this was where they intersect, all right? And that's the sort of well-known phenomena which implied that there was multiple systems within the retina in terms of light sensitivity.

8 - Photoreceptors

So let's take a little bit closer look at, rods and cones. They have a certain similar structure. They have this, we'll call it an outer part, but remember, the, the pigment epithelium is up here and the light comes in that way, but this is called the outer part because it's this inverted retina thing. All right, and this inverted part is where you have these disc shaped elements, that react to light, and they react to different wavelengths of light, we'll talk about the different wavelengths in a minute, and those wavelengths and those reactions set up chemical reactions in the, in the cell. And of

course it is still a cell, so it's got these inner elements, these organelles and a cell nucleus, and then what's important is it has this synaptic body where it changes the chemical properties at the synapses depending on the light. So let's take a look just at the rod for a minute. The rod has this chemical, these disks made out of what's called rhodopsin, all right? And here, this is an electron micrograph of what it actually looks like. And this outer segment is made of up these plates, rhodopsin plates, and what happens is that the, the rhodopsin is a chemical that when it gets hit by photons, it splits into two chemicals, I don't remember both of their names, but one of them is actually associated with vitamin A. Remember your mother always told you you had to eat your carrots in order to see well? Where does that come from? Well, the vitamin A, in, in the carrots is involved in the chemical that's involved in, in, rods being able to see light. So I'm presuming that that's why your mother told you that. Just ask her, all right? To stress the sort of chemical nature of what's going on, there's a slide here that just shows the same rod, and there's a sort of this change in these sodium ions. And, oh here I, I wrote it down in, in notes that, when the rhodopsin splits, one part's called opsin, and the other part of this vitamin A, bit. And, when light comes in, this change is sometimes referred to as bleaching the, the rods. All right, so they, they, they start to change more. And that, and how bleached they are is sort of how adapted they are. Right? So when they're not bleached at all a very small amount of light, now coming in will change it, after you've bleached them a bit. Now you need a different, you're now operating at a different, light level. In fact, there were studies that were done that were showing that a single photon, so those of you who are familiar with the physics of light. A single photon will, can make a change in the cell that this system can detect, which is just remarkable, right? The ability of the system to detect small changes of illumination. And, you can sort of ignore this diagram over here. It's just showing you that it's understood at a pretty detailed level how the light makes a change in the chemistry, and that the chemistry then changes what goes on in these synapses. Now, we said before that each of these photodetectors respond differentially To wavelengths of light. And we showed you short, medium and long. The one we didn't show you before was the one that includes rhodopsin. All right, so here's the blue or the short pigment, the green and the red, which we've looked at before, and here is what the, the rod responds to. You see the rod is relatively broad spectrum, okay. Which is good because it means that if there's any of this kind of wave lengths of light coming in at night, the rods will be able to see them, even though they're very, very low amount of illumination that they're sensitive to. We're not going to talk more about cones and the tristimulus theory, color theory of color because we did that last time. But these are the four operating spectral responses that each of the photo detectors give you

9 - Retina

We did also look before about how the cones are not evenly distributed, right? There are many, many more red and green ones than there are blue ones, remember we were talking about that? And we see that most of your color vision is based just upon the ratio between the red and the green, and the blue is just there to give you a little bit of, sort of sensitivity to blue, but you don't really have high resolution in the blue channel. If you were to take a look in your eye, well, not your eye because this has to be a dead eye. So, some other eye, or your eye after you're dead. If you want to donate your eye when you're done just send Megan a postcard. If you peal the retina out and you stick it under electron micrograph, if you take a look at that, you'd see two, you'd see that as you moved along the eye it was not the same everywhere. In the middle, which is the area referred to as a 'fovea', about one and a half, degrees, you see this magnificent hexagonal packing of photo receptors, okay? So, hexagonal pack, so i, if you will, the pixels in the human eye are not on square grid, but they are on a grid. They're on a hexagonally packed grid, and the regularity of this is pretty remarkable, right? I mean, it, it, it gives you a very good sampling, a ve, very dense and regular sampling of the underlying visual field. Which is one of the reasons that you have such good high resolution up here and really lousy resolution out here. You see, so I'm looking straight at Megan, wake up. Okay, good. So I'm looking at, I can't really tell how many fingers I'm holding up. I mean, I happen to know, of course, because they're my fingers. So I can tell there's seven of them, but or, whatever. But I can't, if you do that yourself just do it this way. If you want to have a

