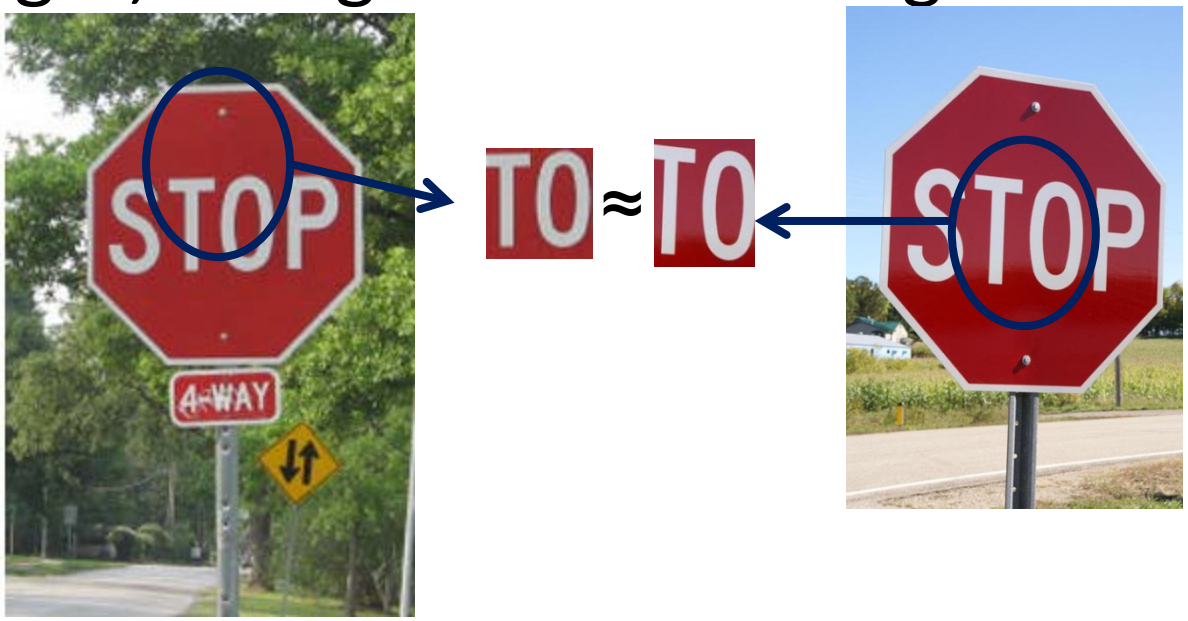
From Edges to Lines

Vinay P. Namboodiri

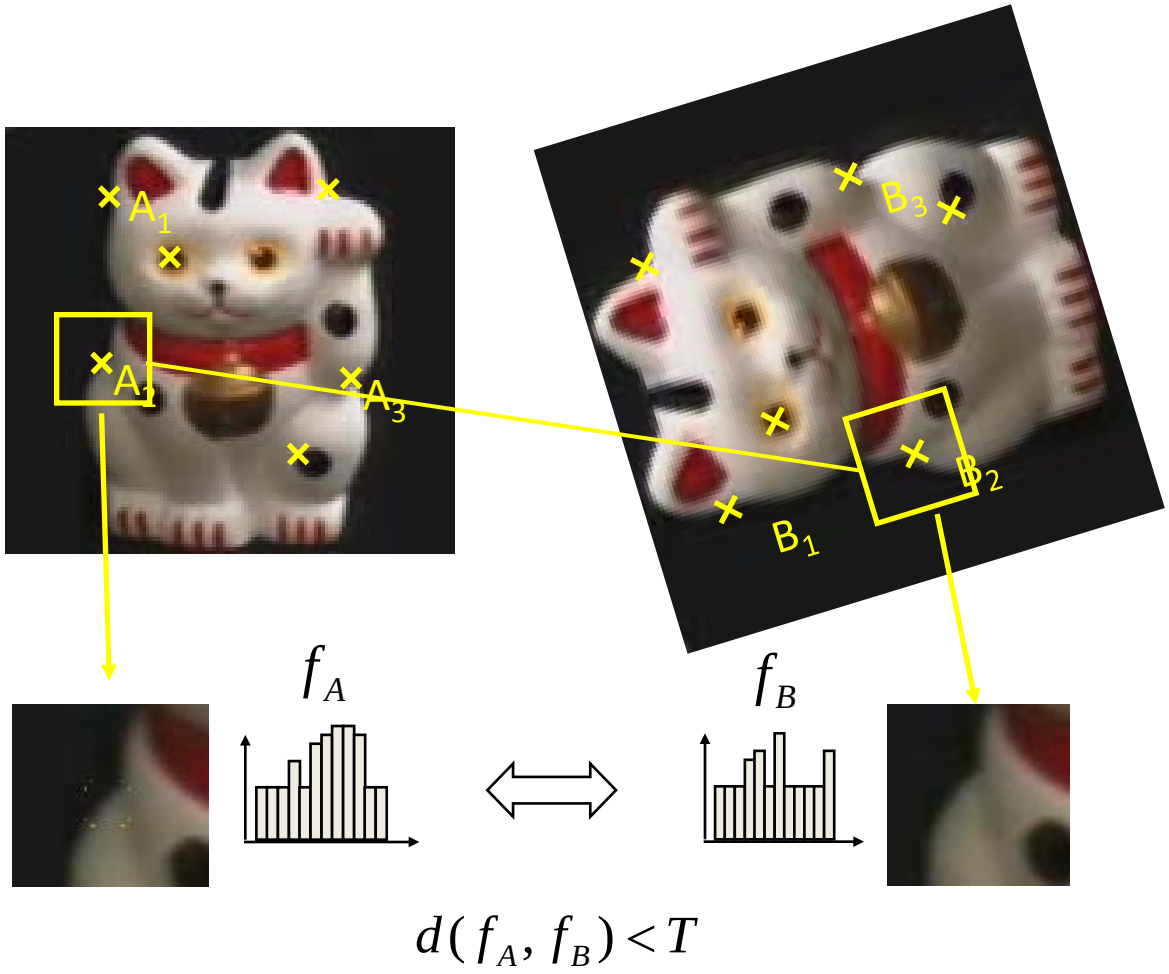
- Slide credit to James Hays, Derek Hoiem and Grauman&Leibe, Marshall Tappen

