Detailed Computer Vision Course Summary

CH1:

First a description about computer vision

It shows some differences between image processing applications and computer vision

It emphsis important applications in low level:

* Photo manipulation
* Size
* Color
* Exposure
* X-Pro II
* Feature extraction
* Edges
* Oriented gradients
* Segments [insemantic]

Also it showed some applications of mid level:

* panorama [stitching images]
* multi-view stero [two images from different but close views, expensive]
* strucutred light scan [same output as multi-view but cheaper]
* range finding [time of flight]
* optical flow [which shows that the object X has moved by a mangnitude Y in direction Z]
* time lapse [like when you record the construction of a building for months and then display this video in seconds]
* LIDAR

Also it showed a little about high level:

* Detection
* Semantic Segmentation
* Instance Segmentation
* Retrieval
* Image classification

This is all in this chapter

CH2:

This chapter talks about what is an eye, and how we see and how we understand the visualality, and how our brain process the image inputs

When looking to something far the rays will be nearly parallel, otherwise diverged

Common eye types:

* eyespot [smallest and simplest, lowest acuity, photosensitive proteins, e.g. Euglena, 40-65 times, 28 of 33 phyla]
* piteye [better than eyespot a little]
* [pinhole eye, better than piteye, better understanding of directions]
* lens eye
* mirror eye
* ...
* complex eye [6 of 33 phyla, high acuity]

Human eye:

* complex
* light hits its retina
* the info goes to mind through optic nerve
* has 120 millions rods and cons in retina,
* has a blind spot, nerves then retina unlike octpus [retina then nerves]

Rods:

* 120 millions
* 1:1 photon
* slow
* operate in low light [acuity]
* takes about 7 mins

Cons:

* 6 millions
* 1:N photons
* fast and fine
* operate in high light [acuity]
* mostly packed at fovea

Fovea:

* small circle at retina around 1.5mm
* highest visual acuity

Fixational eye:

An issue caused to any eye, so it needs to fixed by moving the eye, so these moves called

* Microsaccades
* Short linear movement
* Sporadic
* Ocular drift
* Constant slow movement
* Microtremors Tiny vibrations
* Synchronized between eyes
* For seeing fine details

It doesnt mean because you are seeing you need a brain, but the brain can be useless withot seeing

Ganglia:

* the main cells that transimits the data
* 1 millions
* M cells: depth, movement, orientation/position of objects
* P cells: color, shape, fine details

Visual cortex:

* is the part of human mind that process images, it takes 2/3 brainpower [electricity]
* and its size is 30% of the brain

It consists of 5 layers:

* V1: primary visual cortex
* Edge detection
* Highly spatially sensitive
* V2: secondary visual cortex
* Size, color, shape, possibly memory
* Sends signals onward to V3, V4, V5
* Sends strong feedback back to V1

Visual cortex is splitted into two sides [ventral, dorsal]

Damage to dorsal system:

* Can recognize objects
* Poor visual control for tasks like grasping

Damage to ventral system

* Cannot recognize objects
* Can still manipulate them, grasping, etc

How a brain sees 3d images:

* One eye
* Focus - how much your lens must change to make object clear
* Blur - objects that are blurry are at different depth
* Parallax - observer or object moves, gets multiple views
* Two eyes
* Stereopsis - images from eyes are different
* Convergence - where your eyes are pointing
* Brain
* Kinetic depth - infer 3d shape of moving objects
* Occlusion - objects in front are closer
* Familiar objects - you know how big a car is…
* Shading - 3d shape from light/shadow cues

Next thing is what we see, what we see actually is wavelengths reflected

Visible light: ~400-700 nanometers

The "color" of an object depends on both the incident light and the objects reflectance

Photoreceptors:

* Each receptor has a responsiveness curve
* Receptors more responsive to some wavelengths, less responsive to others
* Rods: peak around 498 nm
* Cones: 3 kinds - Short: peak around 420 nm
* Medium: peak around 530 nm
* Long: peak around 560 nm

Our perception of color comes from cones

Each cone has essentially one "output"

Many variations:

* refers to the type of cones each creator has
* human is Trichromacy

Remember Not every color can be represented in RGB!