partner, look, look them straight in the eye and have them bring their, their fingers in. You'll see their fingers are there. You can have them wiggle them, but if you, if they ask you how many fingers are they holding up, you cant really tell until it gets pretty close. And yet, you feel like you got all this high resolution vision out here. That's an illusion that your brain is creating for you, out here in your periphery you just have good sensitivity in motion, in fact, wiggle your fingers you'll, you'll see them right away. Why might you want to do that? Well, maybe you want to saccade your eyes over there. So, the system is very good at detecting motion in the periphery, and controlling the eye to look at it and foveating on what's there. But you actually only have high resolution in this very small area. In the periphery, as shown here, you have a few large cones. So, the cones are actually big. And they're giving you overall color information. And then you have this high density, again, of rods so, with low levels of illumination, you can still see out really well. There, let's take a look at this picture. So this picture is degrees from center, so temporal is off to the side, nasal is towards the nose. And the first thing you'll see is that you've got all these cones that are mostly in the middle, in the fovea. That's exactly what we said. Notice, the same way we have rods out here in the periphery, notice you don't have any rods in your fovea. Over here's your blind spot, right? You don't have anything there. That's why it's blind. So, what would one of the implications be of having no cone, no rods in your fovea? It would mean that you don't have very good low light perception in your fovea. And here's another experiment you can do. Some night, go outside when there's a bunch of stars, some that are really bright, some that are less bright. See if you can find a star that's less bright that stays there 'til you look at it, and when you look at it directly, it disappears. And that's because your fovea can't see the low light that your rods do.

10 - Dynamic Range

So now let's get back to our retina. So here we are. So we, we've sort of talked, you know, very briefly about the photo detectors. Now we're going to talk about the ganglion cells. One of the things about the ganglion cells is that they take information from these things called these horizontal and bipolar cells and the horizontal cells just as you might think they take information across a local area and bring them together to the ganglion cells. So you might ask why would you do that. Why would you do that? Thank you, Megan. Okay. There are a lot of reasons. One of which has to do with dynamic range. In fact, you know what, let's go to this picture first, and then we'll go to the numbers. So here's a series of photographs where you adjust the amount of light coming into the image from a thousandth of a second to a quarter of a second. You see here there's an outdoor part of the scene right there. So a thousandth of a second isn't enough when you open it up a little bit more it gets a little bit easier to see these things. And then as I let in enough light that I can see the interior really well this outdoor thing is totally blown out. And yet, if you were to stand there, what you would see is something that looks like this. You'd have no problem seeing the indoor stuff and outdoor stuff. It might be a little bit harder to see, but not anything like the camera and that's because your system deals with dynamic range so much more effectively. Okay. Here's another example of photographing a high dynamic range scene. Right, so, if you wanted to see the stained glass really well you'd have to miss the rest of the inside of the church. If you want to see the rest of the inside of the church these things get blown out. But you would have no problem seeing this. You would see that beautiful stained glass, and you would see the church. You now know that there are a couple of photography methods called high dynamic range, so you have a high dynamic range on your iPhone, or your little phone, or whatever, or in Photoshop, something like that. That's a way of blending together a series of pictures to automatically select which regions from which image should be selected, or, or how much you dial up or dial down each region of, of the image in order to composite the whole thing into a high dynamic range image.

11 - Ganglion Processing

Putting some numbers behind this, the dynamic range that the human eye can deal with is incredible. It ranges over 14 orders of magnitude, and there's almost nothing in the real world that works over 14 orders of magnitude, right? So that means 10 to the 14, one and 14 zeros times from

the darkest dark to the brightest bright that you can deal with, and by the way, that does include looking, like, directly at the sun, because, frankly, you can't actually see much when you look directly at the sun. It's too bright. All right? But you can go from somewhere from 10 to the minus six to 10 to the eighth, what's called candelas, which is a unit of energy, per square meter. It's a measure of the amount of the flux of the light, and flux falls on a particular area. So, the human eye can set its dynamic range, its set point, to anywhere from this really low, this 10 to the minus six, 10 to the minus five, 10 to the minus four, so that'd be a range of two, up to 10 to the six, 10 to the seven, 10 to the, another range of two. It can set it anywhere along that spectrum. At any given point in time, it's operating at at least a hundred to one, maybe a thousand to one, maybe, maybe even more than that. It depends upon how you're doing. And in any given sort of real world scene, like I was just showing you, the actual ratio of sort of the darkest area to the brightest area, can sometimes be in the order of 100,000 to one. Right? So you're standing there, it's really bright outside, it's kind of dark in here, that difference might be under if, if it was that far, even your eye would have difficulty dealing with it. But it's way better than most imaging systems. Now, the thing is, everybody knows about your pupal, right? When it's bright out the pupal closes down, when it's light you know, when its, dark, the pupal opens up. That's a very small range of variation. A colleague of mine, professor actually, used to say that it was like an order of 10 to 100 variation, in terms of the size. He actually said 10, but I think it's a little smaller than that. But how do you do the this great huge variation? Well, it's actually the retina's ganglion cells that make this happen. And the way it does it is, and this is sort of looking at just a single section, so remember, light's coming in this way, not too important for the, here we have our photo receptors, here we have our horizontal cells, our bipolar cells, Amacrine, and then there's these ganglion cells. But notice that this thing is referred to as the surround, the center, and the surround. And you can think of this as being a slice through a circular operator, okay, and that circular operator would look something like this. Okay? In the retina, they are refered to as center-surround receptive fields, and what that means is, that in the positive ones takes the illumination that's from the middle, and they subtract off the illumination around the back. The negative ones do the negative in the middle and subtract, and add, positive. But basically what they're doing is they're comparing the local illumination to the, a little patch of the illumination to the overall illumination. So what happens when things get much brighter? Well if it's just getting brighter over the whole thing, not a lot different changes. So one part of your retina can be sort of electrically and chemically setting its dynamic range at one place, and another part is at another place simultaneously. So the part that's looking at the bright window in the church is operating at one place. The part that's op, that's looking at the the darker part, it, it, it's doing its center-surround on a darker area, but it's looking at those relative contrasts. This plot here is just a notion of what a center-surround field looks like.