This section: correspondence and alignment

- Correspondence: matching points, patches, edges, or regions across images



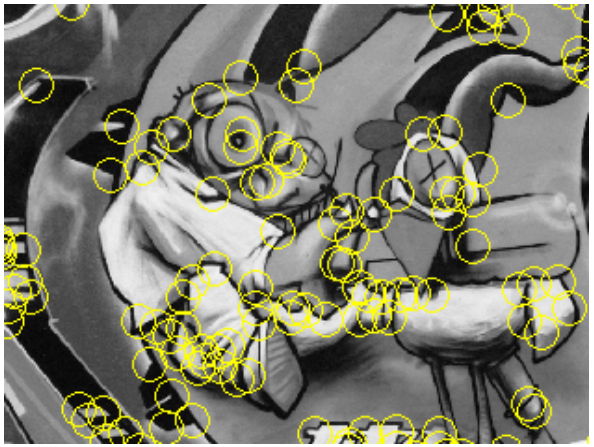
Overview of Keypoint Matching



1. **Find a set of distinctive key-points**
2. **Define a region around each keypoint**
3. **Extract and normalize the region content**
4. **Compute a local descriptor from the normalized region**
5. **Match local descriptors**

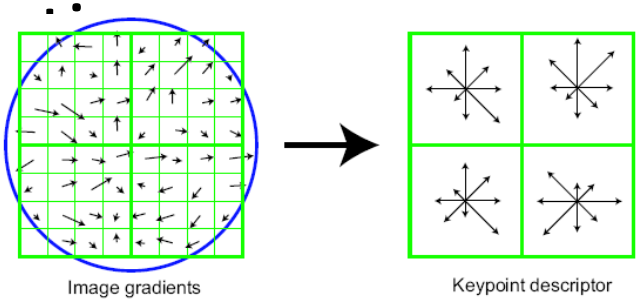
Review: Interest points

- Keypoint detection: repeatable and distinctive
  - Corners, blobs, stable regions
  - Harris, DoG, MSER

