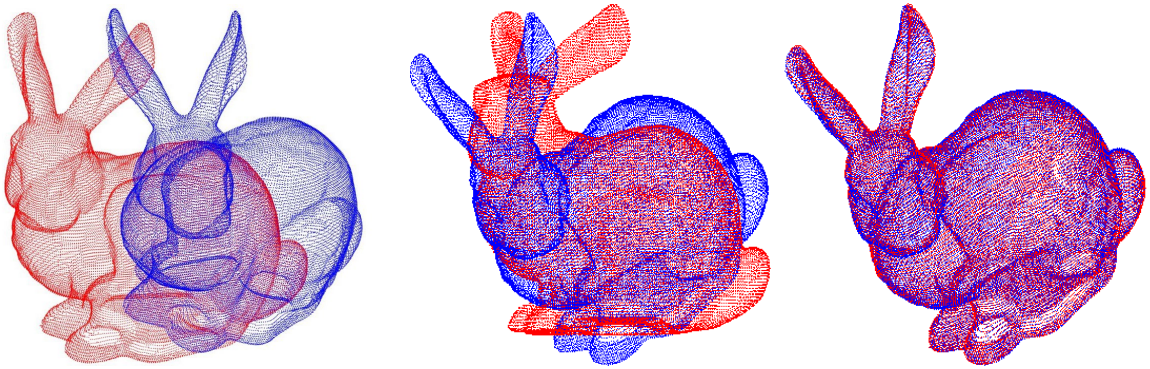
Perception:
Optical Flow and RANSAC

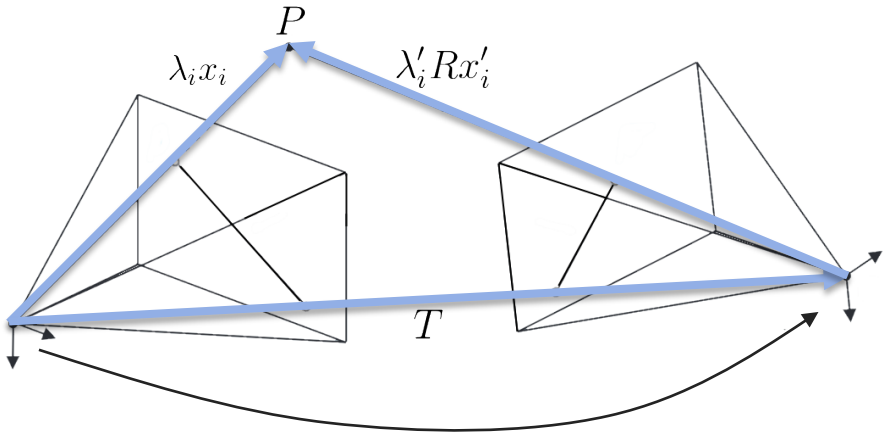
Last Time:

Procrustes Problem: $\arg\min_{R\in SO(3),T\in\mathbb{R}^3} \sum_i \|RP_i + T - P_i'\|^2$



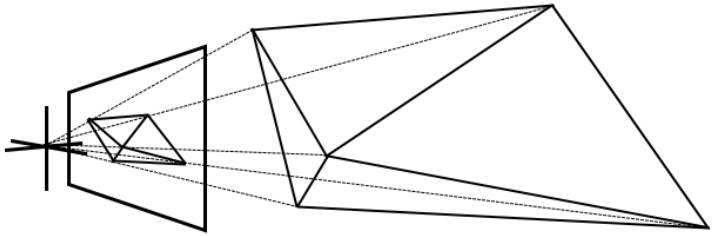
Epipolar Geometry and the Essential Matrix:

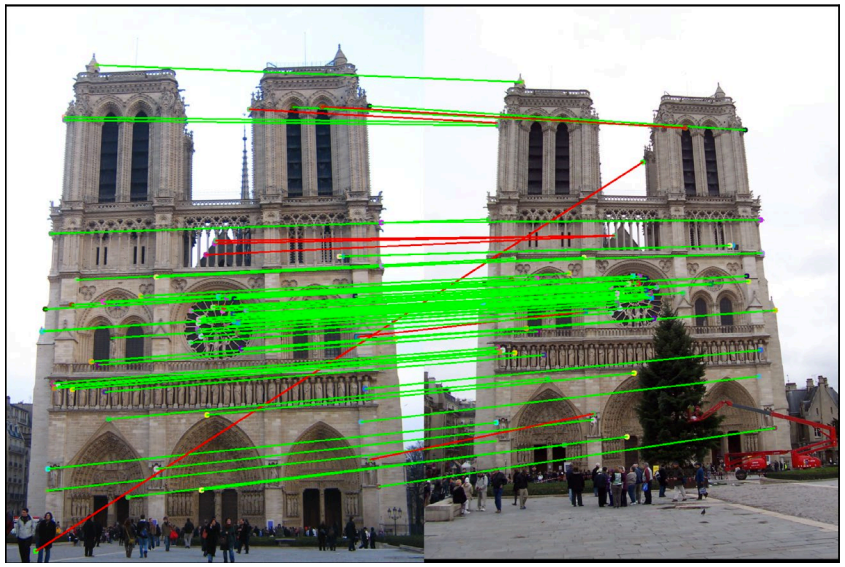
$$x_i^T [T]_\times R x_i' = 0$$

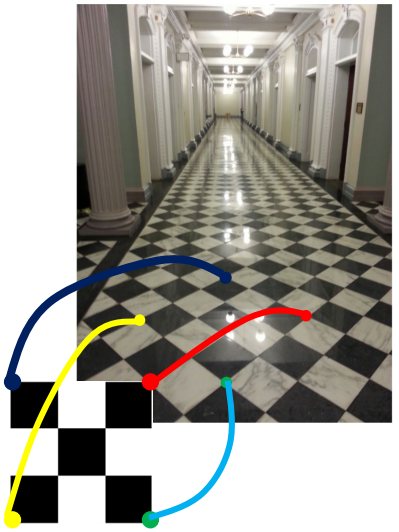


Now: Point Correspondences

In all the algorithms we have covered thus far we have simply assumed correspondences were given – but how do we compute them?







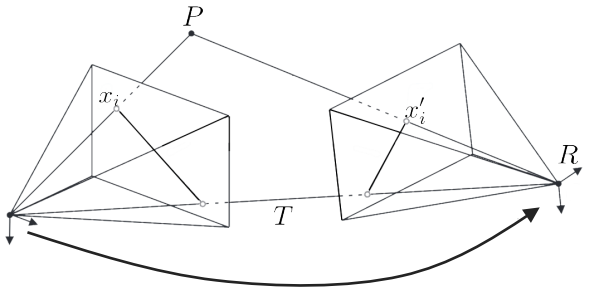


Image Motion

To try and solve this problem let's consider two consecutive frames in a video

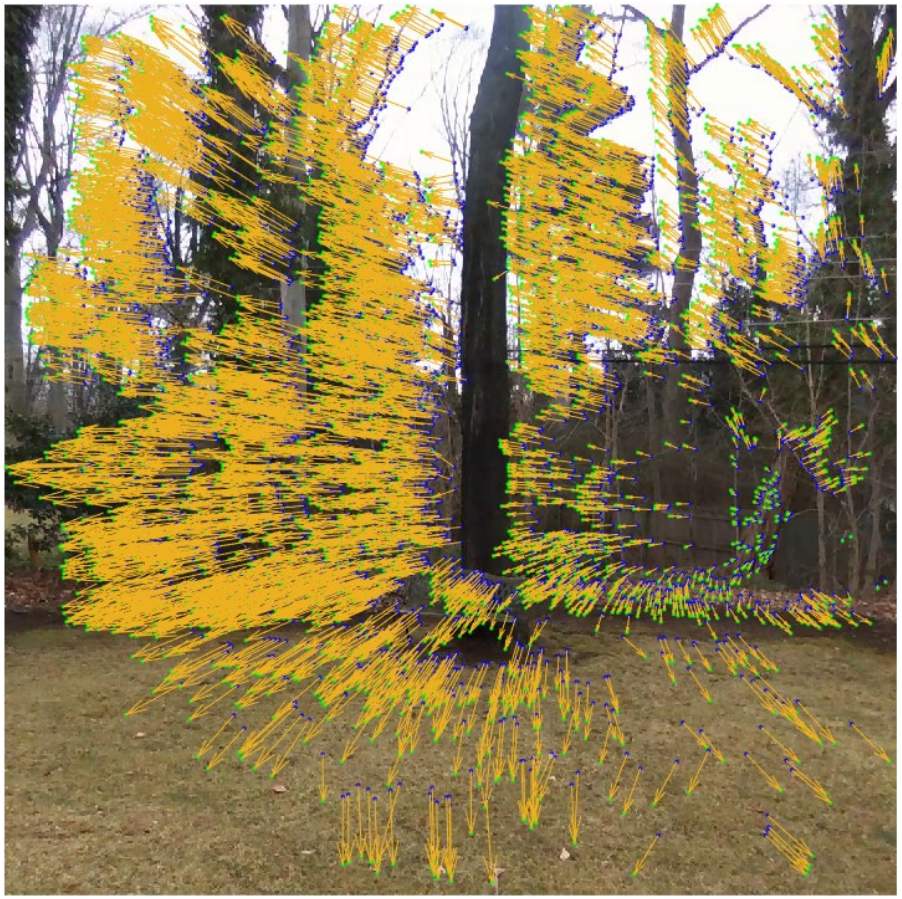
$I_t(x, y)$ $I_{t+1}(x, y)$

Pictures taken in Kostas' back yard

Image Motion: Goal

We want to find correspondences between the two images

$$I_t(x, y) \longrightarrow I_{t+1}(x, y)$$



Optical Flow

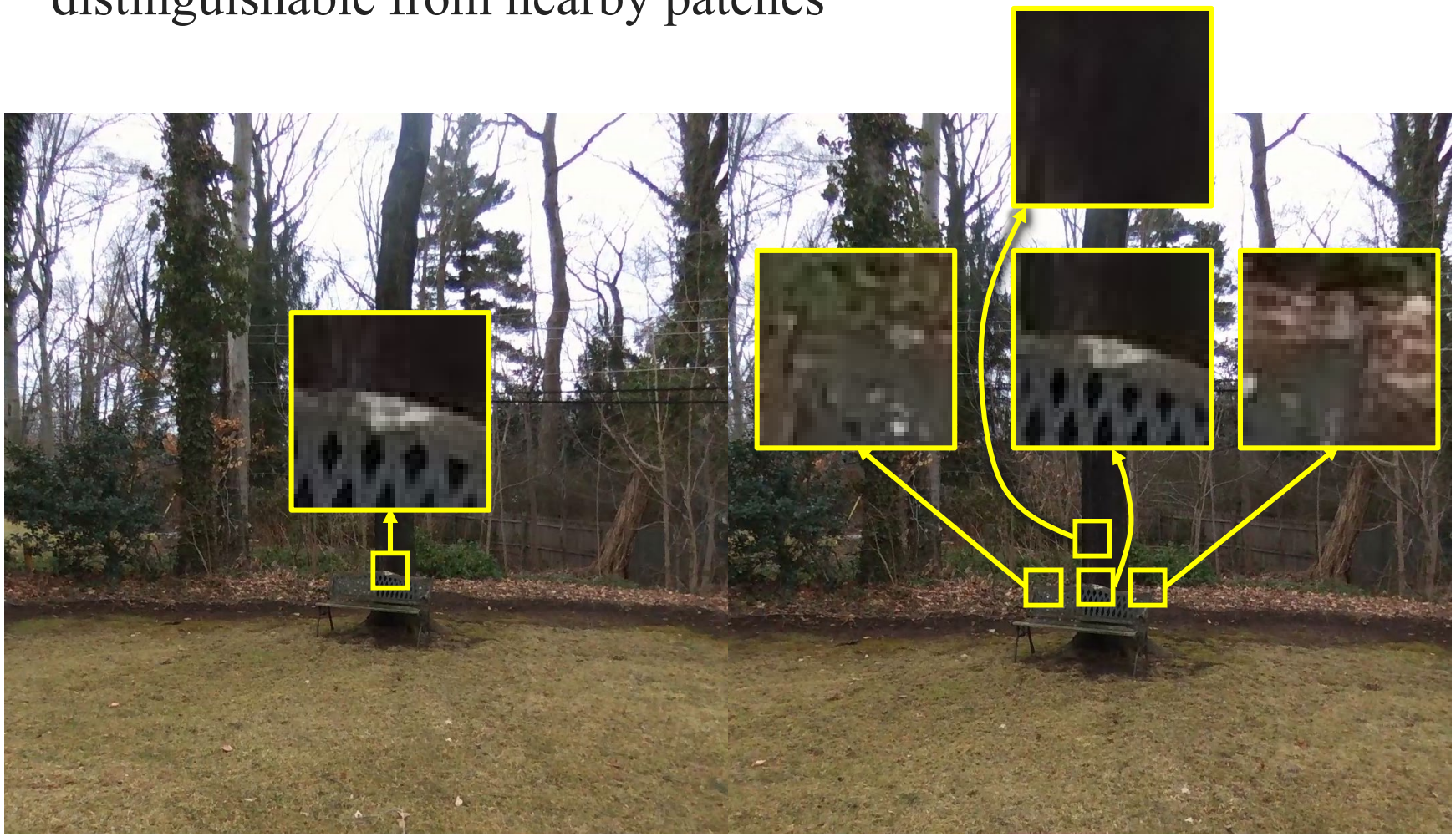
Looking the frames overlaid with each other you can better see where the motion is

In this perspective, we see we could move each pixel in the first to match the second. The motion of each pixel in this is called optical flow

Optical flow tells us which pixel in the second image a pixel in second image corresponds to