12 - Cell Recording

Now, one of the cool things is, you can actually measure the center surroundedness of a ganglion cell while it's working, okay. I wouldn't recommend doing this to you or to anybody else you have any care for whatsoever, but basically, the way you do recordings is here, here's a schematic of a neuron. So it has dendrites where they receive inputs from the synapses, and when the, the inputs are correct, they will generate a spike, a firing down here, and then these terminals impact the next nerves that it's in contact with. And if you were to put an electrode right over here, you would occasionally see these spikes. And neural coding, we tend to refer to as the more spikes per second is a higher level, so we've gone essentially from analog to digital. Right? In an analog system, the amount of the voltage tells you how high it is or how low it is. In a digital system, there's something discrete that tells us, okay, you know, there's 15 ones and two zeroes, that's a higher number than all zeroes or, et cetera. Well in, in the neural system, the way we do is, we look at how, how often is the neuron firing. And you can actually go ahead and do this. This is a picture of a a micropipette, an electrode getting ready to record from a neuron. So you do these with neurons that are still living, right? This is why I said you wouldn't want to do this to anybody you cared for, all right? And what you can do is, you can then put different kinds of stimuli at different locations in the eye and see what happens, all right. And if you do that, you can find areas, so here we have,

when it's just black, this neuron doesn't do things very much, okay? But when it's white in the middle, okay, and black out here, all right. And, and what this is, oh, by the way, this is time going this way. So this is when the, the spot was turned on. You can see when it goes from the back, this is called the background activity. When it goes from the background, when it turns on, this thing starts firing crazy. And then you turn it off and it stops again. Okay, fine. But what's then cool is, if I then turn on a really bright spot. Okay? Nothing happens. It stays just like my background. And in fact, if I want to turn the cell off even more, I turn on just the outside and I make the middle black. And here what you can see is, it actually just shut down the background altogether. And then when I turn the thing off, it wakes back up again and then returns back to its normal. So this is an on cell. Okay. That is, its center is positive. Exact opposite cell here is negative. Okay. And both are found in the, in the retina. The other thing that's found in the retina are ganglion cells whose size in terms of their response Is different. So there are ones for very small, for medium and larger. And your system is actually tuned to have more of a particular size, and that tuning is what gives you this phenomena. So, again, this is going to be a function of how your monitor is calibrated. All right. But the, contrast sensitivity function, so for me, so what this is is, this is a sinusoid where the frequency gets, goes up this way, and contrast goes up as you get further down, all right? And the question is, where can you see the bars? And for me, I can see the bars kind of like, oh, like that. Okay. So the question is, why can't I see the stuff here, but I can see the stuff there, for the same level of contrast? The answer is, my retina has more and better tuned receptors at this size frequency than at this size frequency. This phenomena that you're seeing is a function of what goes on inside your eyeball. And that can be graphed out looking like this. Two things interesting about this graph. First is, this is a function of frequency. And, here they use different luminance levels. So they make it brighter and, or brighter and, it's sort of how sensitive you are. And then something, and for the PETA folks amongst you, please ignore this. You'll notice that the human data and the macaque, macaque is a type of monkey, are very, very similar. Now, this is done, of course, without doing anything to anybody, right? You just ask them what do you see, et cetera. But what this data show is that the vision system, the retinal vision system of the macaque monkey is really quite similar to the human. So if you were to do experiments on primates measuring physiological things, remember those electrodes, we have reason to believe that what we learn there is the same, would apply also to the human system. So, when you have psychophysical experiments like this, it shows you the similarity between human and macaque.