CH3:

This chapter talks about how we store image, and the basics of interpolation specially enlargement [resizing]

We have studied in this chapter bilinear [applicable], NN [applicable], bicubic [not-applicable]

We also studied bayers pattern that is being used in CMOS sensors

The higher light intensity the brighter pixel

Pixels can be int [0-255] or float [0-1]

Two ways of indexing:

Cartesian:

* (x,y), (3,6) is column 3 row 6

Matrix:

* (r,c), (3,6) is row 3 column 6

The DEFAULT IS cartesian

They are two ways of storing image array:

* HW [height then width]
* WH [width then height]

Those methods for one layer [channel], but for channels we go with:

* HWC [Height-width-channel]
* CHW [Channel-heigt-width]

The DEFAULT IS CHW

To find the index [entry] of a pixel in CHW use this role:

* x + y\*W + z\*W\*H, where image = (x, y, c) <(15,192,2)>

We also studied in this chapter the HSV colorspace, and its affect when we adjust one layer of HSV layers

Image can be represented as:

* Im: I x I x I -> R
* or
* Im': R x R x I -> R

NN:

* Nearest neighbor
* Looks blocky
* Common pitfall: Integer division rounds down in C
* Note: z is still int

ALG = f(x,y,z) = Im(round(x), round(y), z)

so you are just rounding the value

Triangle interpolation:

* ALG = Weighted sum using of triangles: Q = V1\*A1 + V2\*A2 + V3\*A3
* where 'A'th is Area of the corresponding tringle
* and 'V'th is the value of the pixel

Bilinear interpolation:

* ALG = Q = V1\*A1 + V2\*A2 + V3\*A3 + V4\*A4

This algortihm is the simplest form of this general role:

* q1 = V1\*d2 + V2\*d1
* q2 = V3\*d2 + V4\*d1
* q = q1\*d4 + q2\*d3

Both works

* Smoother than NN
* More complex
* 4 lookups
* Some math
* Often the right tradeoff of speed vs final result

Bicupic interpolation:

* A cubic interpolation of 4 cubic interpolations
* Smoother than bilinear, no "star"
* 16 nearest neighbors

Both NN and bilinear are bad at shrinking, Lots of artifacting, Staircase pattern on diagonal lines

This is all for this chapter

CH4:

In this chapter it talks about interpolation defination, and general enlargement algortihm

also it shows how can you enlarge 4x4 -> 7x7 image

And it demonstrates different types of enlargements using bilinear, bicubic, ...etc.

Interpolation - the insertion of an intermediate value or term into a series by estimating or calculating it from surrounding known values

Also it shows some common filters [kernals] and their effects like:

* laplacian and composite laplacian
* sobel [H & V]
* Gradient sobel
* Canny
* Weighted average

And it gives simple diffnation about convolusion which will be appear in next chapter

that this defination is for correlation not convolusion

Also it shows how to fix bilinear and NN problem with shrinking

and the answer is to use Weighted average which smoothing the image, and this smoothing

gives better shrinking

The problem with NN and shrinking is that they only keep the middle pixel without

changing which lead to staircase pattern problem

But weighted average is not ideal choice because it has a problem called box filter artifcats

Not weighted average is weighted sum

This happen when for example you have 9x9 kernal and all pixels in this kernal is green

except one pixel is different color [e.g. black], this pixel makes a big bias that leads to artifcats

specially if this pixel at the edge of the kernal

So the solution to this problem was using gaussian, because gaussian is highly affected

at the center while it very low affected when you go to edges

Also in this chapter you understand how composite laplacian has been generated

and the effect for H and V sobel, and how if we combined them together we get the Gradient