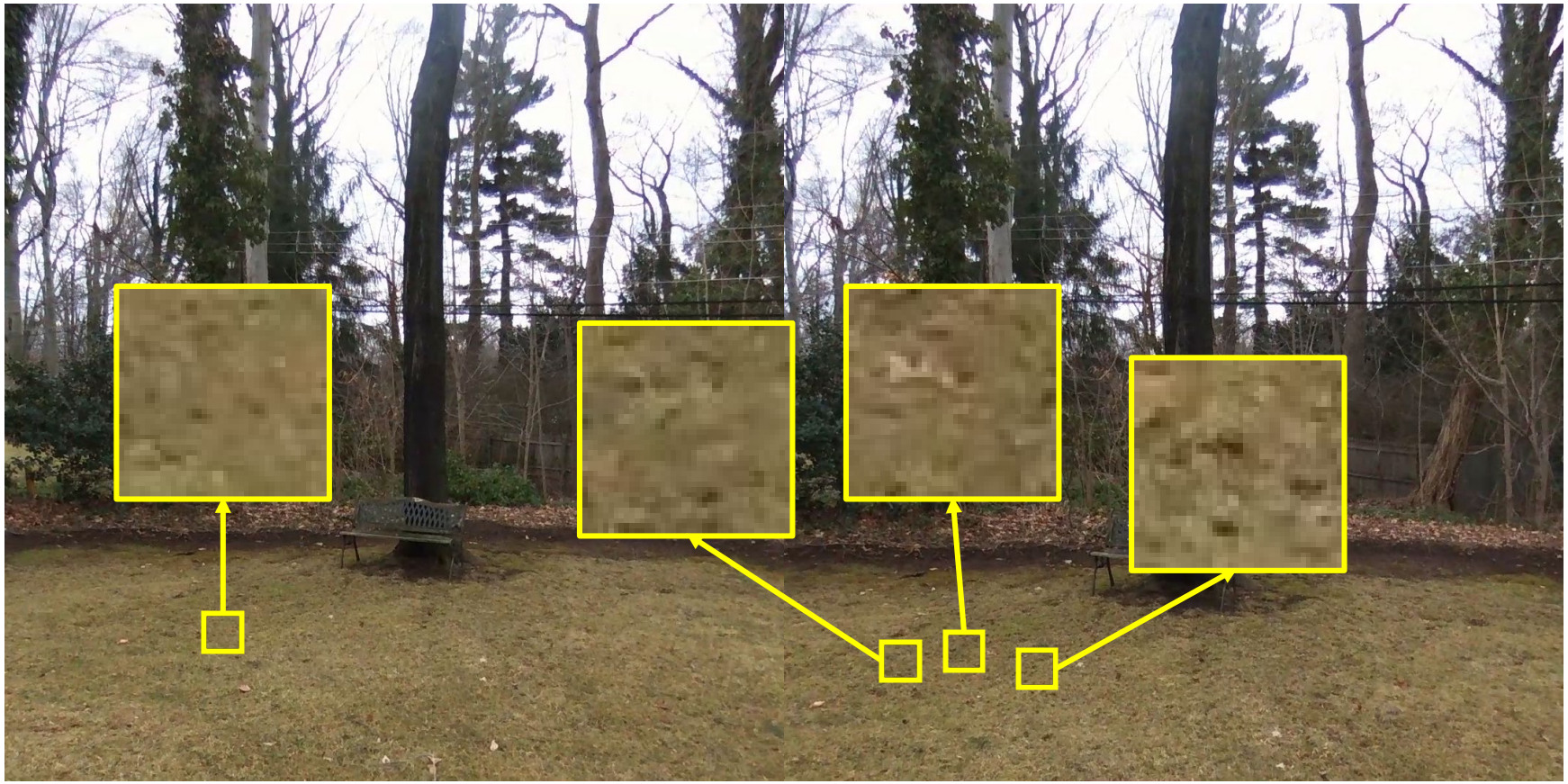
Optical Flow - Patches

You can see that some of the patches are easily distinguishable from nearby patches



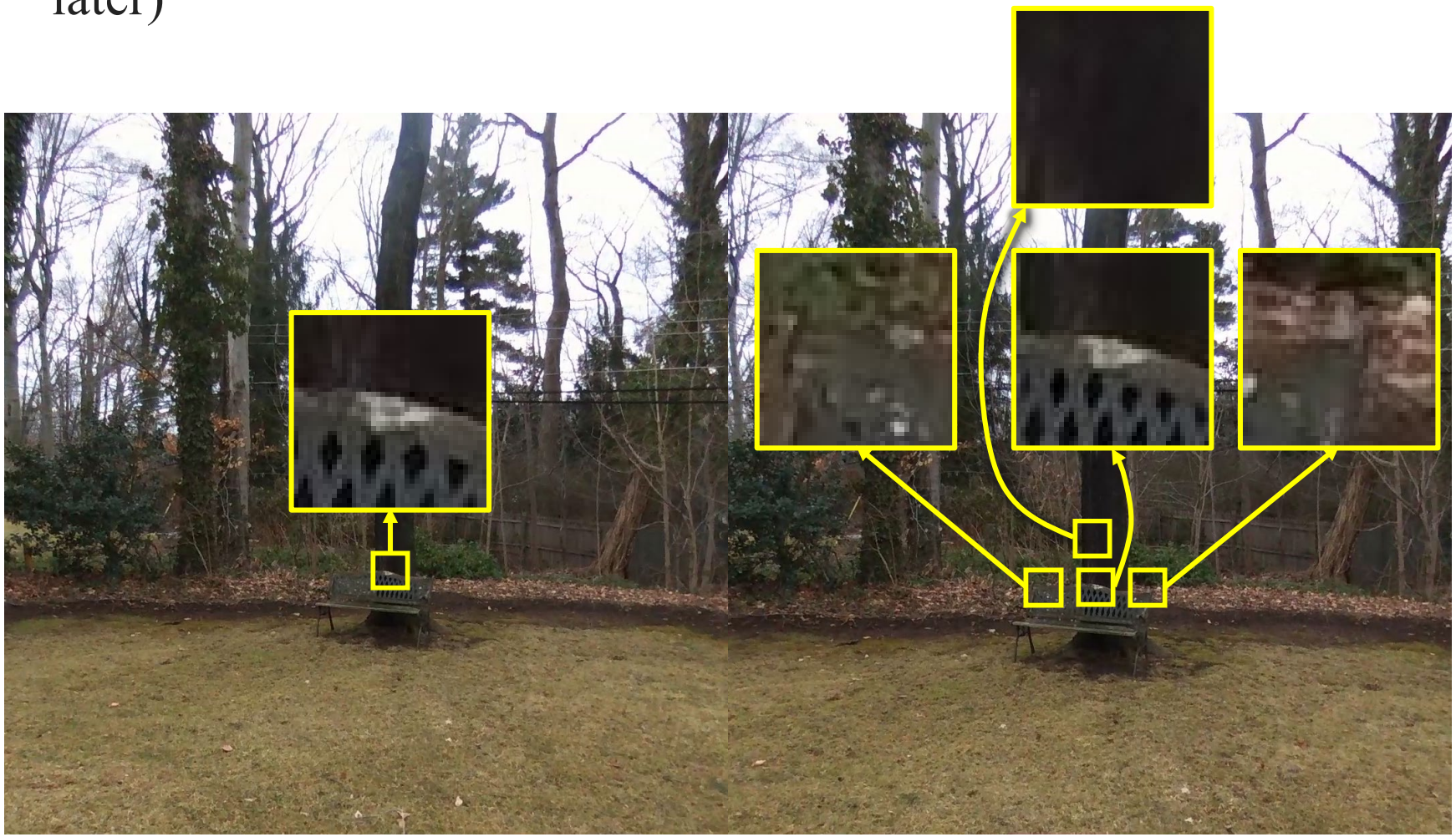
Optical Flow - Patches

Some are not…



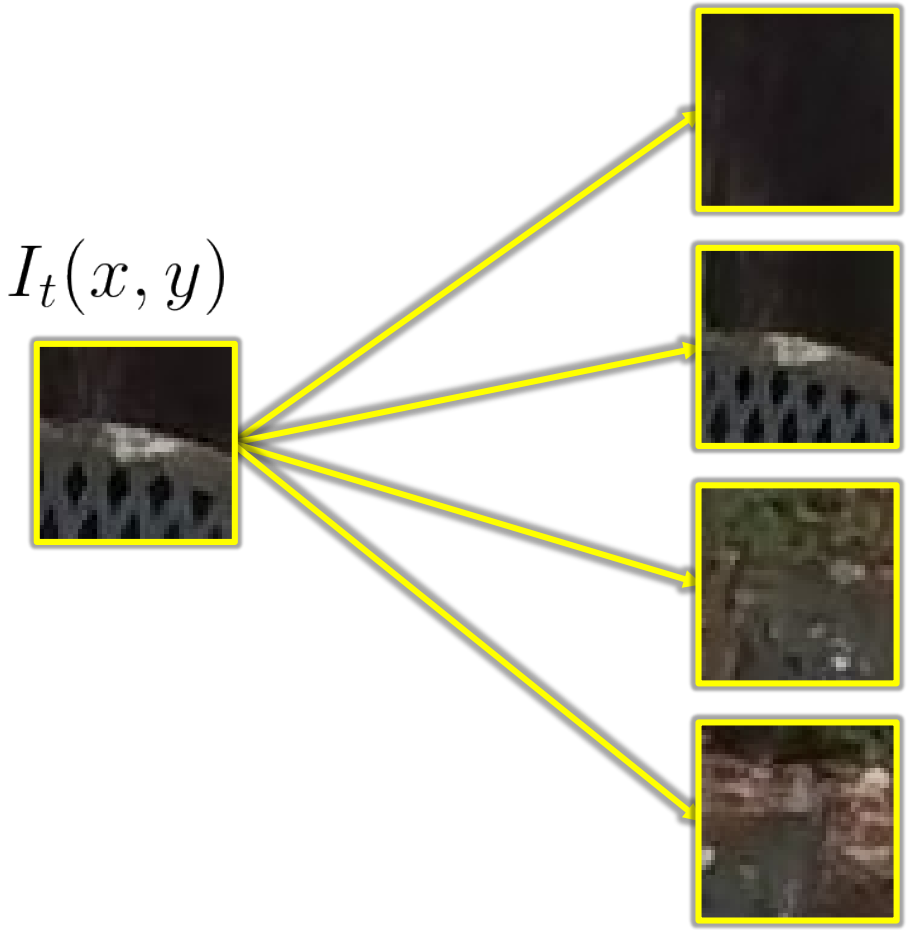
Optical Flow - Patches

Let's focus on the ones we can match (we'll characterize them later)



Optical Flow as Local Search

If the video frames are close enough we can look for these salient points in nearby images!

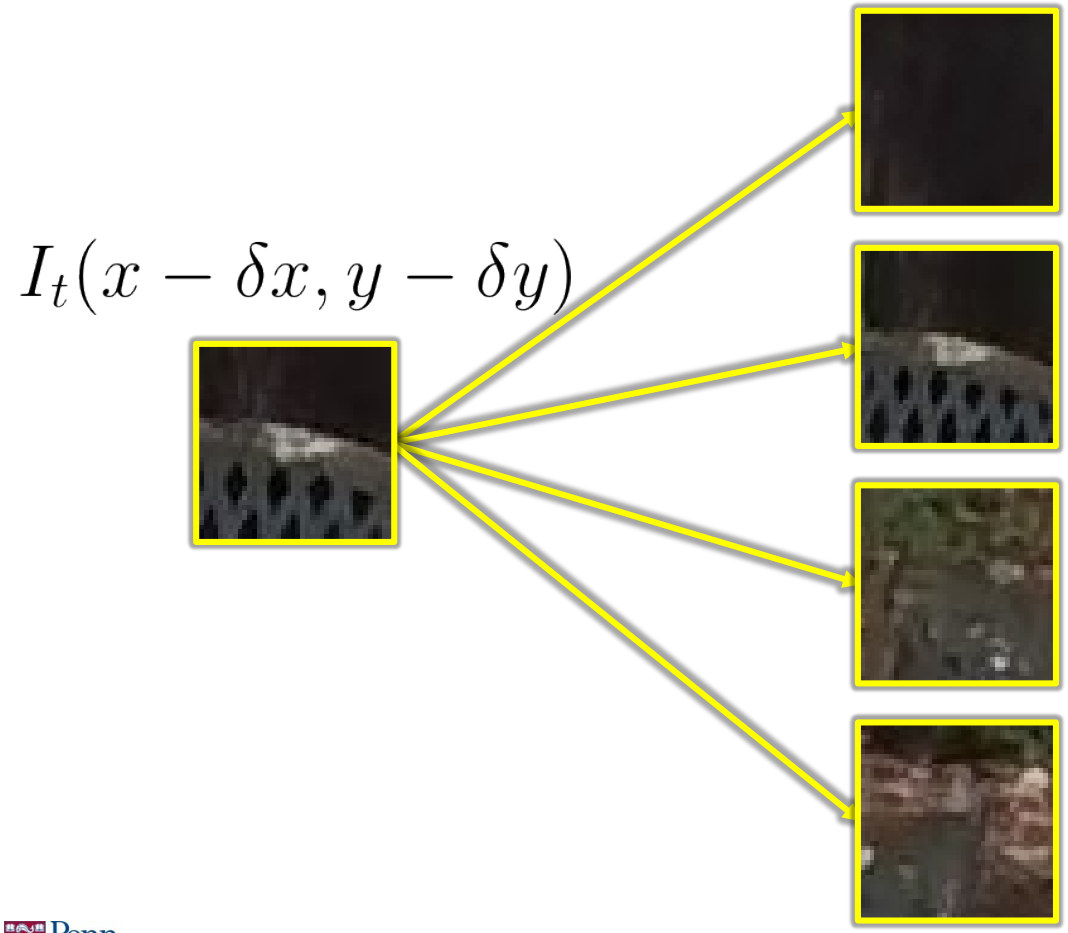


$$I_{t+1}(x + \delta x, y + \delta y)$$

Which nearby patch in the next image is the closest to the current frame?

Optical Flow as Local Search

Side note: Some people use the reverse convention in terms of which image to search on, but the math is the same



Which nearby patch in the next image is the closest to the current frame?

Local Search Algorithm

The function we are trying to optimize is (assuming a continuous image):

$$(\delta x, \delta y) = \arg\min_{\delta x, \delta y} \underbrace{\iint_{(x,y)\in\mathcal{N}(x_0,y_0)}}_{\sum_{(x,y)\in\mathcal{N}(x_0,y_0)}} (I_{t+1}(x,y) - I_{t+1}(x+\delta x, y+\delta y))^2 dxdy$$

If we discretize:

(x_0, y_0) Pixel where we are computing the optical flow

$\mathcal{N}$ Neighborhood patch around a pixel

$(\delta x, \delta y)$ Offset we are optimizing for – the optical flow

If time is no issue, we can solve this by exhaustive search.