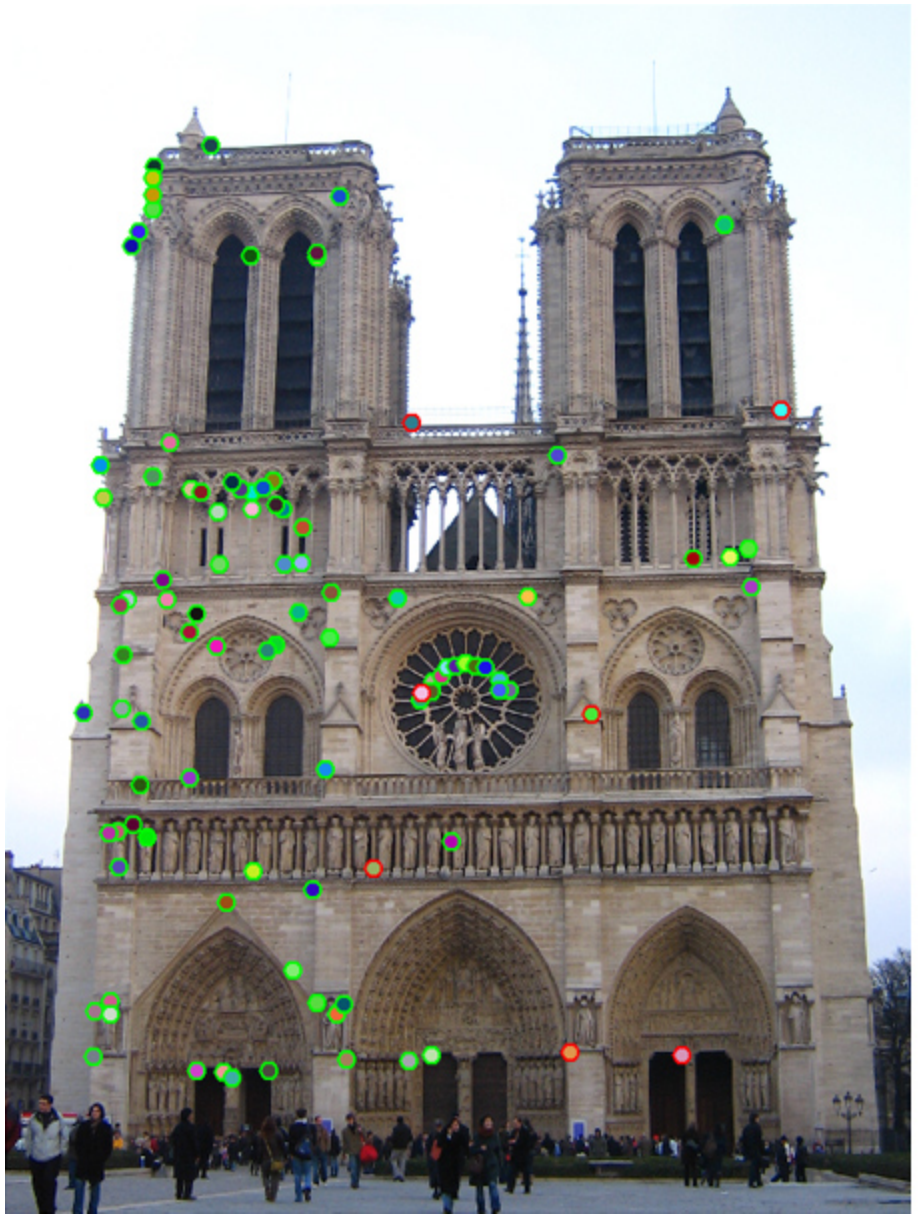


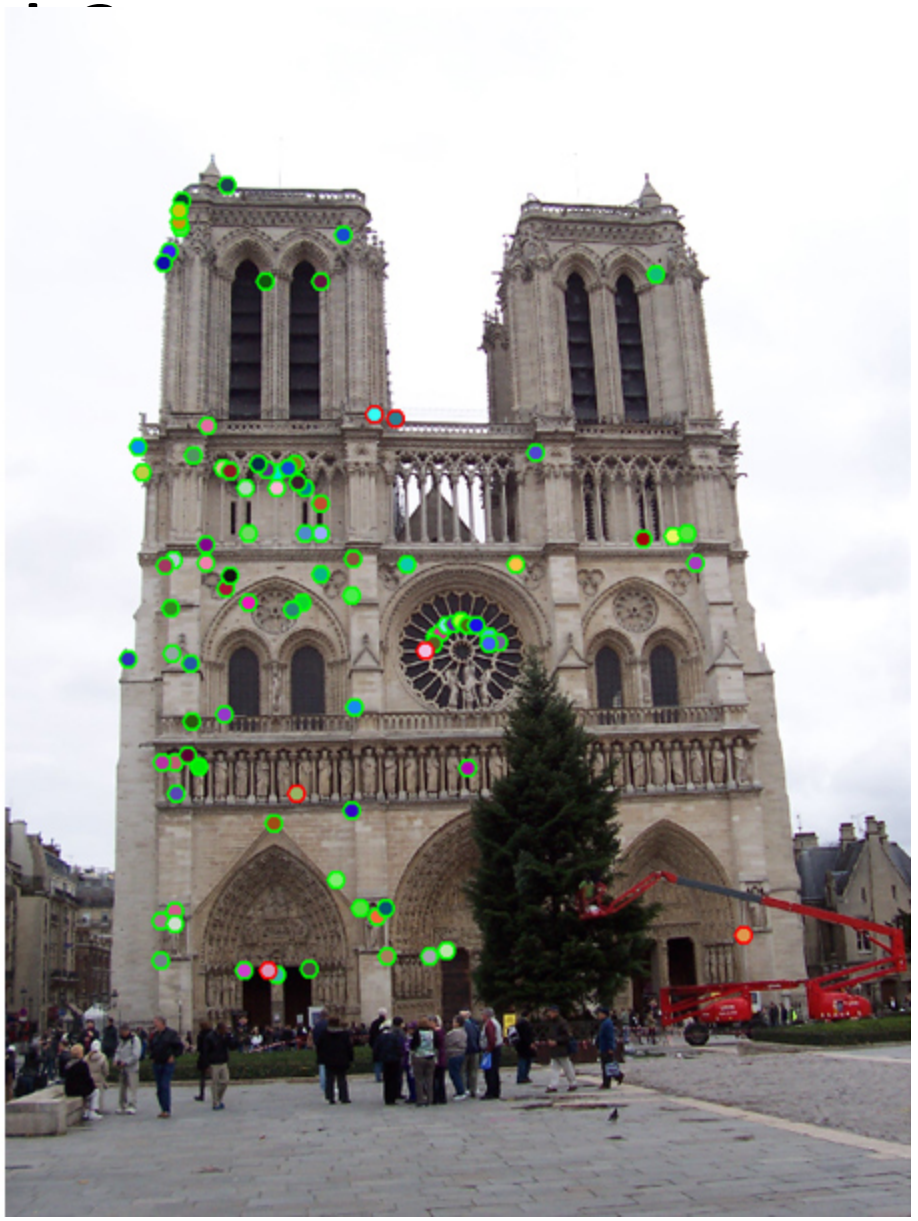
Review: Local Descriptors

- Most features can be thought of as templates, histograms (counts), or combin
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient
- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used