This is all for this chapter

CH5:

First it talks to you about correlation and convolution

Convolution is correlation but rotater 180 colckwise

Also it show some basic operations that is safe to apply:

* Commutative - A\*B = B\*A
* Associative - A\*(B\*C) = (A\*B)\*C
* Distributes over addition - A\*(B+C) = A\*B + A\*C
* Plays well with scalars - x(A\*B) = (xA)\*B = A\*(xB)

And because of this roles you can separate a filter into two parts

like 2d gaussian you can separate it into 2 x 1d gaussian

and the supressing thing it is faster

Most computer vision applications are convolution based

We are going to study in this chapter edges and a basic knowledge of features

The chapter starts by a good question, what is edge

To understand an edge we need to represent it into 3d where the hight of each stack represnts

the depth value

Image is an function [f(x)], while edge the change between place `a` to `b`

to it is the rate of change, and what does rate of change mean?!

yes it means differentations

So edge is differentations of a function [image]

Edges can be high response [increasing] or low response [decreasing]

For first derivative of image, where `h` = 1

we got a filter [-1,1]

Which is not so useful

But when `h` = 2

we got a more useful filter ([-1,0,1])/2

But after that we discoverd the edges are noisy, the image is noisy which doesnt return accurate

output of edges

So because of noiseness, we need to smoothing and what is better smoother than gaussian

The impressive thing is when you multiply ([-1,0,1]/2) with guassian filter

you got what is called today vertical sobel

But sobel wasnt good at detecting low response [decreasing] edges, so we needed better

solution

So they started to use 2nd derivative, which can detect low response

And this is also impressive, because the image is not f(x) but it is f(x,y)

it has two variables, so because of that they did partial differentations on x and y

and returned the answer WHICH IS laplacian filter

Laplacian filter works because it measures the "divergence of the gradient"

* Negative Laplacian, -4 in middle
* Positive Laplacian, 4 in middle

But also laplacian sensitive to noise

So because of that they created Laplacian of Gaussian, LoG

Another solution for edge detection is instead of using spatial domain we should use frequency

domain, WHY???

This is because as we said edges can be high response or low response which mean

they can be represented as frequencies

So this method called Difference of guassian [DoG]

* Gaussian is a low pass filter
* Strongly reduce components with frequency f < σ
* (g\*I) low frequency components
* I - (g\*I) high frequency components
* g(σ1)\*I - g(σ2)\*I
* Components in between these frequencies
* g(σ1)\*I - g(σ2)\*I = [g(σ1) - g(σ2)]\*I

DoG can look too similar to LoG

Another approach is to use gradient magnitude which doesnt use frequency domain neither 2nd differentations

It is just using magnitude of gradient

but this methods gives a spreaded-stroked edges, which we dont want

We want a lines-shape [thinner] edges

So in order to do this we are going to use Canny method

Canny method is additional step on gradient mangnitude method

Its algortihm:

* Smooth image (only want "real" edges, not noise)
* Calculate gradient direction and magnitude
* Non-maximum suppression perpendicular to edge
* Threshold into strong, weak, no edge
* Connect together components

Note that threshold step works like that:

* Only want strong edges
* 2 thresholds, 3 cases
* R > T: strong edge
* R < T but R > t: weak edge
* R < t: no edge

Strong edges are edges! - Weak edges are edges iff they connect to strong

This is all about edges, next is features

Features can be extracted by many ways [not descibed in this chapter]

A good feature can be used as useful patch

Good features are unique

The basic function to find how close two patches are is to use `Sum squared difference`

* Σx,y (I(x,y) - J(x,y))2

Sky is bad patch and feature [not unique]

Edge could match other patches [edges] which mean not a unique patch

Corners can be considered very good patches because they are not repeated and unique

To find a unique patch use this formula [auto-correlation]

* ΣdΣx,y (I(x,y) - I(x+dx,y+dy))2

The end