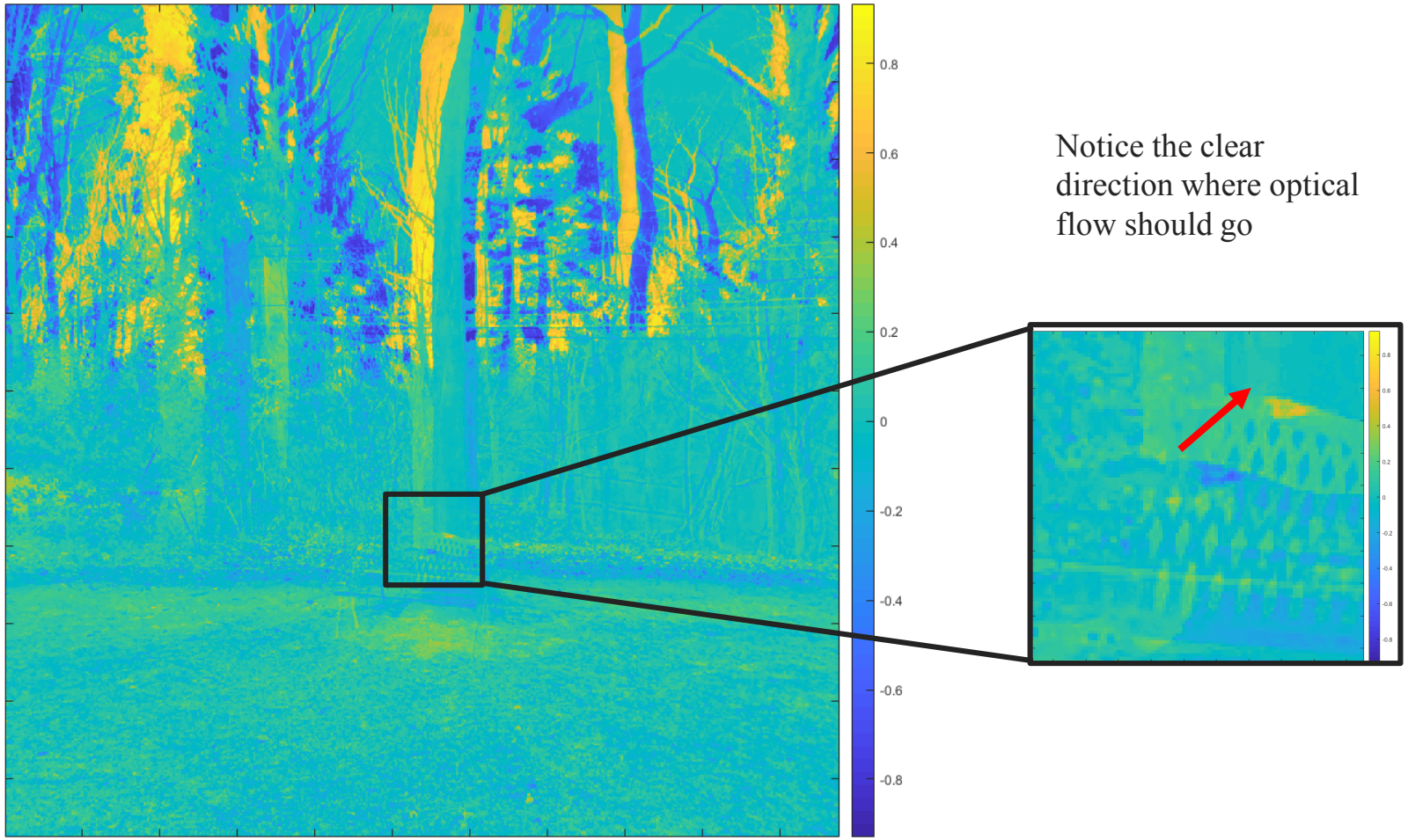
Local Search Simplification

If we want to speed things up. Assume that the change is small, we can use Taylor expansion:

$$
\begin{aligned}
& I_{t+1}(x,y) - I_{t+1}(x+\delta x, y+\delta y) \\
\approx\ & I_t(x,y) - \left(I_{t+1}(x,y) + \nabla I_{t+1}(x,y)^T \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} \right) \\
=\ & \Delta I_t(x,y) - \nabla I_{t+1}(x,y)^T \begin{pmatrix} \delta x \\ \delta y \end{pmatrix}
\end{aligned}
$$

Local Search Simplification

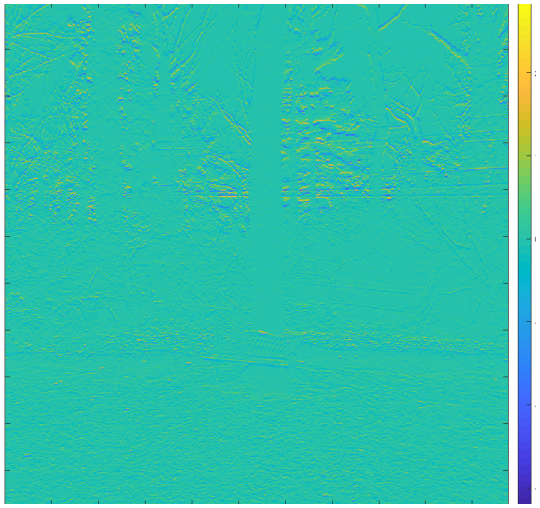
$$= \boxed{\Delta I_t(x,y)} - \nabla I_{t+1}(x,y)^T \begin{pmatrix} \delta x \\ \delta y \end{pmatrix}$$



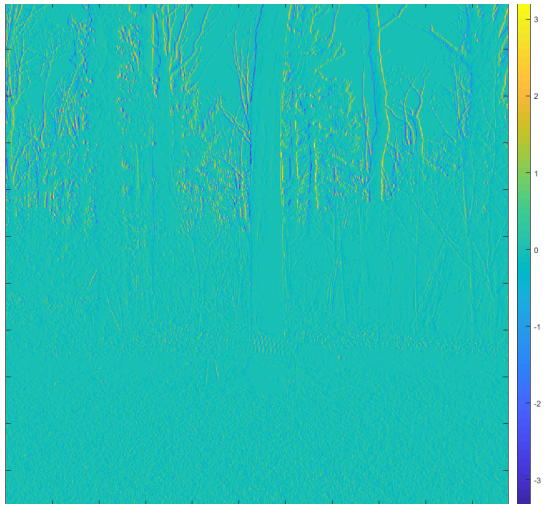
Notice the clear direction where optical flow should go

Local Search Simplification

$$= \Delta I_t(x,y) - \boxed{\nabla I_{t+1}(x,y)^T} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix}$$

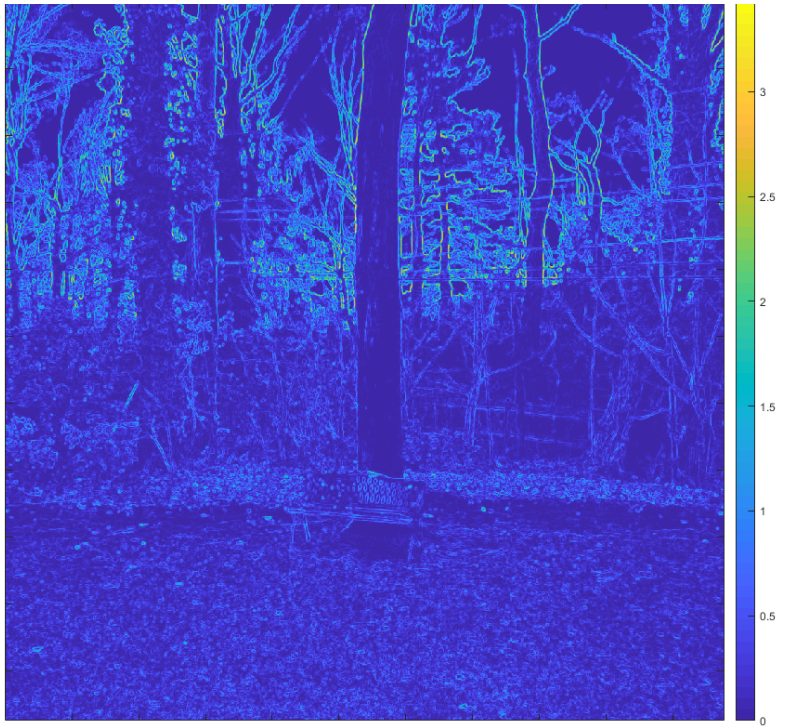


$\nabla_x I_{t+1}(x,y)$



$\nabla_y I_{t+1}(x,y)$

$\|\nabla I_{t+1}(x,y)\|$



Local Search Simplification

The new cost function becomes (in discretized form):

$$(\delta x, \delta y) = \arg\min_{\delta x, \delta y} \sum_{(x,y)\in\mathcal{N}(x_0,y_0)} \left(\Delta I_t(x,y) - \nabla I_{t+1}(x,y)^T \begin{pmatrix}\delta x \\ \delta y\end{pmatrix}\right)^2$$

$$= \sum_{(x,y)} \cancel{\Delta I_t(x,y)^2} - 2\Delta I_t(x,y)\nabla I_{t+1}(x,y)^T \begin{pmatrix}\delta x \\ \delta y\end{pmatrix} + \begin{pmatrix}\delta x \\ \delta y\end{pmatrix}^T \nabla I_{t+1}(x,y)\nabla I_{t+1}(x,y)^T \begin{pmatrix}\delta x \\ \delta y\end{pmatrix}$$

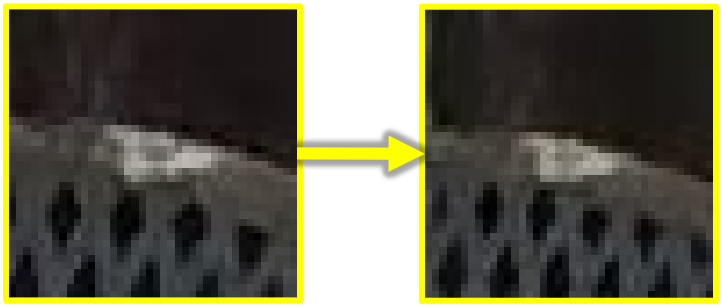
constant

$$= \sum_{(x,y)} -2\Delta I_t(x,y)\nabla I_{t+1}(x,y)^T \begin{pmatrix}\delta x \\ \delta y\end{pmatrix} + \begin{pmatrix}\delta x \\ \delta y\end{pmatrix}^T \left(\sum_{(x,y)} \nabla I_{t+1}(x,y)\nabla I_{t+1}(x,y)^T\right) \begin{pmatrix}\delta x \\ \delta y\end{pmatrix}$$

We can solve this directly!

$$\begin{pmatrix}\delta x^* \\ \delta y^*\end{pmatrix} = \left(\sum_{(x,y)} \nabla I_{t+1}(x,y)\nabla I_{t+1}(x,y)^T\right)^{-1} \left(\sum_{(x,y)} \Delta I_t(x,y)\nabla I_{t+1}(x,y)\right)$$

Assumptions We've Made



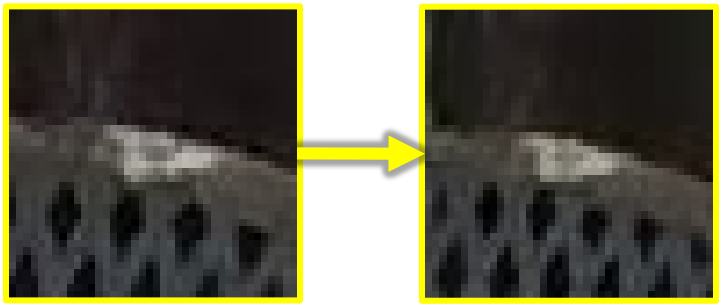
Brightness Constancy: Color doesn't change between different viewpoints

Counterexample

No occlusions: things stay in sight:

Counterexample

Assumptions We've Made



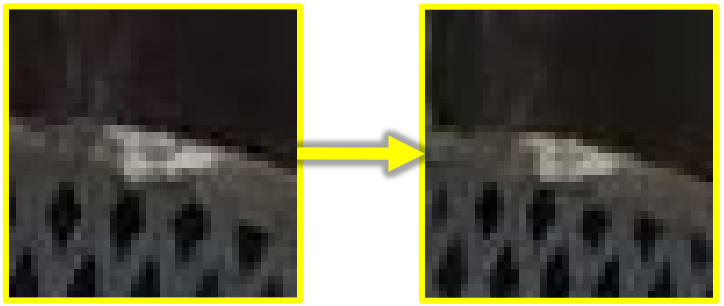
Minimal geometric deformations: No large rotations or scaling

Counterexample

Minimal patch displacement: Taylor expansions only work for small translations

Counterexample

Assumptions We've Made



Patch is sufficiently 'interesting':

$$\begin{pmatrix}\delta x^* \\ \delta y^*\end{pmatrix} = \boxed{\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T\right)^{-1}} \left(\sum_{(x,y)} \Delta I_t(x,y) \nabla I_{t+1}(x,y)\right)$$

We assumed we could invert this

When would this fail? And what does that mean?

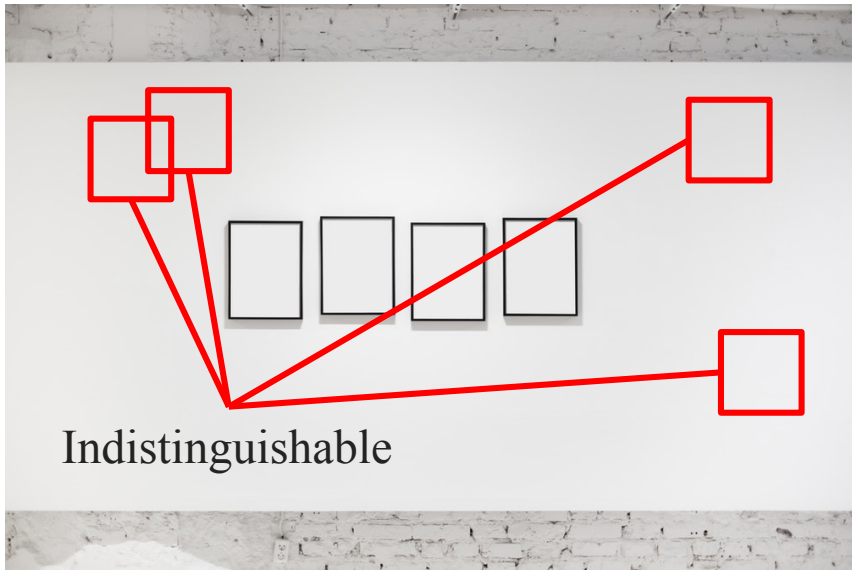
$$\det\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T\right) = 0$$

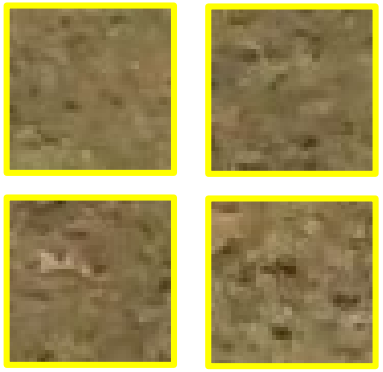
White Wall Problem

$$\begin{pmatrix}\delta x^* \\ \delta y^*\end{pmatrix} = \boxed{\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T\right)^{-1}} \left(\sum_{(x,y)} \Delta I_t(x,y) \nabla I_{t+1}(x,y)\right)$$

When would this fail?