13 - End

So, at the retina, this local contrast information is being measured, it's being selectively done on different colors. And that's going to go out the optic nerve, and then start getting its way into the rest of the brain, starting with the lateral geniculate. And then further processing takes place up in the cortex. And both of those will be the discussion for the next lesson.

10B-L1 Vision in the brain

1 - Intro

All right, welcome back to Computer Vision. Today, we're going to finish our little tour of the visual system of the I'm sorry, the, of the human visual system. The anatomy of the human system. So when we last checked in with our heroes we had this little model going on here. Remember, here's our nose over here. Here's your eye, and it's got the optic nerve going through the sa, optic chiasm. And it goes to this first area called the lateral geniculate. And then from the lateral geniculate most of the stuff goes to the visual cortex, although some goes to superior colliculus, which we'll also talk about. Last time also, we discussed in some nauseating detail what goes on in the retina. So today is going to be sort of what goes beyond from above the retina.

2 - Thalamus and Cortex

This little section here, this little lateral geniculate, is actually part of the thalamus. If you were to take that thalamus and take a slice of it, which is what they're showing here. And then you stained for cell bodies and, and the neuroanatomists, they really have all sorts of clever ways of applying different kinds of chemicals that light up in different colors or intensities the different types of cellular structures that are there, so you can actually see what's going on. If you took this slice, you would see that the lateral geniculate has got these layers, right? You can see this layer 1 right here, right layer 2, and it's really remarkable that they have these, fixed layers, right? because remember, these things just grow, it's not like somebody got there with a soldering iron and built them. These things grow to have these computational structures, right? So, if you take, you know, the double EEs or computer scientists among you, you're used to seeing schematic charts of high end chips that have all these layered structures in them. You know, you've got your trans, your, your transistor arrays here and different memory things there. These things just grow like this, pretty cool. And the layers are different. In fact, you can actually tell a little bit that layers 1 and 2 seem to be a little bit different from layers 3, 4, 5, and 6. And in fact they are. Layers 1 or 2 are called the magnacellular layers and these are referred to as the parvocellular layers. And the, most important difference for you is that 1 and 2 tend to be the where cells. And these are referred to as the what cells, and that has to do with where they project going forward. And we'll talk about that in a minute. The other thing to know is that both eyes project to the left side for whatever's looking at the right half side of the visual field. And both eyes, project to the right side from what's looking at the left hand side of the visual field. Well, you can see that explicitly in the lateral geniculate. Layers 1, 4, and 6 come from the same side eye, and layers 2, 3, and 5, I get that right yet, yep. Come from the other side of the eye. So we haven't combined the two eyes yet. Right? We still have two separate channels. Why do we have to combine the two eyes? Well, how about things like stereo? Well, that's going to go on up in the cortex, right? You have to be able to get both of the images together. But at the, at the LGN level, lateral geniculate, they are still separate. The what, goes, the where goes up to the superior colliculus, which you can see on this kind of creepy picture here. so here it is, kind of creepy right you got your eyeball. Anyway what you're seeing here is this, coming in at the chiasm, through the lateral geniculate, and then projecting up to the superior colliculus. And the thing that matters as i said, i think earlier, the superior colliculus is involved in eye movements. So one of the things that has to happen very quickly, is if something moves over here, I have to be able to get my eyes over there very, very fast. All right? So you have very good sensitivity to motion in your retina, in the periphery. And then you've got a circuitry that's involved in rapidly going, you know, it doesn't go very, it doesn't make many neural stops along the way, right. We go from your retina which is doing some chemical stuff right away. One electrical stop into the lateral geniculate and directly to the superior colliculus. Now, as I said earlier, this is an overly simplified story. I encourage you to take a course on sensation, perception and neuroanatomy if you are interested in this. But it, the story is pretty close. So anyway, so those layers 1 and 2 go directly to the superior colliculus. But where is the really cool stuff happening? Well the really cool stuff happening is when you leave the lateral geniculate and you get up to the cortex. So here is a another picture of your brain just to remind you of where pieces are. Again, your nose is over here. Here's your cerebellum back here, involved in like balance and motor controls. And here are a bunch of the different corti, cortex, cortical regions. All right? There's the sulcus, which is this sort of gap here, right behind it, this is somatosensory, this is where you feel, your touch senses are. Right in front of it, by the way, not drawn in here, is a bunch of stuff having to do with motor control, controlling your muscles. Auditory cortex in here. And then this massive piece of stuff and the, it doesn't looks quite so massive in here, but it actually folds inside your brain. We're going to have to unfold your brain in a minute. That's all visual cortex and that's all the processing of visual information