How do we decide which features



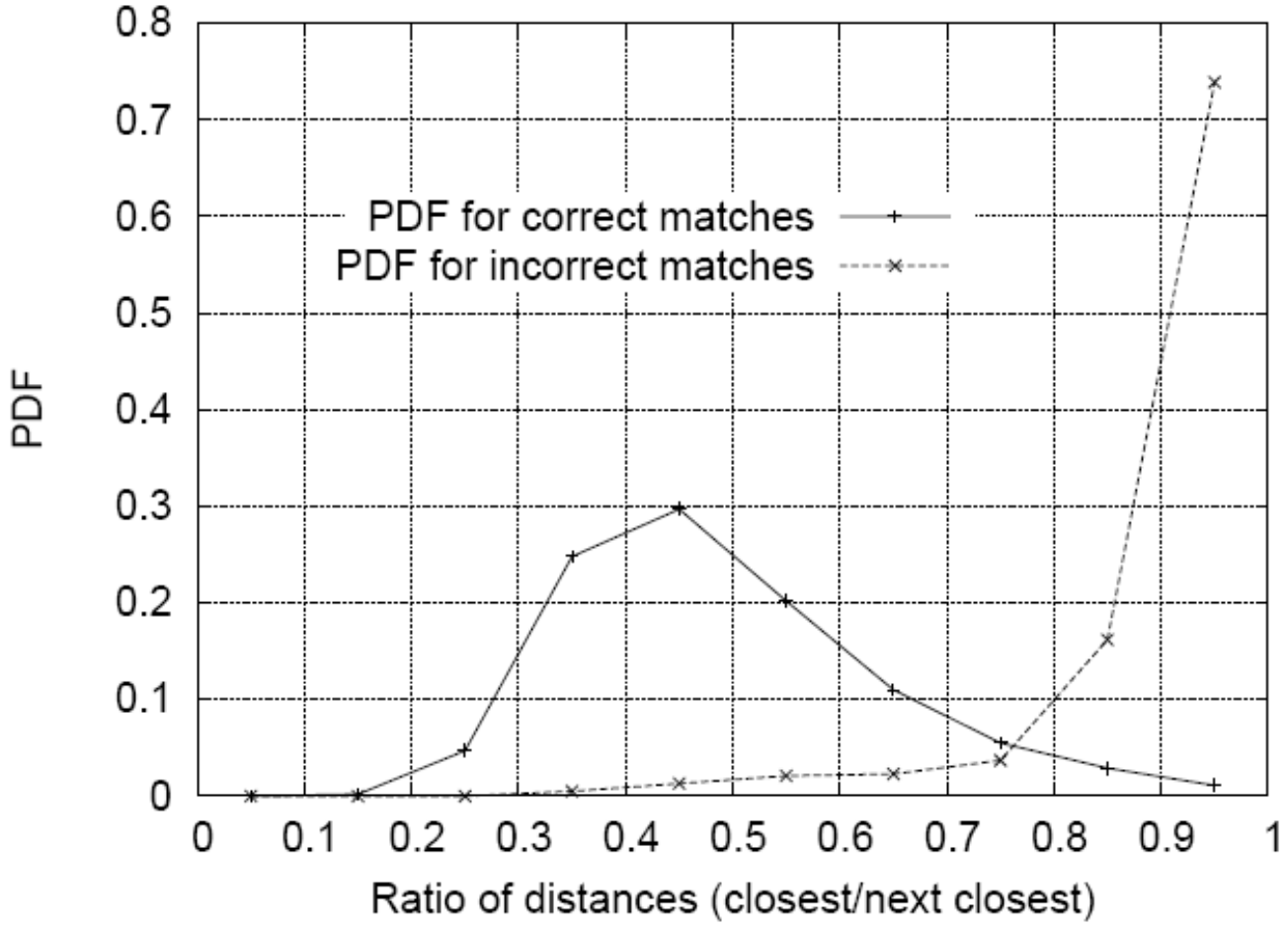


Feature Matching

- Simple criteria: One feature matches to another if those features are nearest neighbors and their distance is below some threshold.
- Problems:
  - Threshold is difficult to set
  - Non-distinctive features could have lots of close matches, only one of which is correct