1) $\nabla I_{t+1}(x,y) = 0$ No gradient = no information.
This is the 'White wall problem'





In our example, the grass problem

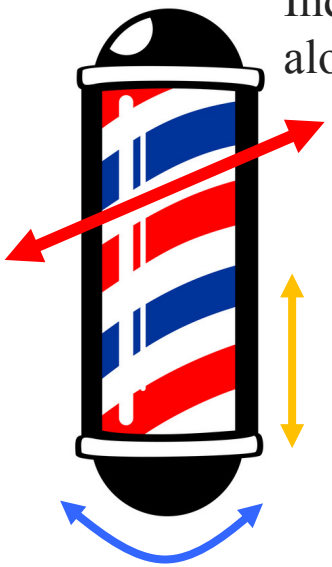
Barber Poll Problem

$$\begin{pmatrix} \delta x^* \\ \delta y^* \end{pmatrix} = \boxed{\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T \right)^{-1}} \left(\sum_{(x,y)} \Delta I_t(x,y) \nabla I_{t+1}(x,y) \right)$$

When would this fail?

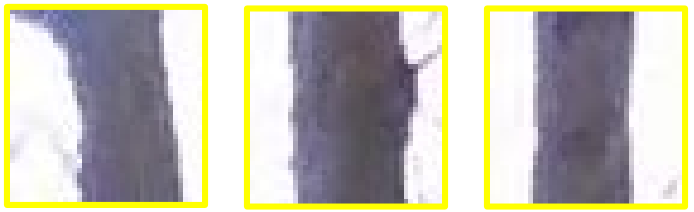
2) $\nabla I_{t+1}(x,y) = \vec{c}, \forall x, y$

Constant gradient = insufficient information.
No information along a direction



Indistinguishable along this direction

Hence it looks like it moves up



For our example, how high along the tree are we?

What This Means

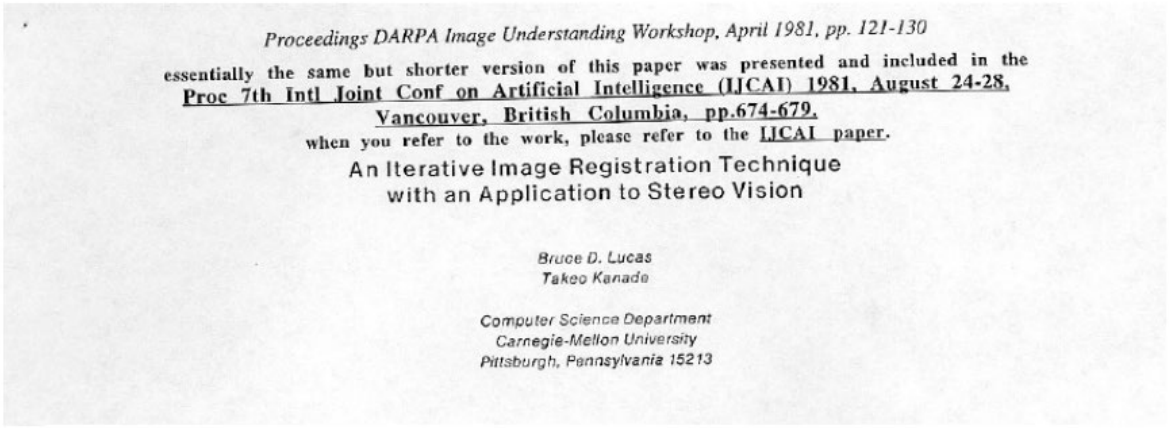
$$\begin{pmatrix} \delta x^* \\ \delta y^* \end{pmatrix} = \boxed{\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T \right)^{-1}} \left(\sum_{(x,y)} \Delta I_t(x,y) \nabla I_{t+1}(x,y) \right)$$

Most places in the image are ‘uninteresting’ – we can’t track them – the interesting places are ‘sparse’.
So we need to detect interesting ‘features’ (also known as corners) in the image that we can reliably track.
How can we find them?

Finding Features/Corners

There is a large literature on this spanning decades, But it typically involves the eigenvalues of that matrix:

$$\left(\sum_{(x,y)} \nabla I_{t+1}(x,y) \nabla I_{t+1}(x,y)^T \right)$$



Proceedings DARPA Image Understanding Workshop, April 1981, pp. 121-130

essentially the same but shorter version of this paper was presented and included in the Proc 7th Intl Joint Conf on Artificial Intelligence (IJCAI) 1981, August 24-28, Vancouver, British Columbia, pp.674-679.
when you refer to the work, please refer to the IJCAI paper.

An Iterative Image Registration Technique with an Application to Stereo Vision

Bruce D. Lucas
Takeo Kanade

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Shape and Motion from Image Streams: a Factorization Method—Part 3

Detection and Tracking of Point Features

Technical Report CMU-CS-91-132

Carlo Tomasi Takeo Kanade

April 1991

Good Features to Track

Jianbo Shi
Computer Science Department
Cornell University
Ithaca, NY 14853

Carlo Tomasi
Computer Science Department
Stanford University
Stanford, CA 94305

Extensions

There is a great deal of research trying to avoid these assumptions:

- Solve for the flow iteratively
- Compute flow bi-directionally for better reliability
- Assume affine deformations on the optical flow
- Compute the flow on multiple resolutions of the image and iterate to capture larger flow

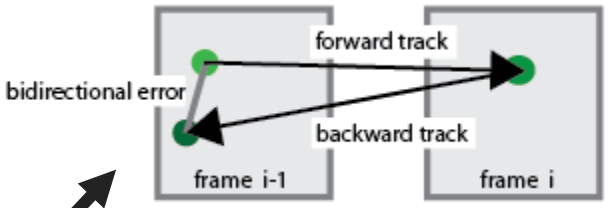
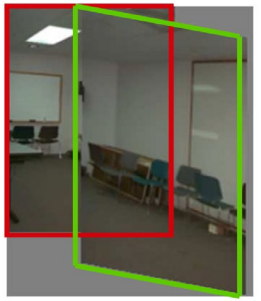


Image from MATLAB site



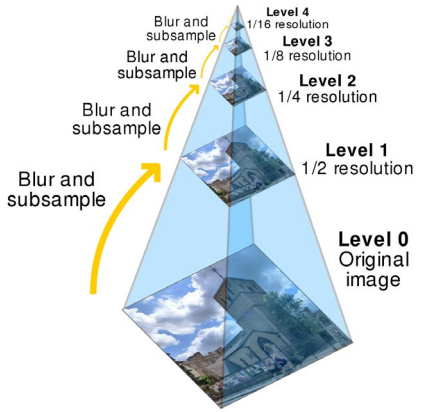
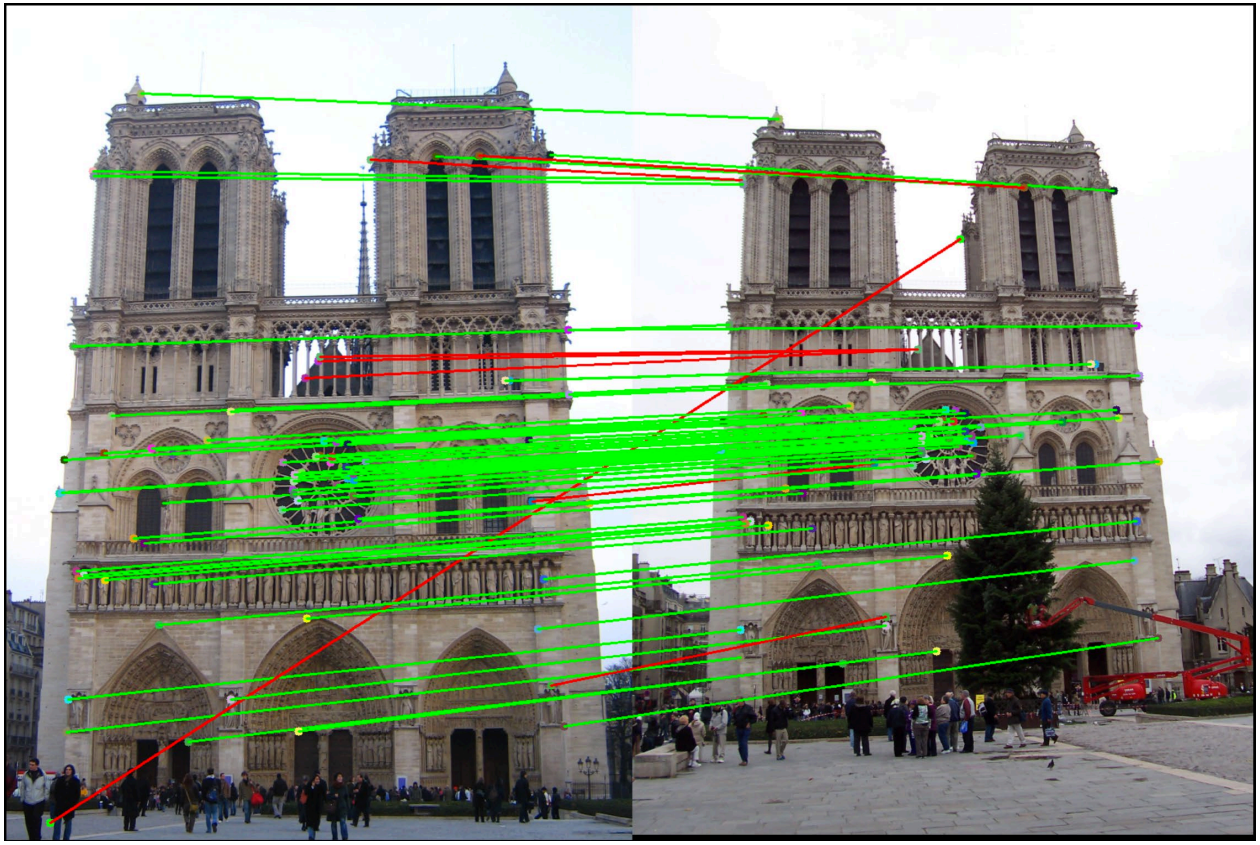


Image from wikipedia

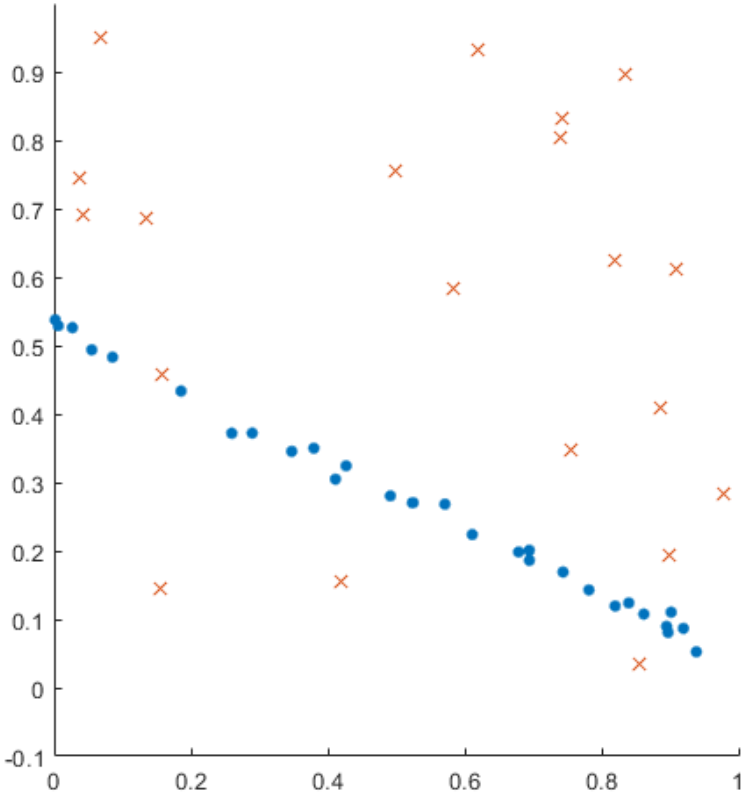
Finding Reliable Features

So we've found a sparse set of interesting features that we now can track. Even with this, sometimes failures happen. We need to make sure when we fit our models we have good features



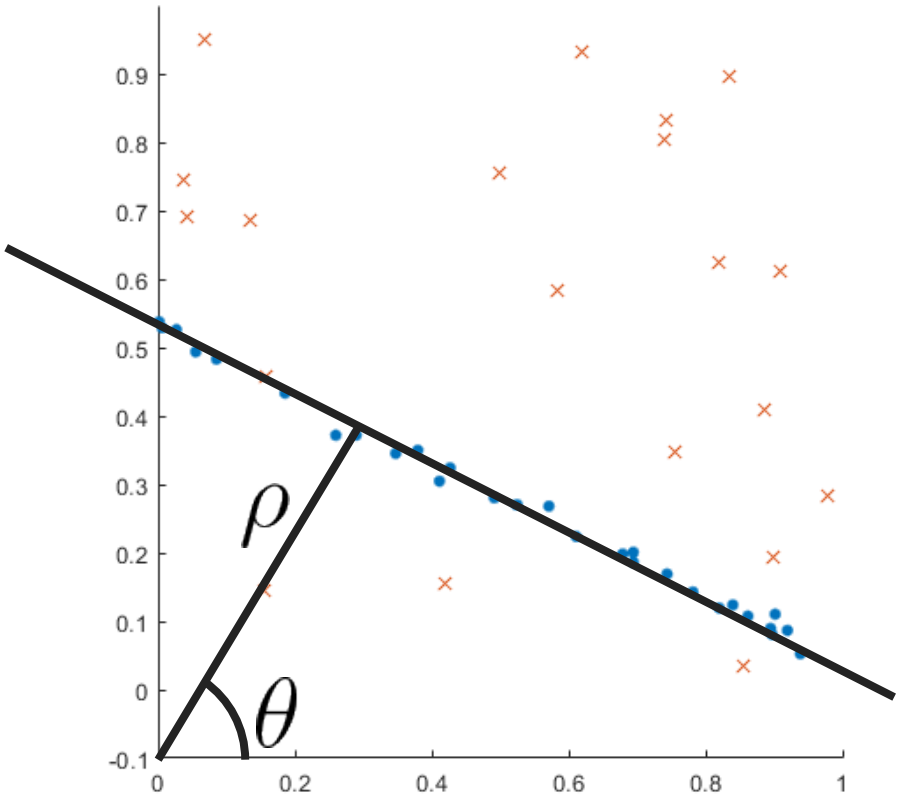
Minimal Example: Line Fitting

The easiest model to fit is a line – we'll use it as an illustration because the parameter space is only 2D