3 - Visual Processing Areas

If we unfold your brain, okay? Which is generally painful, all right? What we have to do is we have to sort of take all this crinkled stuff and then open it up and layer it. What you would see is

something that looks like this. And bare with me here, okay? What's happened here is, this here is your eyeball, right there. This back area is V1, they're showing, this is on the inside, this is on the outside. Here's your nose here. This is the, these are your visual cortex areas here. And, if I were to peel it out, so take this brain and flatten it out, okay? You would get a map that looks something like this, so this big area here is V1, that's this area back in here. It wraps around. Then there's these other visual areas. Anything starting with a v is a visual area. And then it goes all the way up and here's that, so this is wrapping around this whole body, right? Here's that somatic sensory, the feeling. Here's your motor cortex, whole bunch of these other areas. Okay, this is what you would see. All right, but of course this is just a drawing. Now I want to show you a picture that's kind of cool and kind of gross at the same time, all right? And here's what we're going to do, and again, PETA folks, turn this off. All right. You take an animal, a living animal, you immobilize its eye by giving an injection that prevents the eye muscles from moving the eye, okay? You put up a flashing pattern that's in like some rings, that alternate sort of bright and dark spots, and you turn that on, but remember the eye can't move, so the eye is fixed at the same point. You then inject essentially a type of radioactive substance into the bloodstream that binds with sort of the oxygenated red blood cells. And the more cells work, the more oxygen they consume, okay? So, you can measure, so this is like, one of the first ways of measuring the activity of the blood cell, of, of the, of the brain cells by, you have them do this work for a while, while this radioactive tracer's connected to red blood cells and oxygen. And what happens is the blood cells that are doing a lot of work, build up a lot of this radioactive material. But of course, you have to get a picture of that radioactive material, and it's a very thin little bit of material, in a very thin part of the brain. So what you have to do is what they refer to as euthanize the animal. So the animal is, is, killed. You then, very carefully, un-peel its brain. And you can take a radiograph, you can actually measure, you know, you, you can think of it as simply as putting the brain, that brain tissue next to a photographic plate, and the radioactivity makes an image, okay? There's a little more to it than that. When you do that, you get this remarkable picture, and this one comes from 1988. So, even though this is sort of very fundamental work, it's not all that old, okay? And here is what the animal was looking at, okay? It's macaque. All right, so here you can see the, the visual stimulus the animal's looking at. And here is the, the picture of its brain. And what's so cool is that this structure is preserved. This is what's referred to as retinotopic mapping, is preserved within the cortex, okay? So, when you move a little bit in the retina, you move a bit in the cortex. Now what's also really cool is that it is not one-to-one, in terms of, how much cortical area is devoted to how much of the retina? In fact, the way it's drawn out is sometimes referred to as log polar, et cetera. And again, this is where we've unfolded your, your visual field. And you see that like this little area, 1 2 3 4, is looking at a lot on the visual cortex. And this area out here, okay, is much, much smaller. So you use more brain real estate for that tiny little area in the middle of your visual field, than you use real estate for bigger areas out here. Which also, I mean, yes, it makes sense, because that's where all the processing needs to be. But the fact that you can actually find this in the actual tissue is, you know, in some sense remarkable.

4 - V1

The other way that, that we can do certain types of understanding what's going on in the brain is with what's called physiological measurements. And so again using animals, typically, what you'll have is you'll have a screen where certain display, certain things are being displayed. An animal whose head is restrained so it's looking at a particular direction. And there's actually an electrode recording from the neurons in the brain. You hook that up to an amplifier. In the old days, you'd look at it on oscilloscope, and actually, you'd just connect it to a speaker and you'd listen. And more clicks meant you were getting more response, and you have to write it down, and report it. Then you'd have an oscilloscope with traces, now of course, it's all computerized, all right? You've seen this picture before, this is a picture of what it looks like for a microelectrode to be recording from a neuron. And up on top, this is what the actual voltages look like, these spikes. And so then down here, you have something that just sort of counts the, the spikes that go across. So one of the first areas that people want to look is what was called V1. Back in to old days, it was actually called area 17, I think, but that sounded too much like UFOs or something. I don't know. It's V1, the first