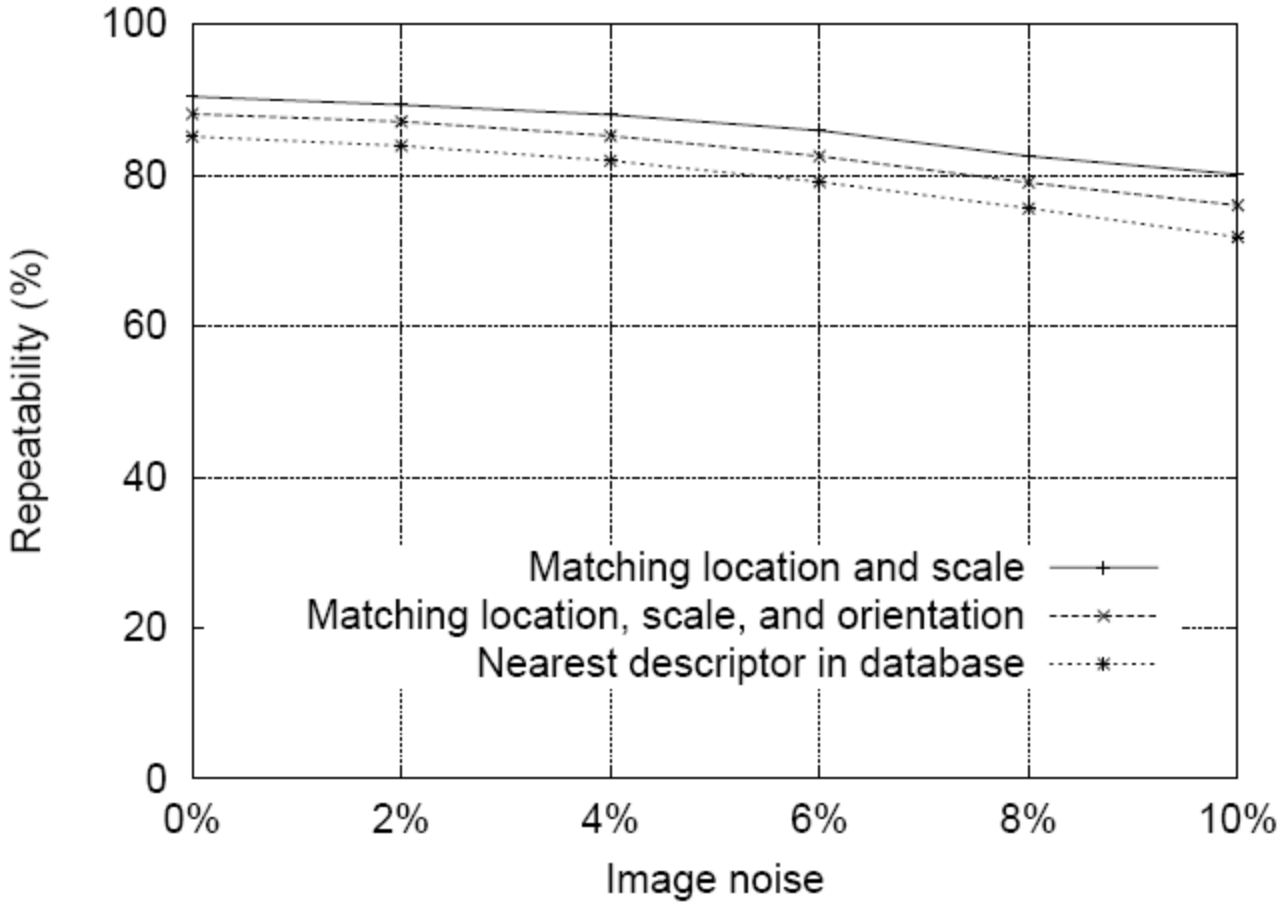
Matching Local Features

- Threshold based on the ratio of 1st nearest neighbor



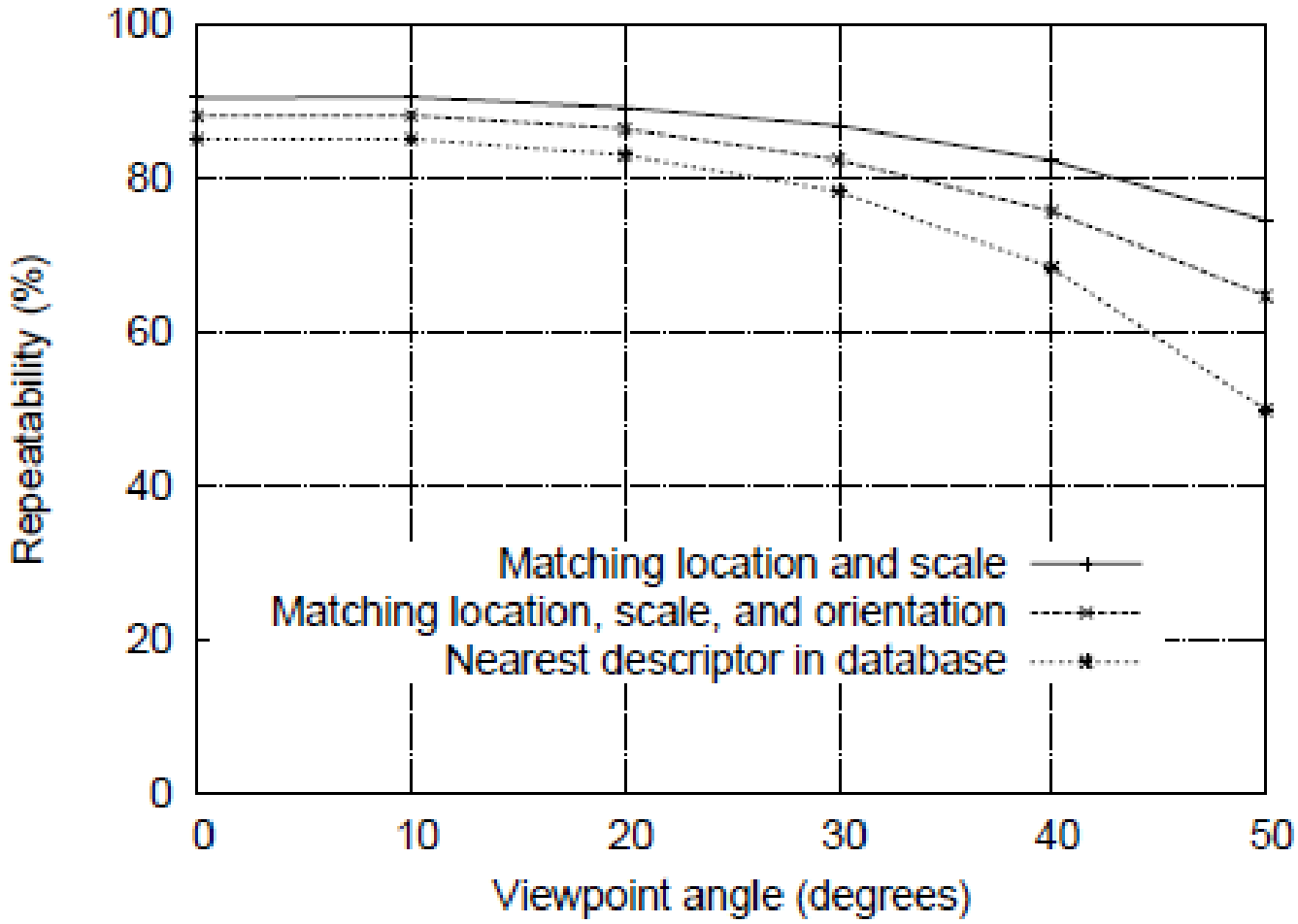
Lowe IJCV 2004

SIFT Repeatability

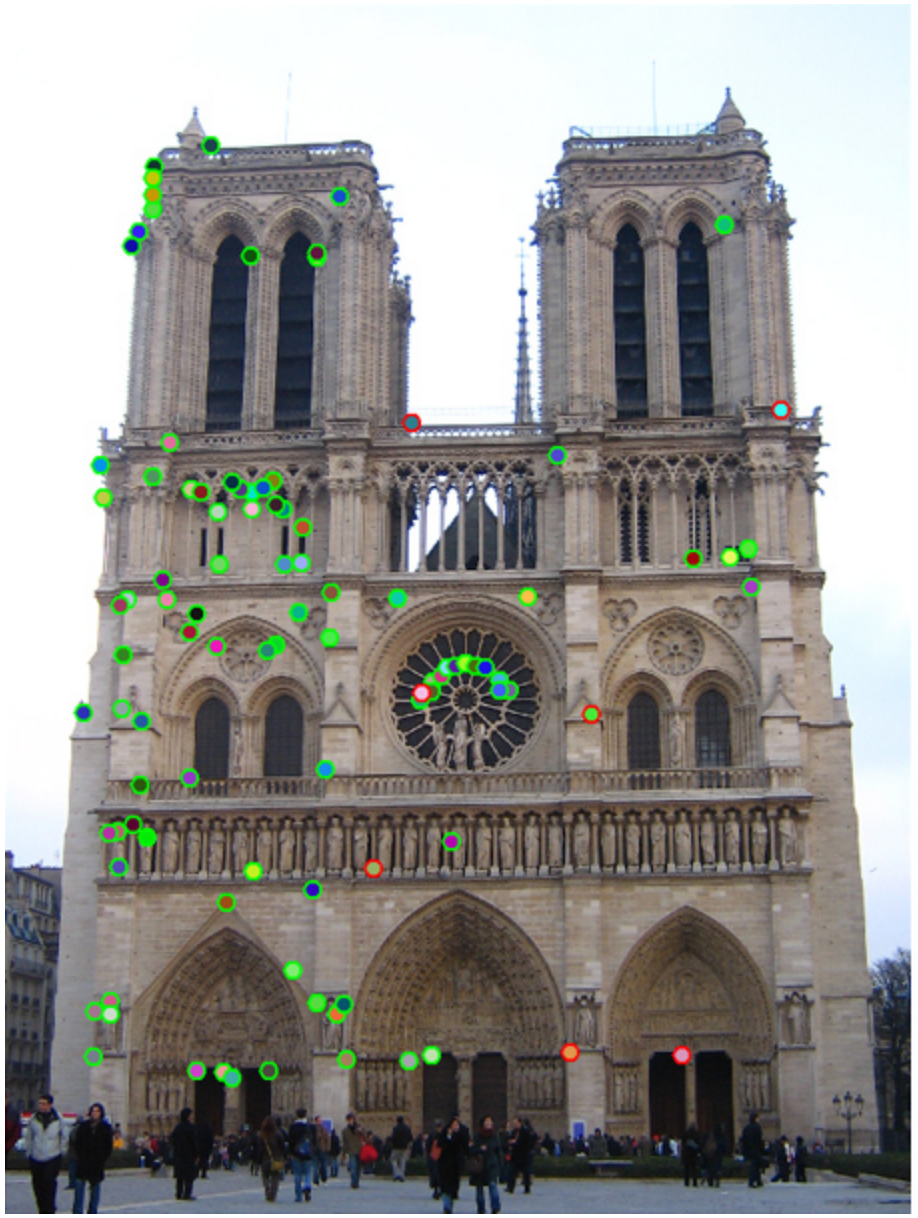


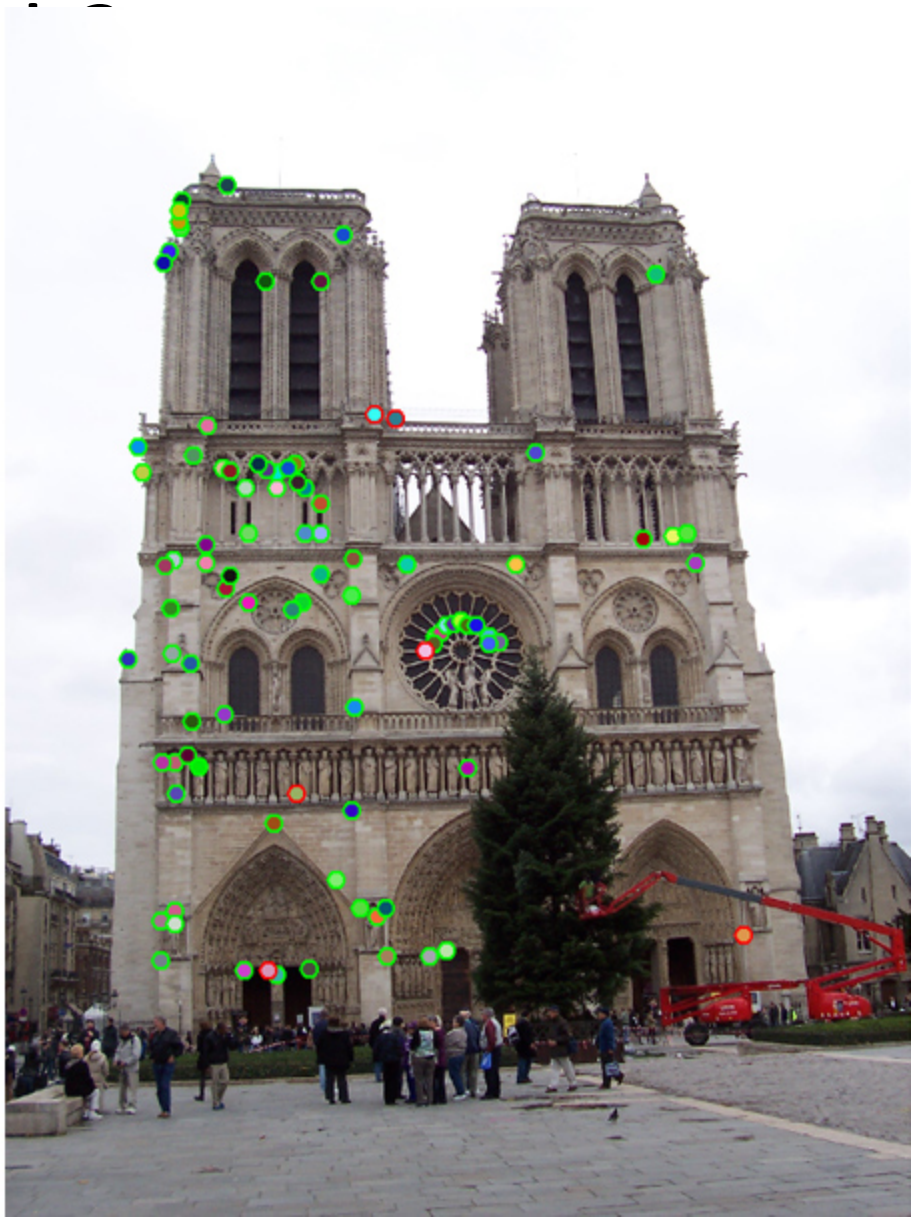
Lowe IJCV 2004

SIFT Repeatability



How do we decide which features





Fitting: find the parameters of a model that best fit the data

Alignment: find the parameters of the transformation that best align matched points

Fitting and Alignment

- Design challenges
  - Design a suitable **goodness of fit** measure
    - Similarity should reflect application goals
    - Encode robustness to outliers and noise