Hough Transform

One way to try and solve this is to use the Hough Transform. It works well when you have a low dimensional parameter space



Parameterize the line differently:

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$

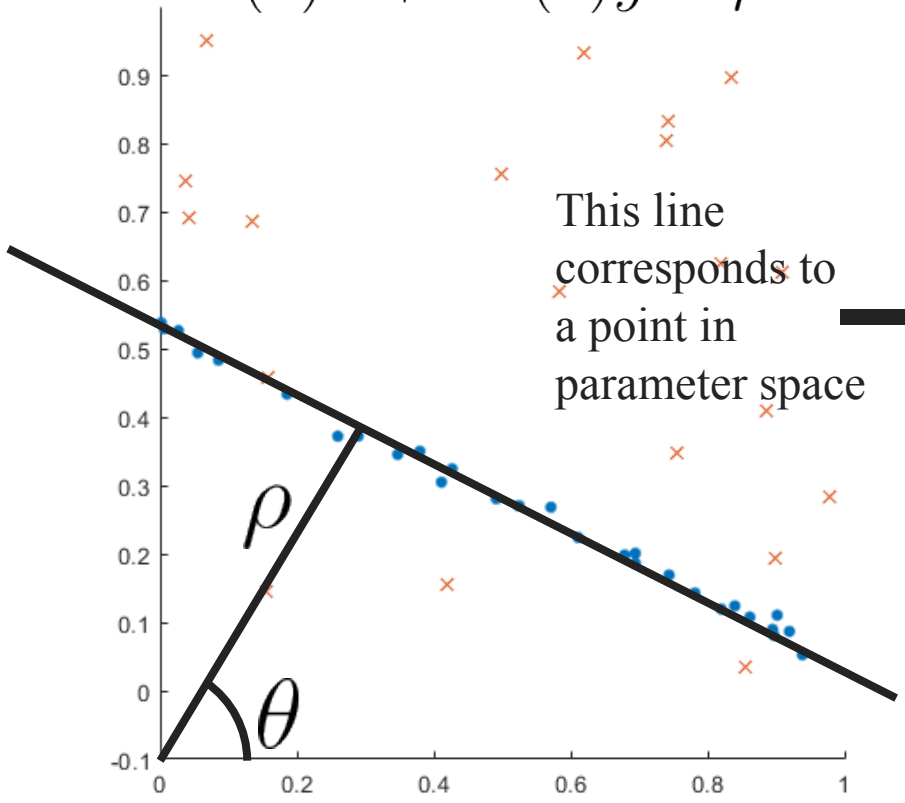
Noise model:

$$p_i \text{ drawn from } \exp\left(-\frac{1}{2\sigma^2}(\cos(\theta)x + \sin(\theta)y - \rho)\right)$$

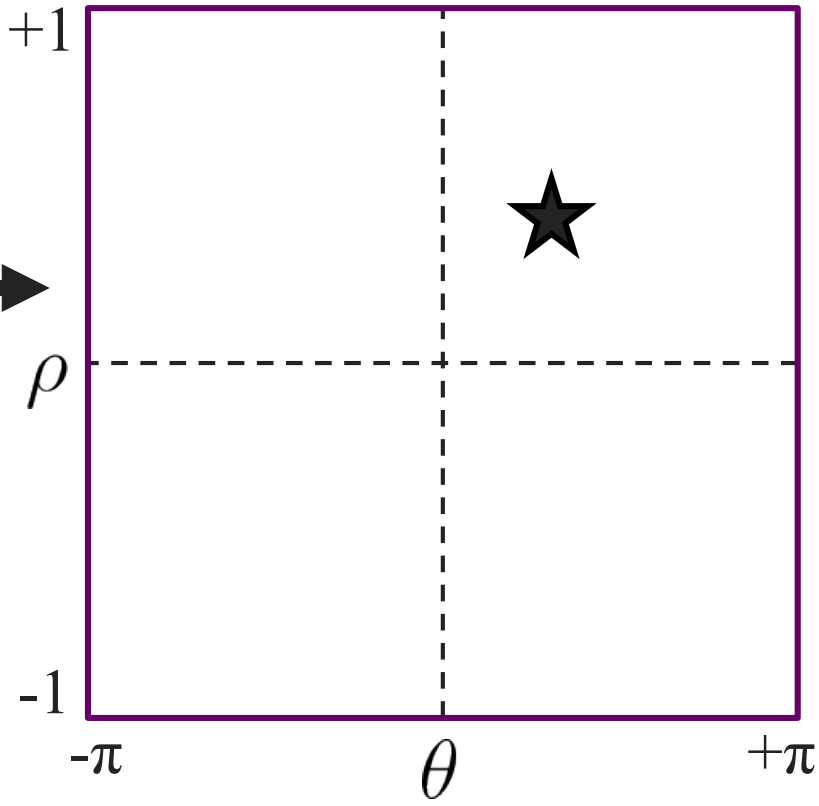
Hough Transform: Algorithm

Set up an array with all possible parameters you could have
In the case of a line, a 2D array

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$



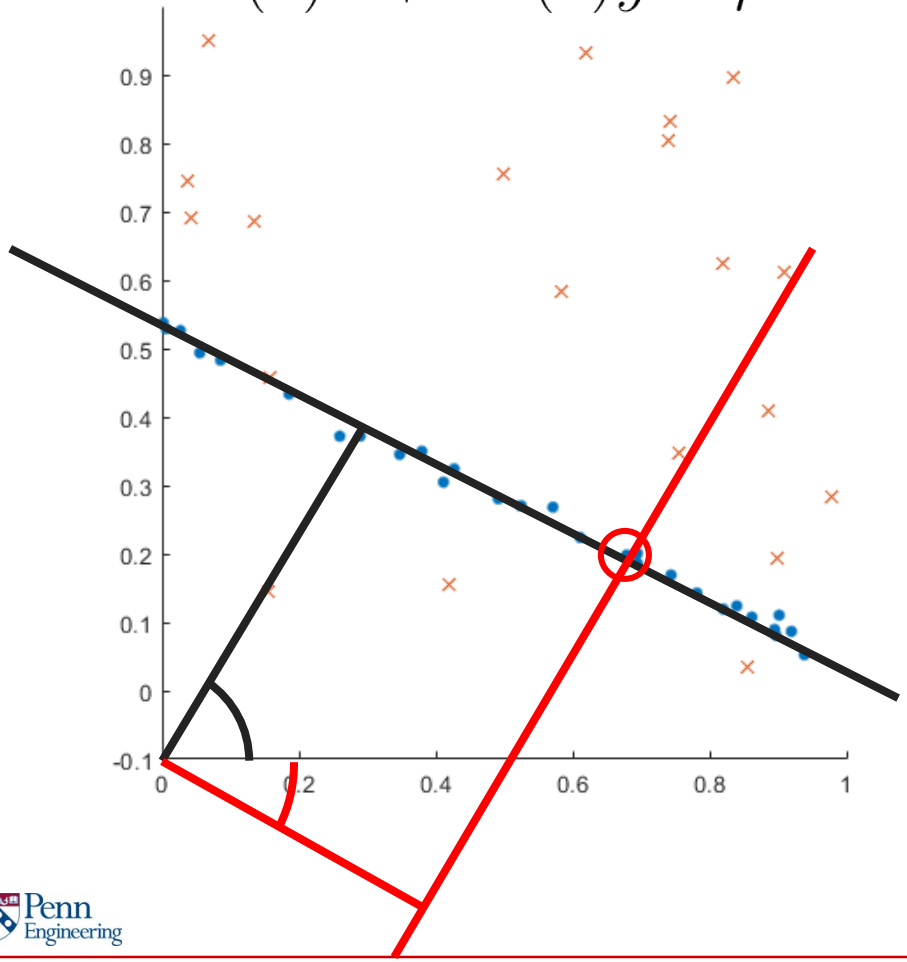
This line corresponds to a point in parameter space

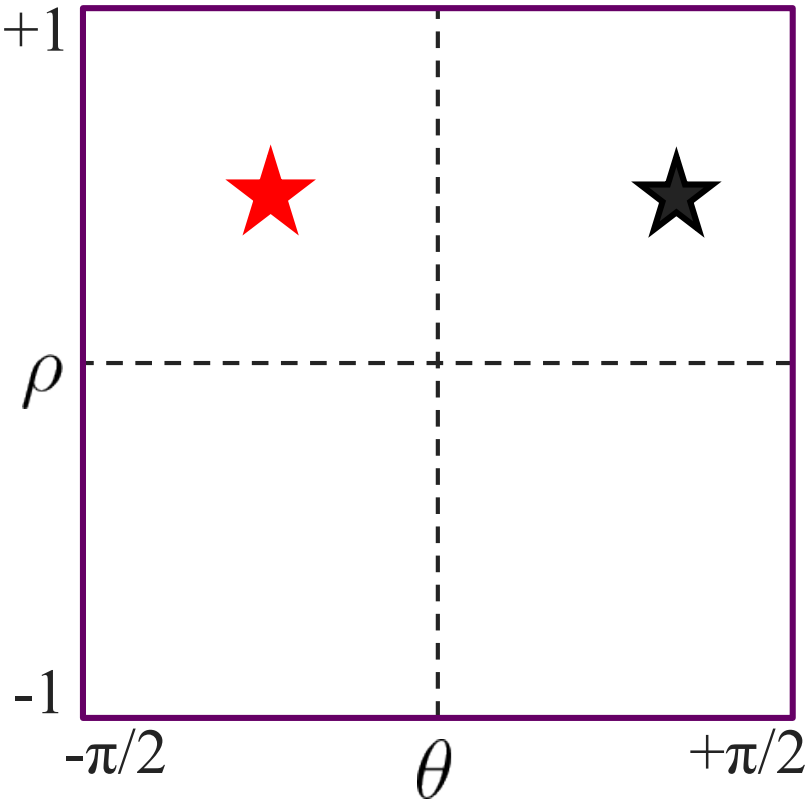


Hough Transform: Algorithm

Every point has an infinite number of lines that goes through it. Mark all those lines in parameter space

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$

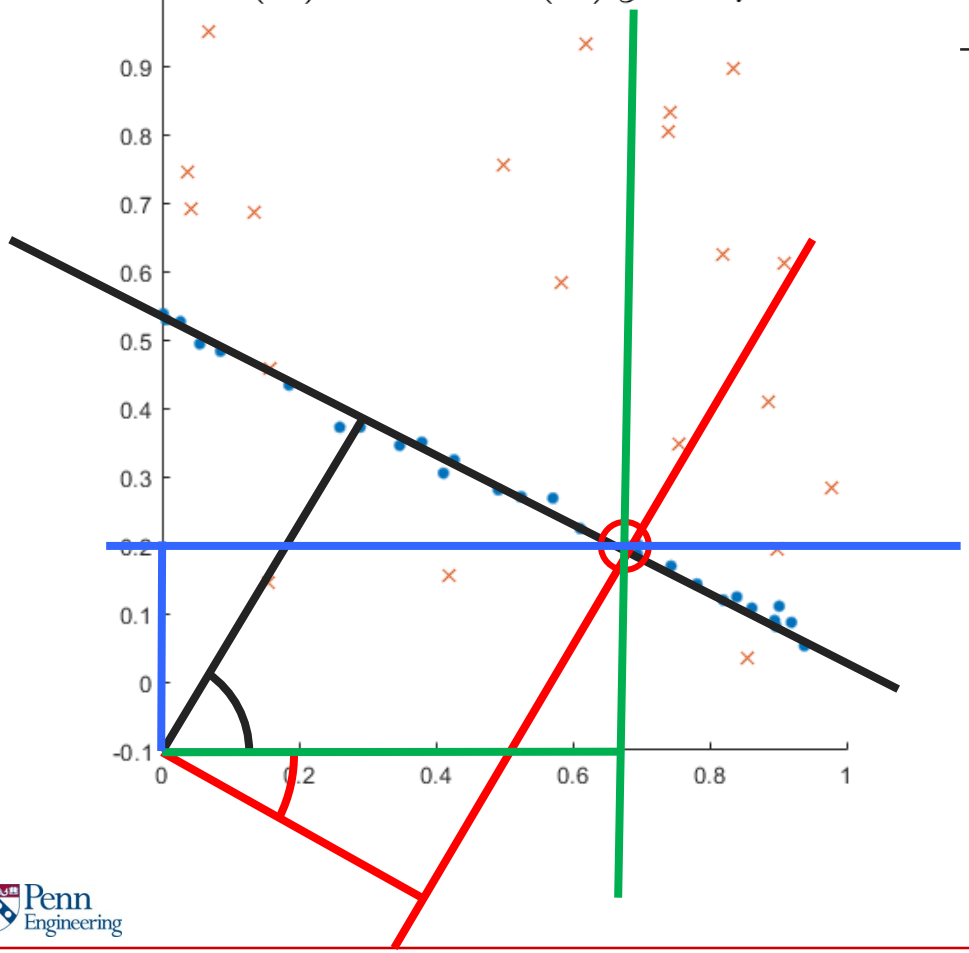


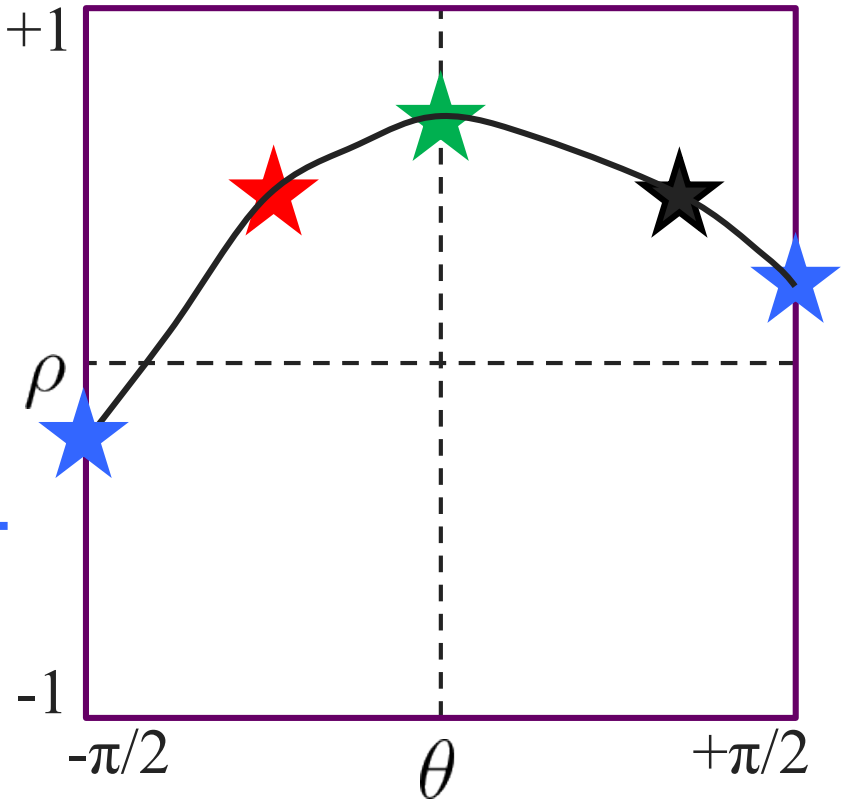


Hough Transform: Algorithm

Every point has an infinite number of lines that goes through it. Mark all those lines in parameter space