visual area. And when you record from the first visual area, and here's an example, the way you get a monkey to look at a particular place is you give him juice whenever he pushes a button when something happens. And he, so he'll attend very carefully there, and it, and it happens, he pushes the right button, he gets juice, and apparently, monkeys really like apple juice. But while you're doing that, you're doing other things to the visual field. And one of the things they've found is that there are cells that if you put up a bar, if you put the bar, let's say horizontally, you get hardly any reaction at all. In fact, put it up at a 45 degree, still hardly re, any reaction. Put it up vertically in this case, and you get a tremendous amount. And by the way, again, this is the part where it's off. This is the part where it's on. This is the part where it's off. Now, for this particular trace, I don't know if the bar was moving at all, or just being flashed up there. The movement I'll, I'll show you in just a minute. But the idea is that within V1, the first visual area of cortex, there are cells that are extremely sensitive to the orientation of essentially edges. Remember, we did oriented edge detection? So this feels an awful lot like oriented edge detection. It would be an oversimplification to say that's what it's doing, but it's related. If you recall, we also talked about motion. Well, it turns out in V1, you also get directional selectivity, all right? So here, we have a cell, and these, of course, are the spike trains. And this particular cell, if you have a bar that's almost, that's tilted just a little bit and moving upwards, you get a small amount of motion. If you have a bar that's here on this 45 degree angle and you moved it upwards, you got a lot of spikes. What's remarkable is when you move it down, you got no spikes at all, or even less than resting state spikes. So not only do we have orientation selectivity, we have movement selectivity. And you can actually make a pretty cool plot of how this distributes in the brain. And so this is, this is essentially, I guess this is your brain on drugs because, I don't know, anyway, it's all colored. But the idea is that each of these colors represents these different orientations. And you can get these mappings, and you can see that it's not randomly scattered individual points. There are regions, all right? And in fact, these were referred to as hyper columns in the sense of that little tissue areas within the brain will all be associated with a particular movement. Now remember, it was retina topic. So this area is all up on one part of the retina. Sure, but within that area, there'll be little sub parts that are for verticals and oriented edges like that, including different kinds of motion. So those were the kinds of things that people saw in these first visual areas. There was a group of neuro-physiologists Hubel and Wiesel at Harvard, who I think they worked in, in cats originally, where they first looked at these some of these cells. They also look at a thing called XY cells having a steady state in motion, and they won the Nobel Prize for this work, okay? So this idea of trying to understand how the visual cortex was processing the visual signal was really in fact, that Nobel Prize was announced during, when I was in grad school. So it was in the 80s at some point. So that gives you an idea of sort of the recency of some of this work.

5 - V2 and IT

So now that we have bars and edges in V1, we can go upstairs a little bit to V2. And people do all sorts of work of looking at different types of patterns that cause different effects. And so some of the patterns were based upon, like, little gratings and shapes and swirls and things like that. And sometimes, they were just on these types of edges and things. And, in Dave Van Essen's lab, and, David, a very well known neurophysiologist. But when you look at things like this, it's, it's a little difficult to understand, you know, exactly how this is moving upstream in the computational world. We understand the need for edges. And then you could maybe make some argument about circles and, or, curved parts. One might say that, oh, you know, what I'm seeing is stuff that may be dealing with some types of intersections or those kinds of things. But the neurophysiologists, they keep looking to understand what sorts of images or low level features will cause re cells to respond. Going upstairs even a little bit further, there's an area called the inferotemporal cortex. And Tanaka did a lot of work on looking at different kinds of features that seemed to have cells that have a strong response to these and not other kinds of things. And the idea was that maybe these are where cells are starting to put things together, okay. And I, I guess one of the most well known examples actually came it was earlier and, you know, the dates go different. There was this so-called hand neuron, okay. And what this was is, this is a neuron that when you showed it a picture of a hand, it

would fire a lot, and the front of a hand, the back of a hand. A, remember, these are kind of like line drawings of hands, and even this one and, and that one, so different size scale. But what was interesting is, if you had an image that just had this little spokes there, it doesn't look like a hand to me, it doesn't look like a hand to you. Well, obviously, it didn't look like a hand to this cell either. And the idea that this was a cell. Now, you might wonder what this picture of this person looking like is. Well, there was this whole conversation of what was called, did we find the grandmother cell. That is, can you find a cell that just fires when it sees my grandmother? And I think this was just a play on that. But on the one hand, it was kind of cool that you would find a single cell that was responding to, say, just hands. On the other hand, we know for a lot of reasons that it can't be as simple as, you know, there's a one for one mapping between concepts and cells, because there's all sorts of reasons why that would be a bad design. But we're getting at this idea that what's being processed is no longer just very simple features, but something like, you know, this, which is obviously a collection of features in a useful way. To further point that out, here's two pictures of what you and I would say are approximately same size silhouette of a rhinoceros and a lion. And these look sort of similar in terms of where the stuff is, okay, but of course they're different animals. And what's nice is that in V1 these trigger similar cells. So it's like cells that are responsive to the, the edges in this location, because the edges are sort of in the same spot. But of course, they respond differently in IT. IT is inferotemporal cortex. You know, the idea is that maybe in the cortex is a cell that understands something about rhinoceros, so it won't respond when it sees a lion. And in fact, you can further confirm that by showing it, if you find a cell that sees lions but not rhinoceros, now you show it two different pictures. One that's a line drawing of a lion, and the other one a much smaller filled-in silhouette of a lion. These objects in some sense look different at the picture level, but they are the same thing. And unlike the previous pictures where they are different things but similar at the picture level, these respond very differently in V1, the early processing, and then there are the same in the inferotemporal cortex. So the idea being that higher up are cells in the vision system that understand something about the appearance of object types. At least, that's the, that, that sort of the, the story.