  - Design an **optimization** method
    - Avoid local optima
    - Find best parameters quickly

Fitting and Alignment: Methods

- Global optimization / Search for parameters
  - Least squares fit
  - Robust least squares
  - Iterative closest point (ICP)
- Hypothesize and test
  - Generalized Hough transform
  - RANSAC

Simple example: Fitting a line

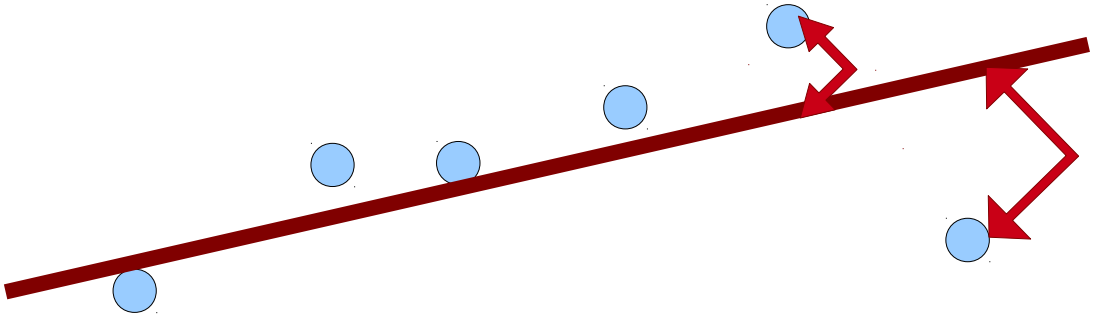
From Edges to Lines

We've talked about detecting Edges, but how can we extract lines from those edges?

We'll start with a very basic, least squares approach

Mathematically

$$L(m,b) = \sum_{i=1}^{N} (mx_i + b - y_i)^2$$



See why it's called least squares?