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$

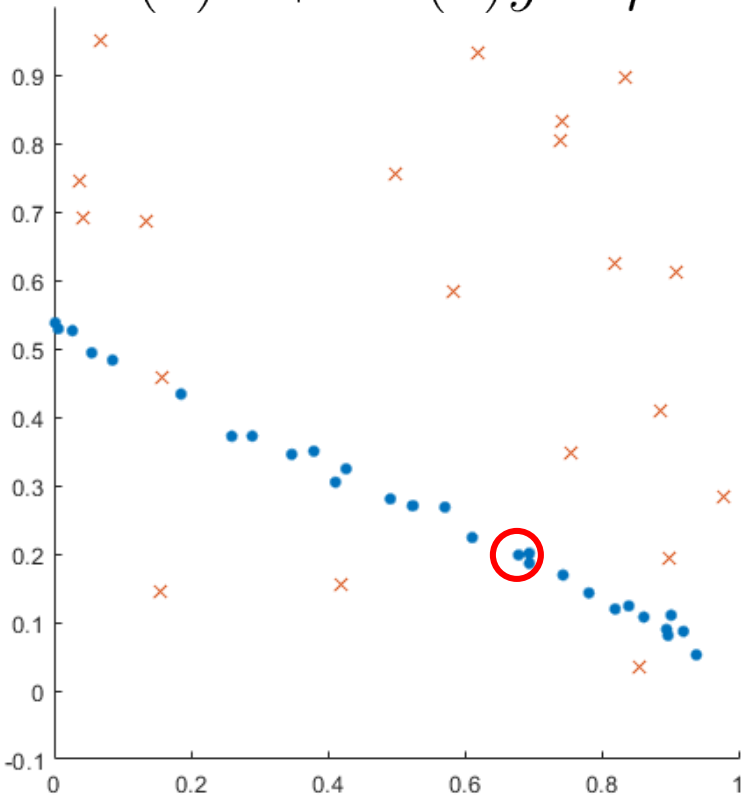


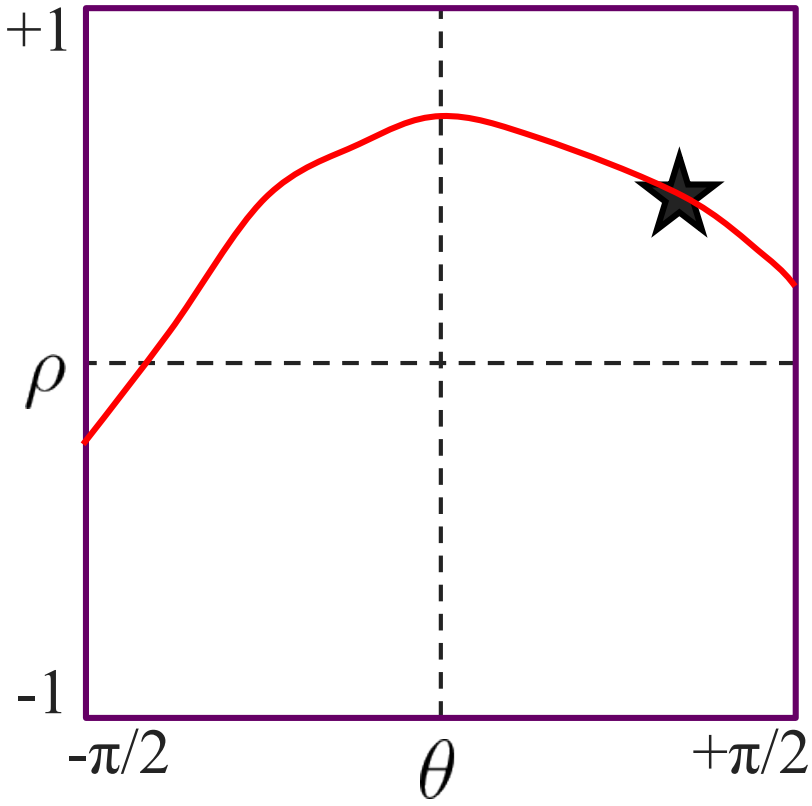


Hough Transform: Algorithm

Every point has an infinite number of lines that goes through it. Mark all those lines in parameter space

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$

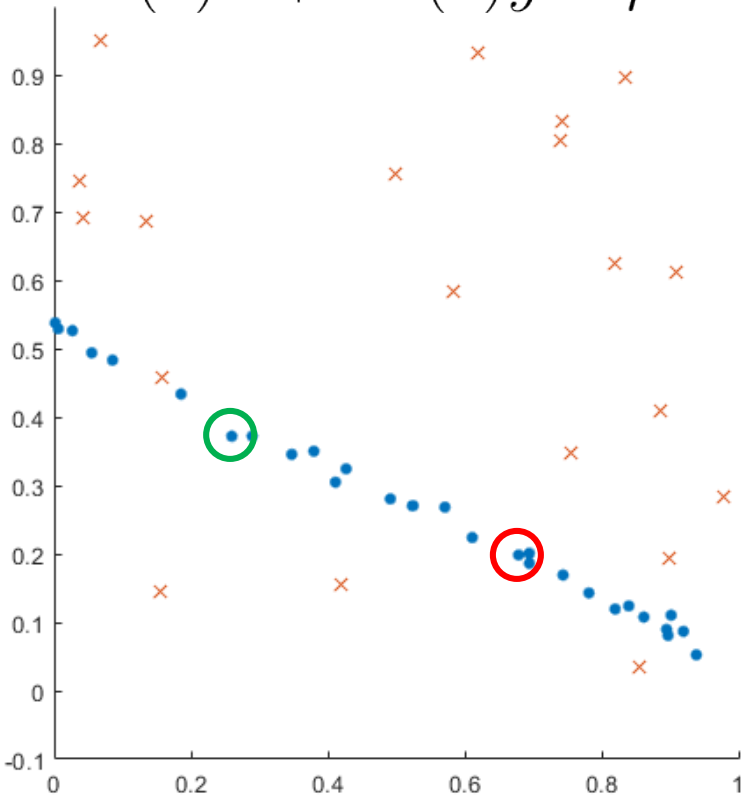


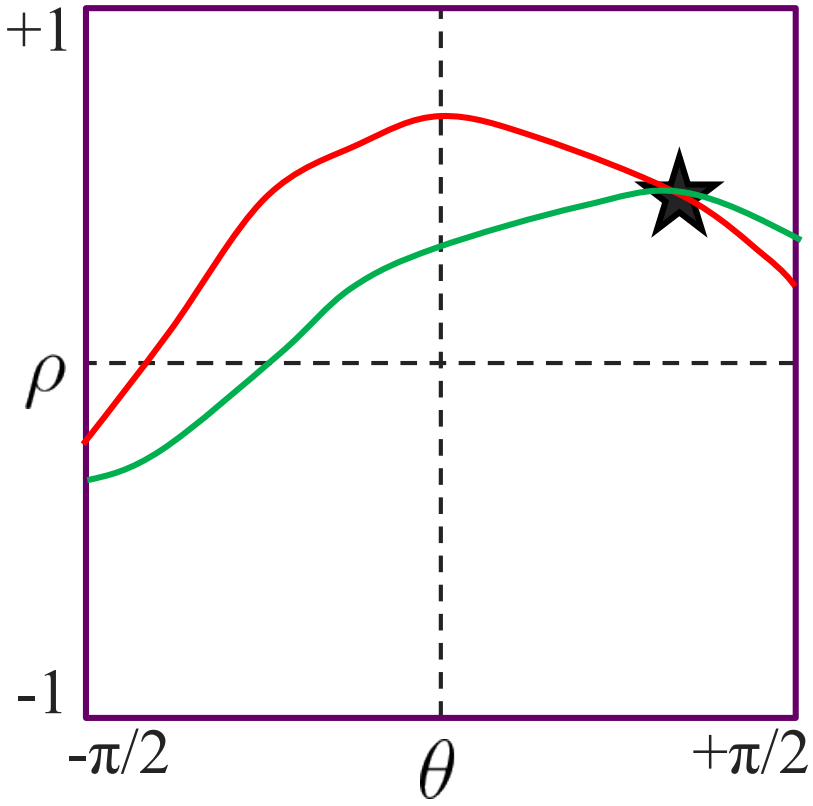


Hough Transform: Algorithm

Every point has an infinite number of lines that goes through it. Mark all those lines in parameter space

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$

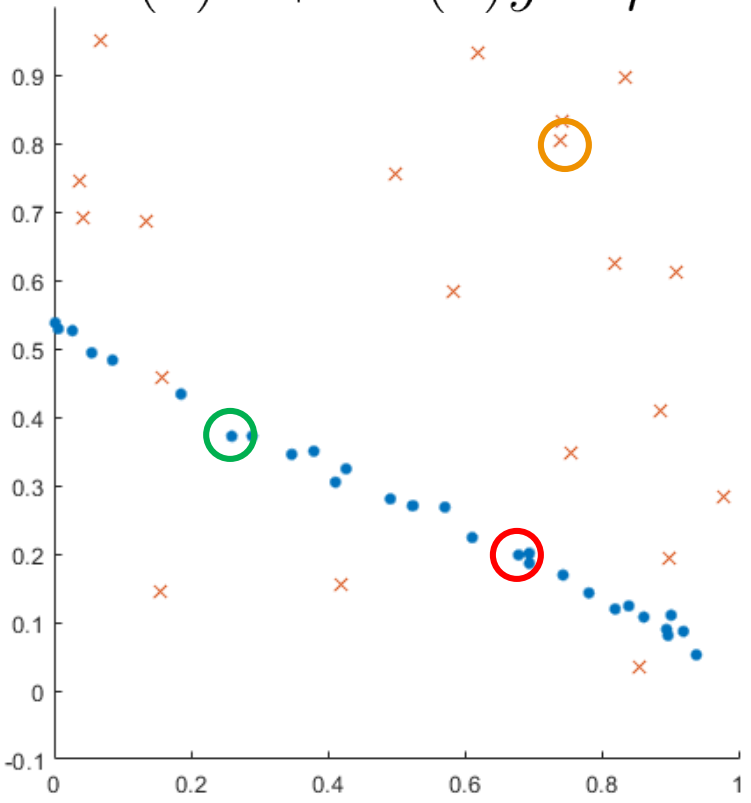




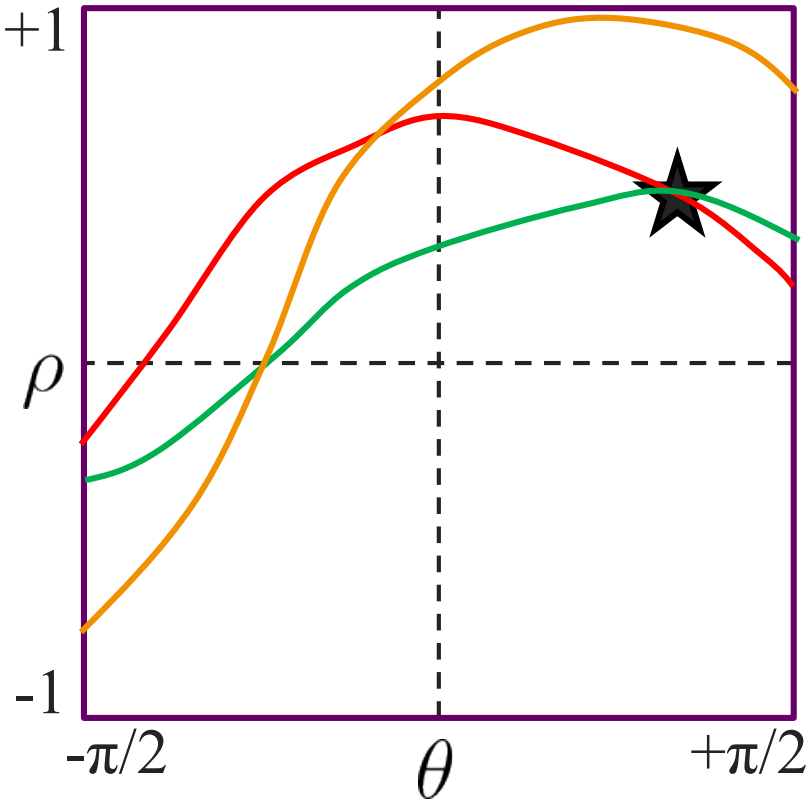
Hough Transform: Algorithm

Every point has an infinite number of lines that goes through it. Mark all those lines in parameter space