6 - fMRI and Facial Response

But what if you want to actually know what's going on inside human brains? And presuming, presuming these are people you actually like. So how would you measure what's going on in their brain? Well, you're not going to stick electrodes in their brain. What you're going to do is something called fMRI. Okay, or a functional magnetic resonance imaging. So many of you probably have had an MRI, or an MRI uses strong magnetic fields to sort of mess around with electrons in order to be able to get differential density measurements. Well, you can do something cool also. You can use a dye contrast that the magnetic fields perturb differently depending upon how much oxygen is in that part of the the blood. So, just like we did before with that experiment with the monkeys where you injected, sort of, that radioactive isotope connect the oxygen. Now what we can do is we can have you have die contrast, go into an fMRI, have you look at things, and see what parts of your brain are lighting up. All right? So here's an example, sort of notionally of an fMRI activation, so this is of course obviously a reconstruction based on the density, and here you can see the activity that's being done. This is a three dimensional reconstruction. This is what a slice would actually look like, right? This is a single activation slice. And these colors are different levels of activation compared against a, a background activation. So one of the areas that people were studying a lot, and arguing about was, was there a part of the brain that seemed very responsive to faces, okay? And it's called the fusiform face area, and it's a little bit controversial as to whether, or not something really responds to just faces, or other sorts of patterns. This is a paper recently 2004, actually Nancy Kanwisher, last author there, was a graduate student, when I was, same department, and she does great work on understanding physiology. And, and these, this paper's one of the, well known papers, Grill-Spector et al. And they're just focusing on this area, this FAA, fusiform face area, in terms of understanding where faces light things up. One of the reasons for looking at faces is that there's been a long history of sort of faces being special in, in human vision. So there's examples that very, very early on in an infant's life they have a preference, and they do test called

preferential looking, if you put something up does the infant tend to look this way, okay? So the, the, the experimenter holds something. The experimenter doesn't know whether it's a picture of actually a face, something blank like this, or something that has the face pieces on it. All the experimenter does is he, he or she holds something up, and he just says, yep, I think he looked this way. Yep. Nope. And the question is do you tend to get the looking more when you actually put up a picture of face? And the answer is, says here er, early minute, very early in life there seems to be a preference for faces. Likewise there were responses of various other things, responding to, this is a cell recording, that when shown faces, responds, and when shown these other things, fruits, hands, et cetera, don't. Likewise, there are measurements using this fMRI of areas that respond to faces. And so, this is another paper, another picture from that same paper I showed you before. And you'll see here that they've got different object types that they're showing, and the idea is that different parts. See, so you see this area here that lights up really well with faces, and doesn't here, here, here, here, or there? And the idea is that these are other objects that, sort of, took the same, amount of location, and intensity, and et cetera, but only lit up when the faces were there. So it was an example of understanding a part of the brain doing a particular process.

7 - More Pathways

Of course work and understanding how the brain processes visual information continues. One of the interesting things has been, sort of, separating out the, there seems to be work in, in, understanding that the layout of your visual field, the geometry of your field is different than understanding what's in your field that is the, the, detection of things. And so notionally here, here's an interesting little picture. The idea is that things come in, in V1 and then V2 starts to go up this area. This is the V3, V4 subsegment that sort of does the recognition. This is sort of what's there. At the same time, there seems to be what's referred to as this action pathway where V1 has a lot of projections into this V3 V5 area that's sensitive to, it is sensitive to shape, but the movements of shapes. And then there's always connections, cause there's connections everywhere in the brain. But the idea is that in the parietal lobes, that's a different structure up here, is more of an understanding of action. So you have this one pathway that's sort of the stuff that's there, and another one that has to do with sort of how stuff's moving, or, or where stuff is moving. There's another whole cool area of, another cool way of investigating what goes on in the brain. If you presume that there are these different kinds of process steps going on in different locations, then you would expect that people who have certain types of brain injuries or, or birth defect issues or whatever, that certain parts of the brain aren't working You would be able to find selective elements of their visual processing not function. We know this is true in language. Strokes in different areas can cause you to lose either semantics versus syntax. So what happens in vision? Well, there's this very well known woman. She has this very interesting phenomena, she cannot see motion. So like, she can't cross a street. She'll look down there and she'll see the car there. And then, you know, two seconds later, she'll see the car here. But she has no perception that the car is actually moving. 'Kay, so she can't actually cross the street. All right? Because she doesn't get any understanding of sort of that space time evolution of things. So that, you can think of that as an example of there's this part that's doing this motion and action stuff that, in her brain, is just not working. Remember she recognizes cars and toothpaste and all the different kinds of things in her world. But she doesn't get a perception of the motion and the action. And this is the type of thing that leads further credence to the idea that there's a separate pathway, this, this separate part of the brain that is processing this kind of information that obviously she doesn't have. So, the idea would be that you know this action pathway shown here Somehow is not functioning as well as it should in her, in her brain.