Review: Line Fitting

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_{N-1} & 1 \\ x_N & 1 \end{bmatrix} \quad \mathbf{m} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Review

So, we can write the prediction at multiple points as

$$\hat{\mathbf{y}} = A\mathbf{m}$$

$$A = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_{N-1} & 1 \\ x_N & 1 \end{bmatrix} \qquad \mathbf{m} = \begin{bmatrix} m \\ b \end{bmatrix}$$

Going back

Notice that

$$\frac{\partial L(m,b)}{\partial m} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right) x_i$$

Can be rewritten as

$$A(:,1)^T(A\mathbf{m} - y)$$

(I'm using MATLAB notation for A(:,1))

Sketch it Out

$$A(:,1)^T(A\mathbf{m} - y)$$

$$\frac{\partial L(m,b)}{\partial m} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)x_i$$

Now we can do the same thing again

Notice that

$$\frac{\partial L(m,b)}{\partial b} = \sum_{i=1}^{N} 2\left(mx_i + b - y_i\right)$$

Can be rewritten as

$$A(:,2)^T(A\mathbf{m} - y)$$

(I'm using MATLAB notation for A(:,2))

Now, setting everything to zero

$$A^T(A\mathbf{m} - y) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A^T A\mathbf{m} = A^T\mathbf{y}$$

$$\mathbf{m} = \left(A^T A\right)^{-1} A^T\mathbf{y}$$

Also called pseudo-inverse

You will often see this appear as

$$\mathbf{x} = \left(A^T A\right)^{-1} A^T \mathbf{h}$$

Why all the matrix hassle?

This will allow us to easily generalize to higher-dimensions

This is also called regression

- Can think of this as trying to predict *y* from *x*
- Classification and regression are two fundamental problems in pattern recognition
- What about non-linear regression?

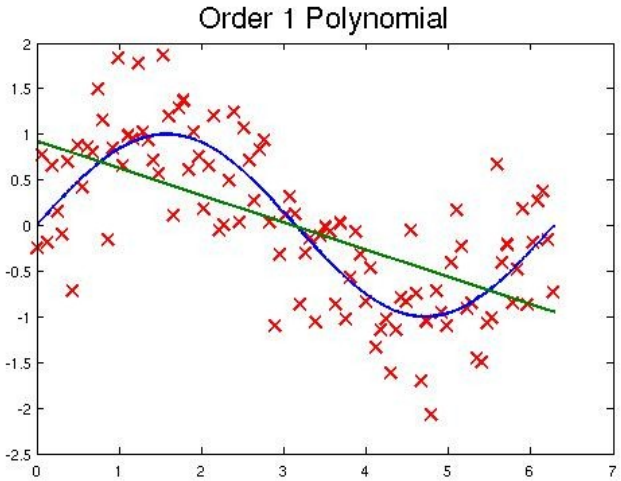
Non-Linear Regression

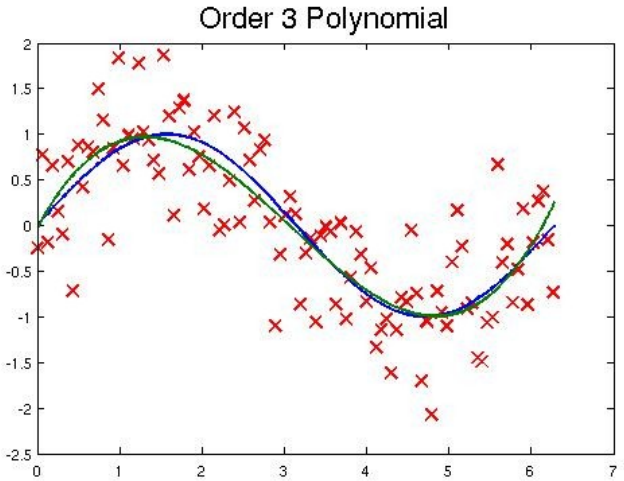
$$\hat{y} = \Phi(\mathbf{x}) \cdot \omega$$

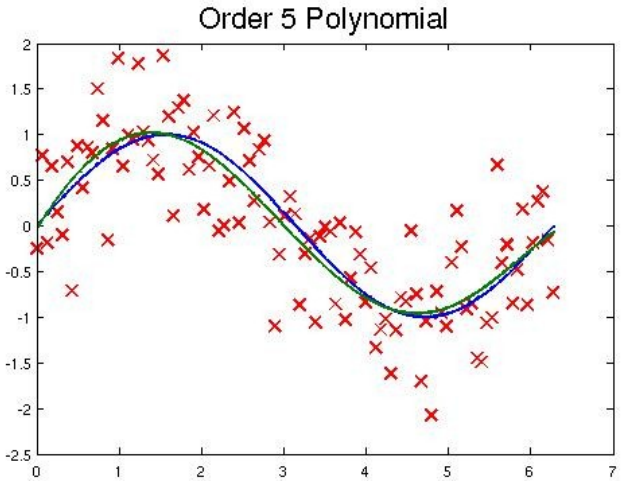
- We can use the same trick that we used for classification
- Make our estimate be a linear combination of a non-linear transformation of ***x***
- Possible example:

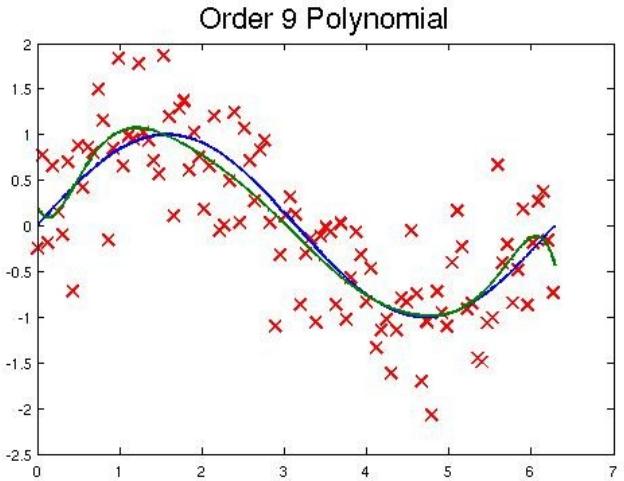
$$\Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ & \vdots & \\ x_N^2 & x_N & 1 \end{bmatrix}$$