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$



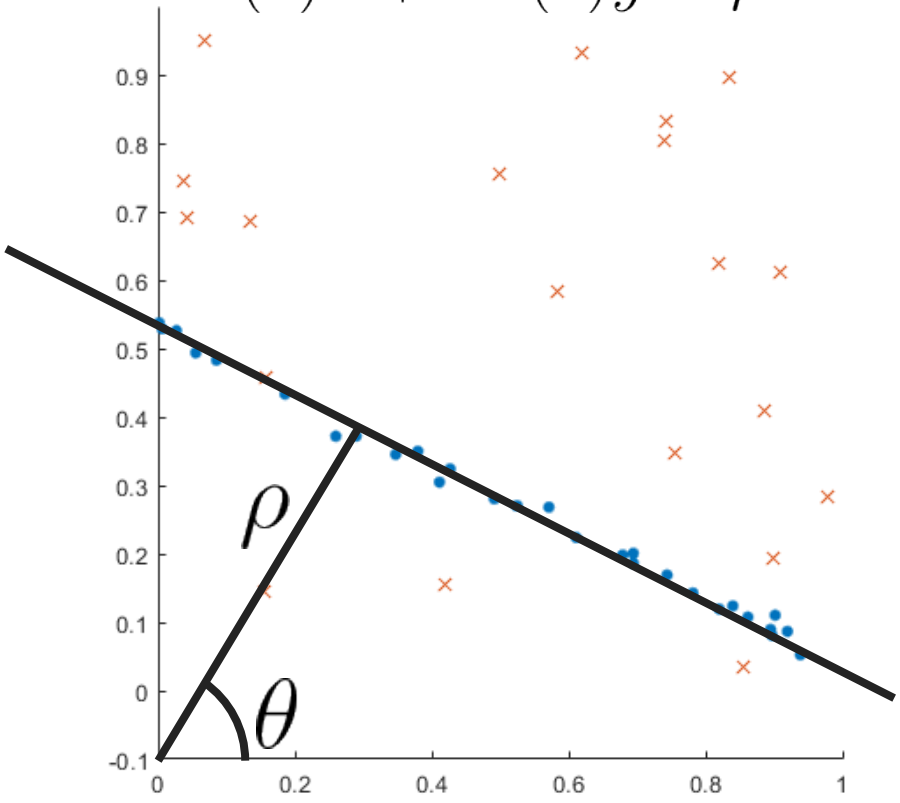
Point off the line intersects at totally different places



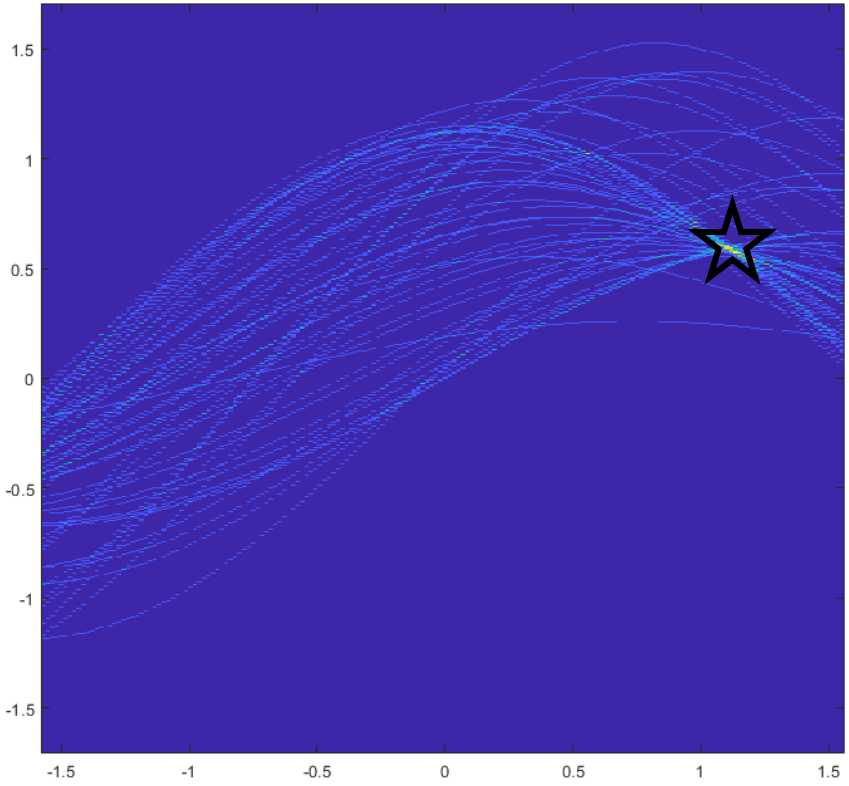
Hough Transform: Algorithm

After marking the lines that go through every point, check to see which lines has the most points 'voting' for it

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$



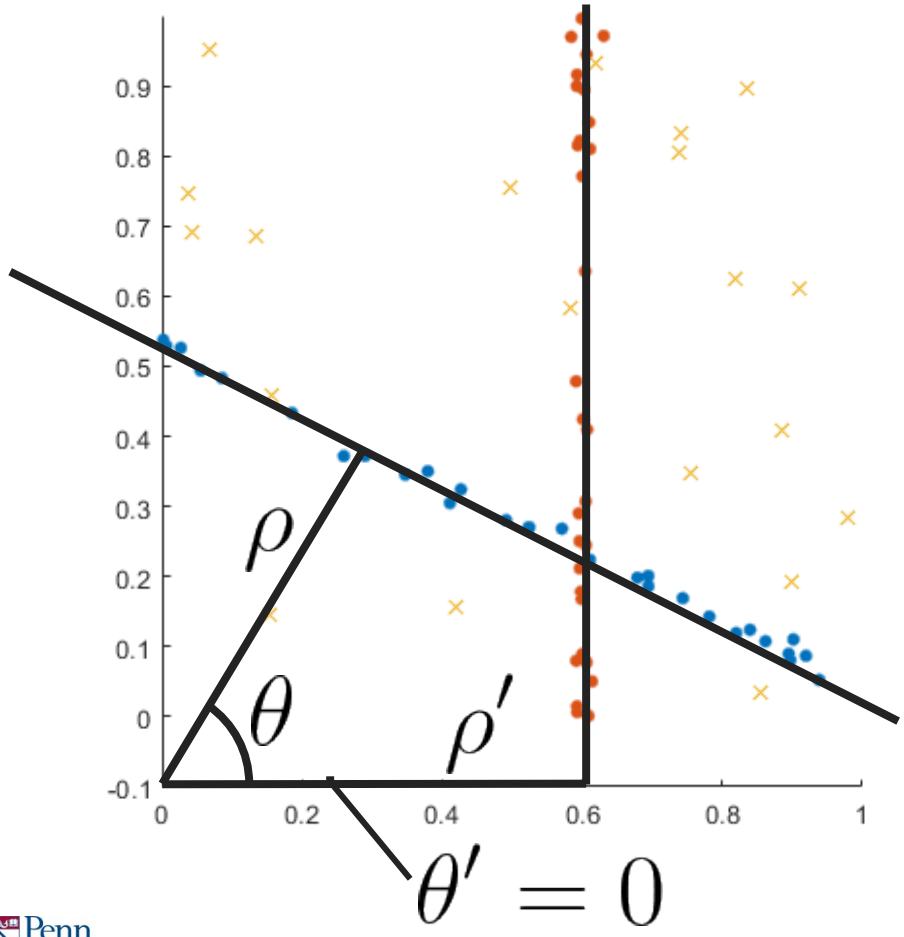
As desired, the most votes are in the right place



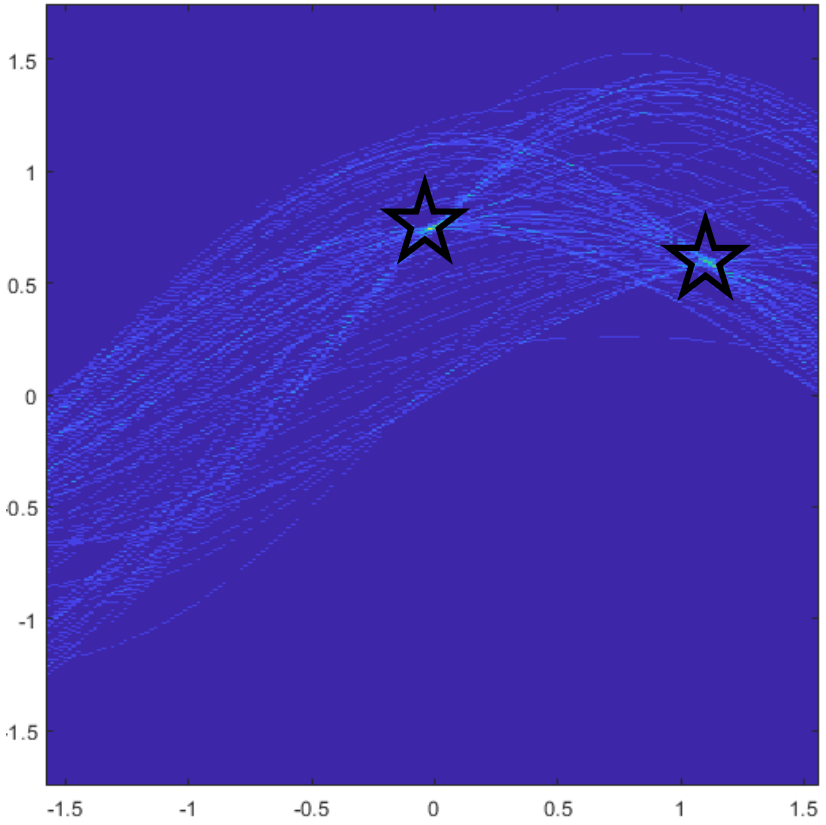
Hough Transform: Multiline Fitting

The advantage of this voting method is that you can even find multiple lines in one image

$$\cos(\theta)x + \sin(\theta)y - \rho = 0$$



Both are in the right place



Hough Transform: Downsides

While this works great for line fitting (so for instance finding edges in images) it has numerous issues:

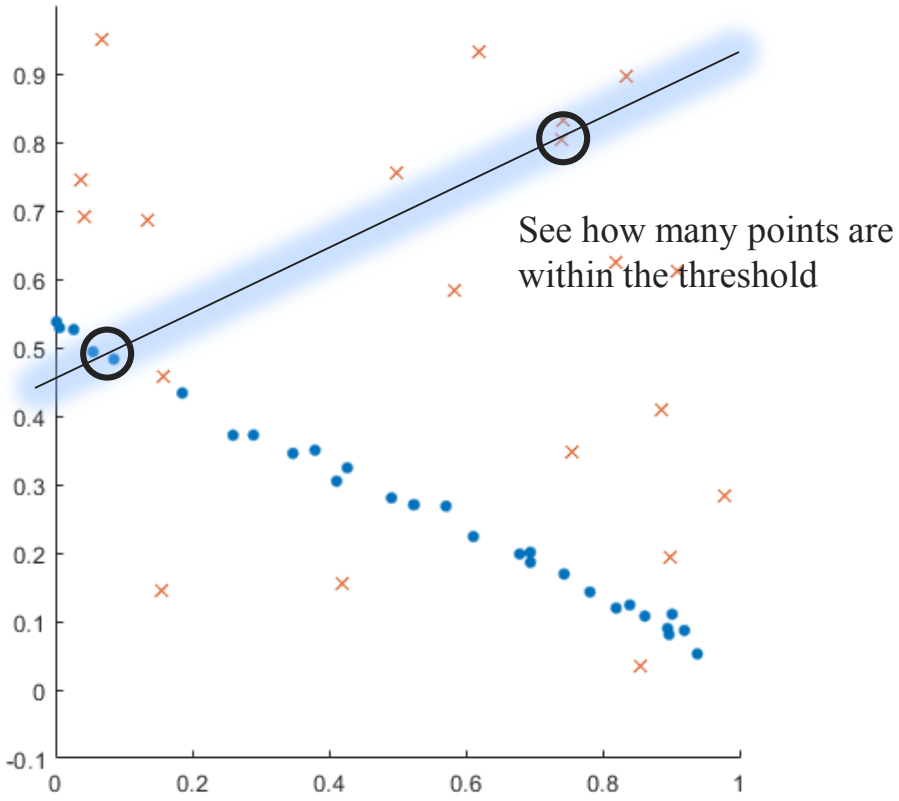
- The memory grows exponentially in the parameter space (why?)
- Need bounds on the parameter space

Is there another method that can overcome these problems?

Sample Consensus

Instead of try to store all the parameters, let's just try to fit all possible minimal models on our data and see which has the most inliers

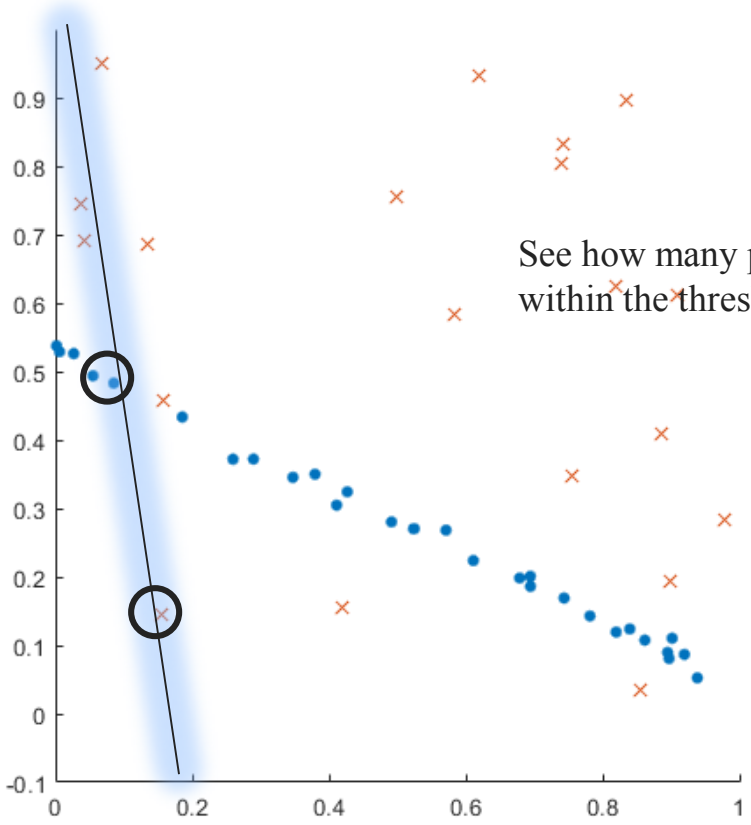
Inlier Count: 4



Sample Consensus

Instead of try to store all the parameters, let's just try to fit all possible minimal models on our data and see which has the most inliers