8 - End

That sort of ends what I'm going to tell you about the brain and visual function right now. It is a huge area of research, as you can imagine. It's, it's, it's really fundamental to the human system. As I said, somewhere around 30% of our cortex is devoted to this. We get insights from computation in understanding what the brain does. And sometimes we get insights from brain and neurophysiology

to thinking about the computation. And it's one of the areas in which we get some direct connection between what you might think of as artificial intelligence and natural intelligence. It's one of the areas that we've pushed furthest on that. There's also work going on, for example, in computational modeling of, aud of hearing, audition right? So to understand what goes on in the human system versus artificial system. And in the other senses things are just getting started. So haptics, how do you go from touch? I can touch an object and I can tell you about it's shape. How do I do that? How can I tell that a texture, oh that's probably the skin of an orange? All right, its a computational problem, and we'd like to get insight back and forth. So, sensation in general and on robotics side, motor control, is trying to get a computational model of what goes on in the brain. All right, so that's really all I'm going to tell you about the brain. It is fascinating stuff and even in the popular literature, there's a lot of information getting out there that I, that I would deeply encourage you to go take a look. And I think you'll enjoy it.

We're Done!

1 - Course End

Welcome back to Computer Vision. Well actually, we're done. It's been kind of a long road, but we just covered a bunch of material. I apologize if I talk too much. I do that and my kids tell me that all the time. But i think you got a bunch of good grounding on the fundamentals of Computer Vision. We manipulated images as functions, everything from frequency content to talking about filtering. We talked about how cameras and camera models work that allow you to reason about the geometry between multiple views. We talked about finding features in one image, and how you would find them in another image, so that you could relate them, either for doing image manipulation, doing for recognition, or just doing the geometry. We talked a little bit about image formation, in terms of how light interplays with material in order to make a picture to begin with. We talked about how in video images change and, and the motion happens in doing tracking. Then towards the end, we focused a bit more on recognition, classification. And we talked about how to do sort of more serious work in that area, you would have to know more about machine learning. And in fact, that's a good sort of jumping off point for where you might go from here. If you actually wanted to do some high-end work in computer vision involving recognition, classification, labelling, you probably would have to get well grounded in machine learning methods. At least, grounded enough that you could understand the methods that are currently being applied. There's another side, in fact, one of the other Udacity Georgia Tech courses on computational photography, which thinks about the manipulation of images with respect to creating new images, or thinking about it sort of an advanced media perspective. And in particular, even though this feels like a somewhat comprehensive course in computer vision, if you were to sort of go out there and you know, try to do work in a computer vision world, there are some things you'll see that you didn't see. We were very, shall we say, careful in many of the images that we gave you. Images are much less well-behaved than the images that we gave you. And one of the first things you find out is that wow, I wasn't getting the types of images that I was expecting, and so you have to think a lot more about how you get the images to be the way you want them to be, and then in order to do the computer vision. The flip side is consuming the information that comes out of computer vision, all right? Is that going to go into some system that's going to do more reasoning about the nature of what's going on? Are you going to set off alarms? Are you going to mark things? Are you going to go shopping? What's going to be the output, how's that output going to be used? So, I'm sure at this point, you feel like you've learned some stuff. There is a lot more to do, but a lot of what needs to be done is really the applications of these basic skills, to sort of the specific set of imagery that you're working with. So, I hope you will do some of that. If you do and it works well, please send us a note. I love it when I hear students come back and say, you know, I I thought your class was okay. It's kind of a lot of work. But man, am I glad I took it, because I'm now doing this sort of imagery analysis of aerial imagery, and all of a sudden I find myself using stuff that we learned. So

if you have any of those experiences, please share them with us, with me. And we look forward to talking to you again. Bye.

2 - After the class

Okay. So with this, now you are at the end of this class. You've watched all of the lectures, done the interactive quizzes, and learned some fun material. Hopefully you've diligently also done all the assignments and are looking forward to the end of the term. Feel free to continue to participate while the variety of forms you've created for this class after the term has ended. Thanks for taking this class and good luck on your future endeavors. [BLANK_AUDIO]