Inlier Count: 7

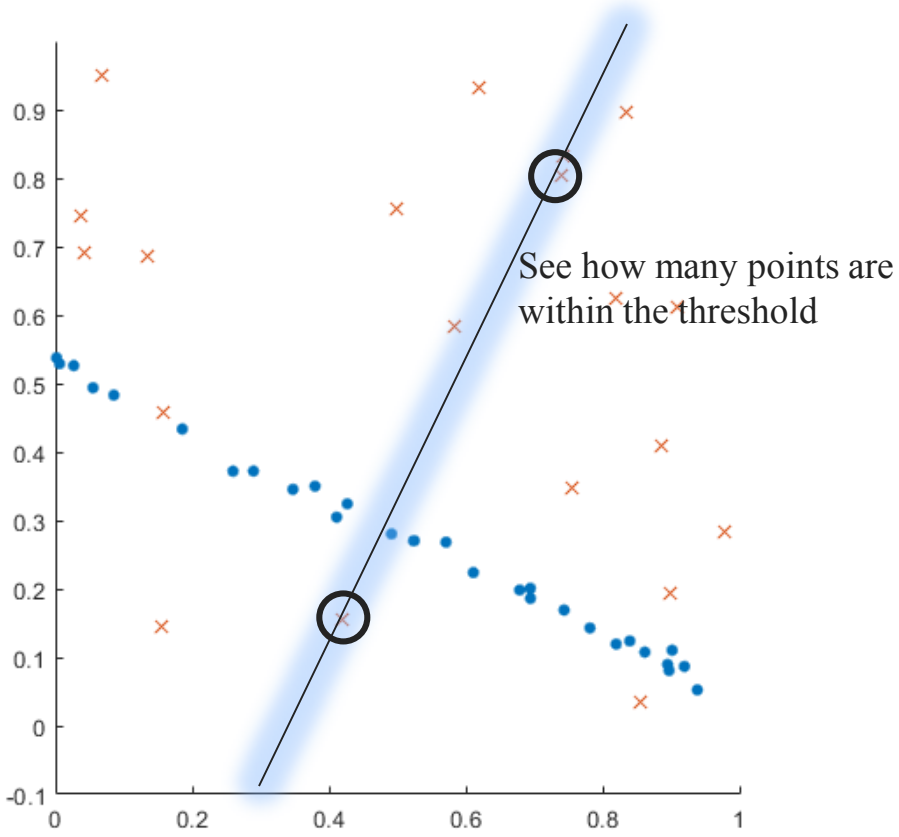


See how many points are within the threshold

Sample Consensus

Instead of try to store all the parameters, let's just try to fit all possible minimal models on our data and see which has the most inliers

Inlier Count: 7

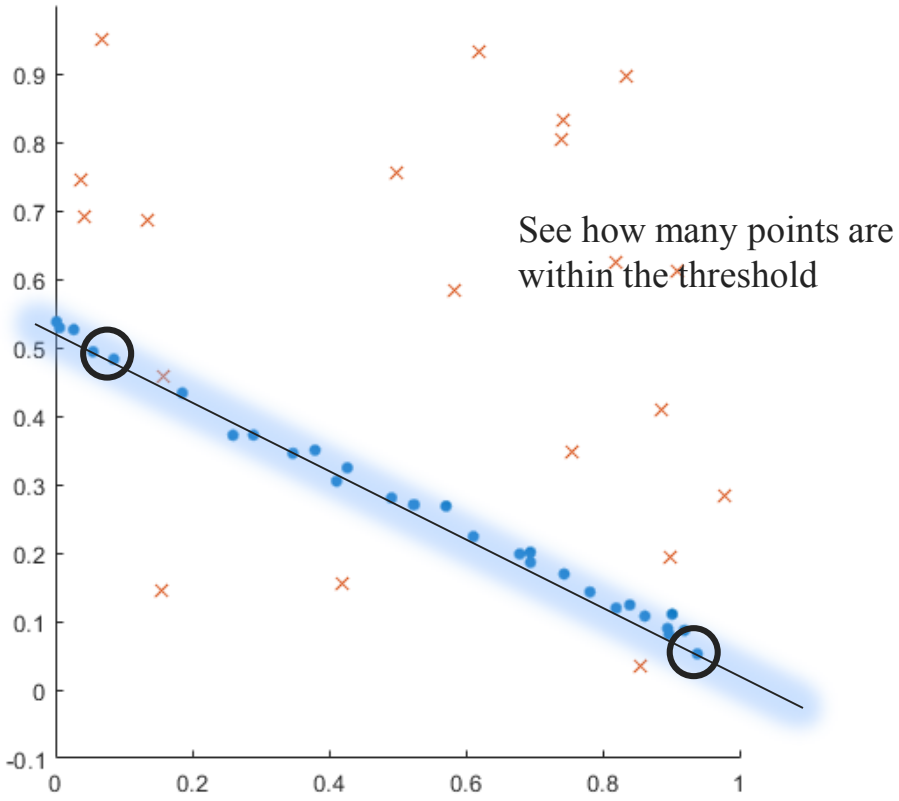


Sample Consensus

Instead of try to store all the parameters, let's just try to fit all possible minimal models on our data and see which has the most inliers

Inlier Count: 29

Better than any of the others so we pick this one



See how many points are within the threshold

Sample Consensus

Upsides:

- Simple

Downsides

- Computation grows combinatorically with the number of parameters of the minimal model

Line: $\binom{n}{2}$ Homography: $\binom{n}{4}$ Essential Matrix: $\binom{n}{5}$

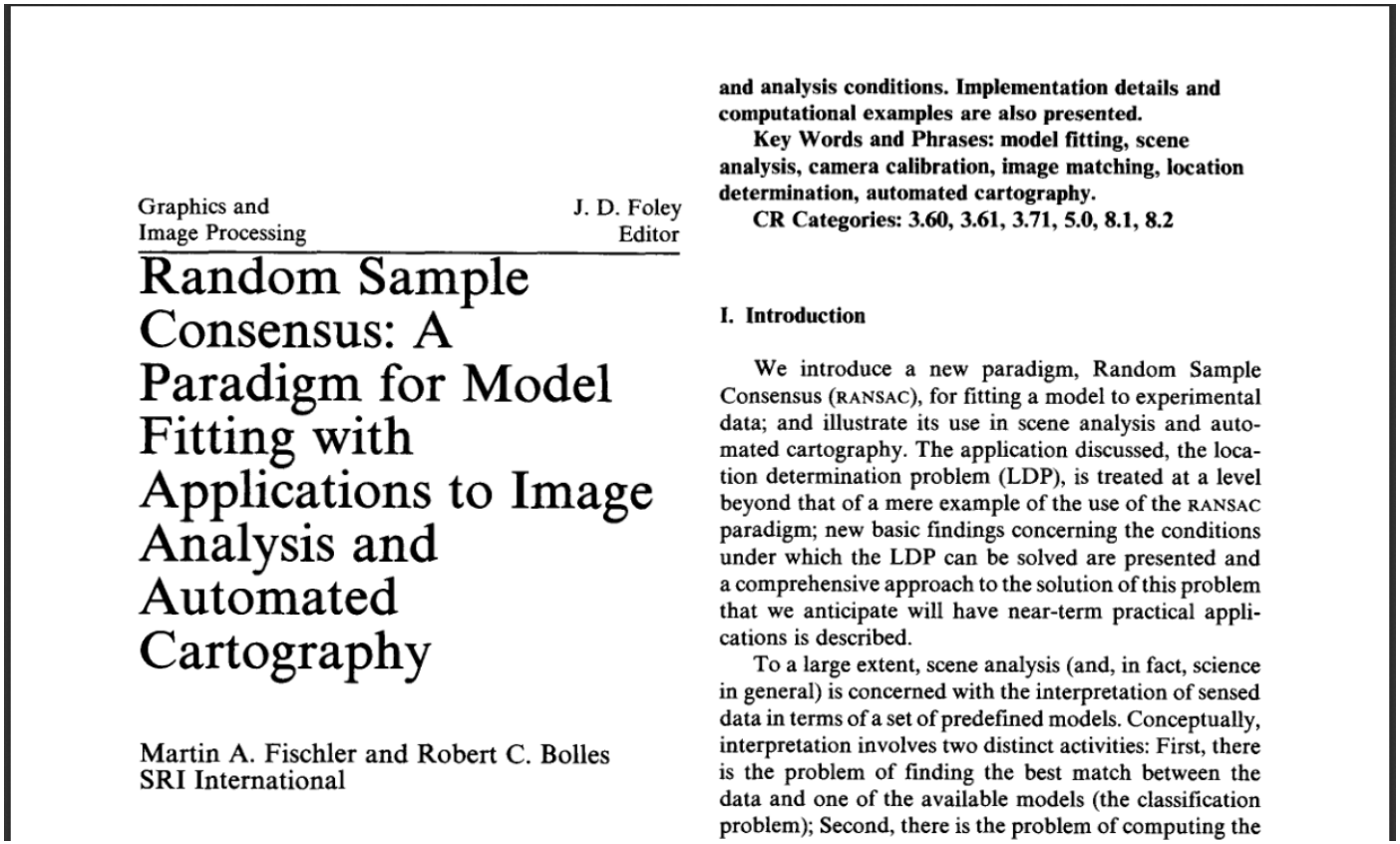
Model	10 points	100 points	1000 points
Line	45	4950	499500
Homography	210	3921225	4.14×10^{10}
Essential Matrix	252	75287520	8.25×10^{12}

Number of models to estimate by number of points

RANSAC: Random Sample Consensus

Popularly used in vision since the 1980s

It is just sample consensus except you pick samples randomly and you stop after *k* iterations



Graphics and Image Processing — J. D. Foley Editor

Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography

Martin A. Fischler and Robert C. Bolles
SRI International

and analysis conditions. Implementation details and computational examples are also presented.

Key Words and Phrases: model fitting, scene analysis, camera calibration, image matching, location determination, automated cartography.
CR Categories: 3.60, 3.61, 3.71, 5.0, 8.1, 8.2

I. Introduction

We introduce a new paradigm, Random Sample Consensus (RANSAC), for fitting a model to experimental data; and illustrate its use in scene analysis and automated cartography. The application discussed, the location determination problem (LDP), is treated at a level beyond that of a mere example of the use of the RANSAC paradigm; new basic findings concerning the conditions under which the LDP can be solved are presented and a comprehensive approach to the solution of this problem that we anticipate will have near-term practical applications is described.

To a large extent, scene analysis (and, in fact, science in general) is concerned with the interpretation of sensed data in terms of a set of predefined models. Conceptually, interpretation involves two distinct activities: First, there is the problem of finding the best match between the data and one of the available models (the classification problem); Second, there is the problem of computing the

RANSAC: Random Sample Consensus

The algorithm (in pseudocode):

```
Repeat for k iterations
1.  Choose a minimal sample set
2.  Count the inliers for this set
3.  Keep maximum, if it exceeds a desired number of inliers
stop.
```

This fixes the problem with the number of iterations growing too fast, but how do we know we will actually hit an inlier set?

As this is a random algorithm, we will need to do probabilistic analysis to see how it works

Probability of Hitting Inliers

Say the proportion of inliers is ϵ, and the minimum number of points for your model is *M*. Then the chance of hitting all inliers is ϵ^M

Thus the chance after *k* iterations of you NEVER hitting an inlier set is: $(1-\epsilon^M)^k$

Thus the probability of hitting at least one inlier set, i.e. the probability of success, is:

$$p_{success} = 1 - (1-\epsilon^M)^k$$

So if you want a given chance of success:

$$\Longrightarrow k = \frac{\log(1-p_{success})}{\log(1-\epsilon^M)}$$

Probability of Hitting Inliers

$$k = \frac{\log(1 - p_{success})}{\log(1 - \epsilon^M)}$$

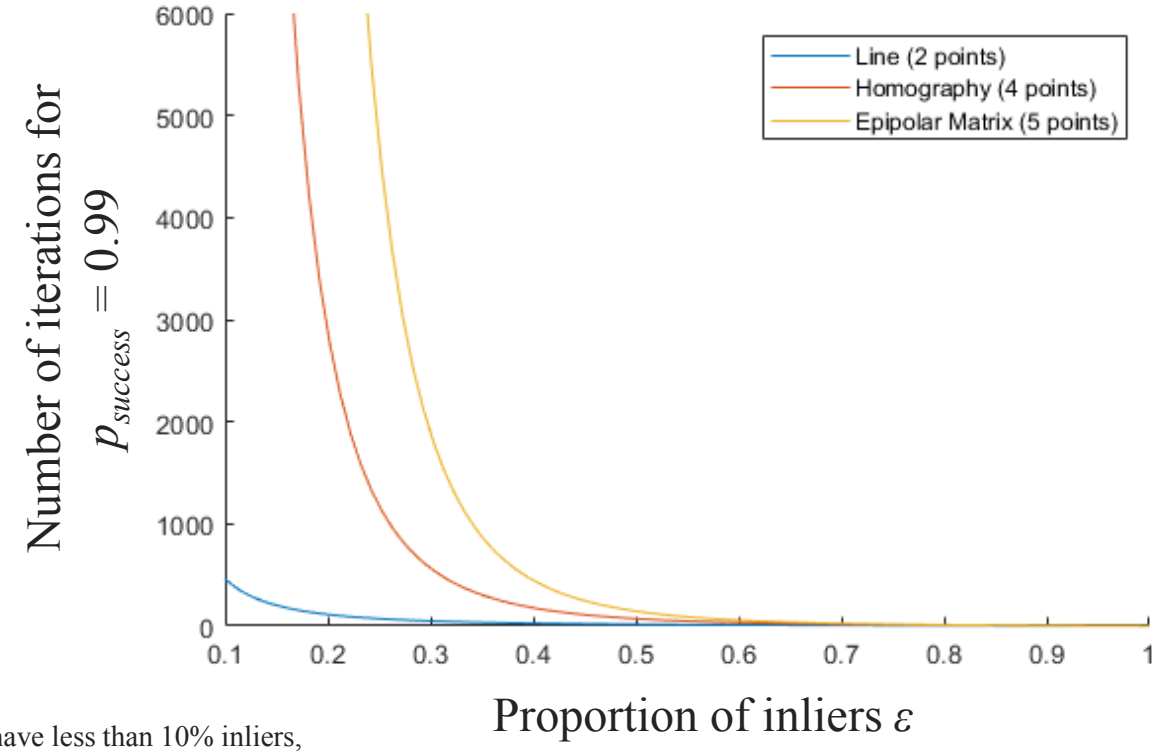
For $p_{success} = 0.99$ the k needed for various models:

Model	$\varepsilon = 0.8$	$\varepsilon = 0.5$	$\varepsilon = 0.3$	$\varepsilon = 0.1$
Line (2 points)	5	17	49	459
Homography (4 points)	9	72	567	46050
Essential Matrix (5 points)	12	146	1893	460515

Probability of Hitting Inliers

$$k = \frac{\log(1 - p_{success})}{\log(1 - \epsilon^M)}$$

For $p_{success} = 0.99$ the k needed for various models:



If you have less than 10% inliers, maybe you should just work on that

Next Time

In previous lectures we learned how to estimate position and orientation from images. Now we want to be able to measure velocity. We have most of the tools already, we just need to get a model of how motion on the image relates to motion in the 3D world