Example



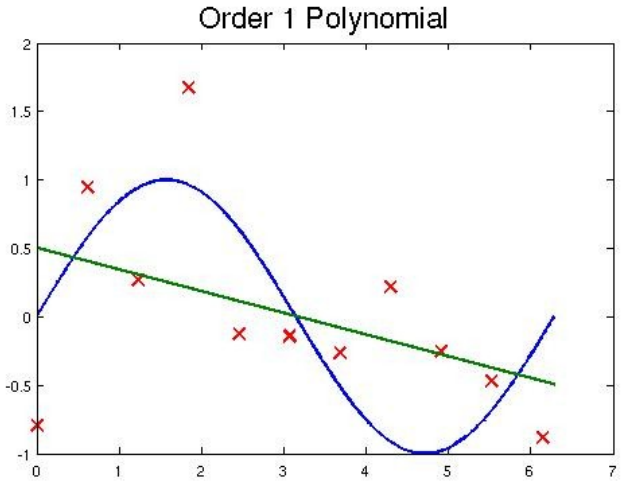


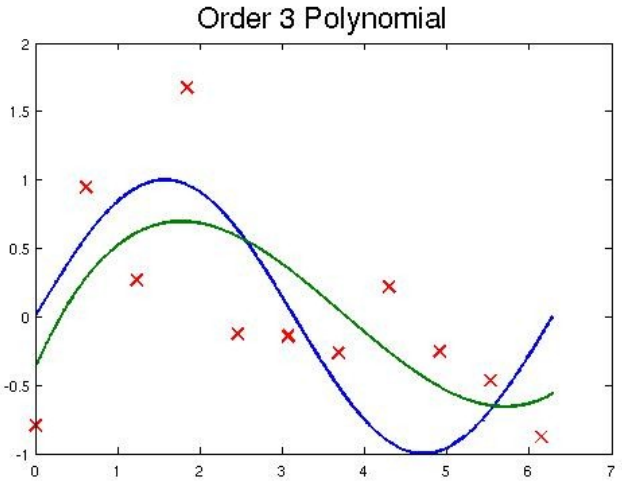


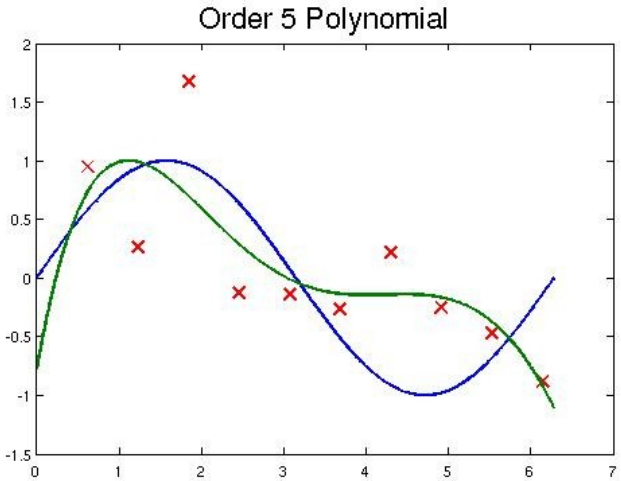


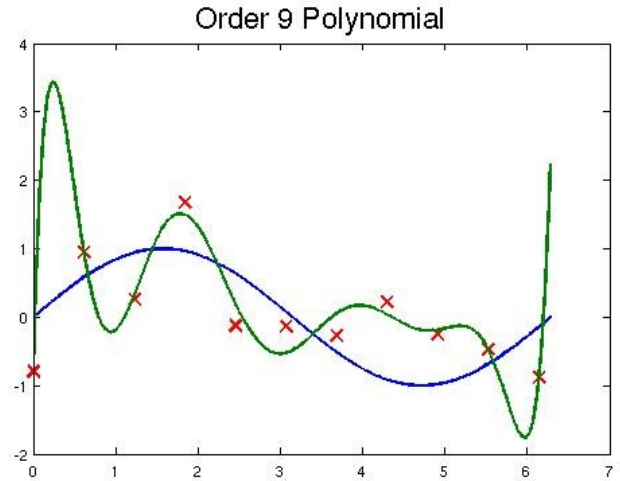
(Similar to Example from Bishop's Book)

What's going on here?





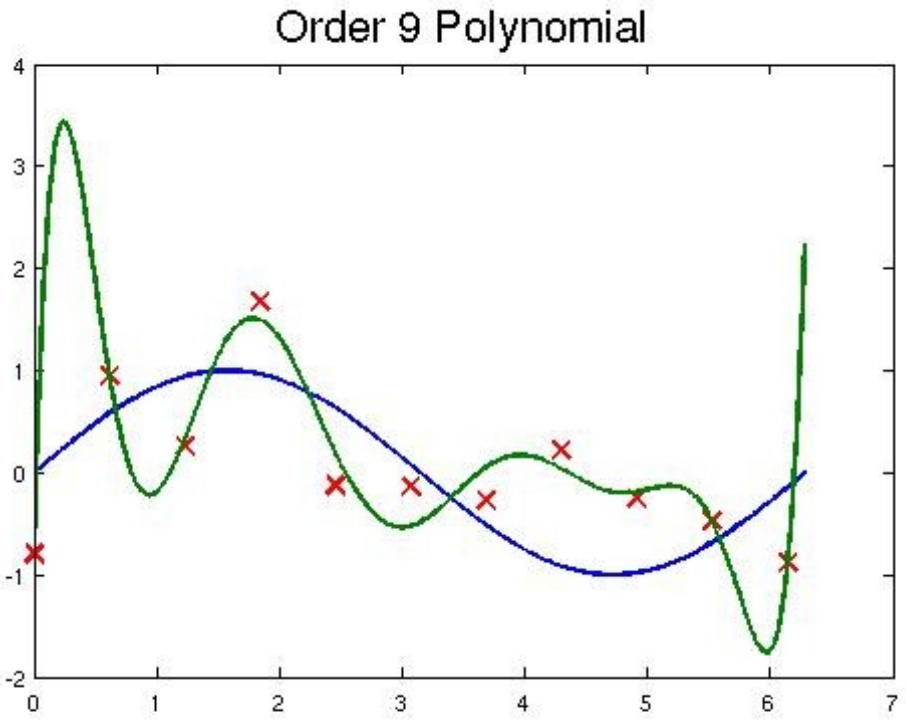




(Similar to Example from Bishop's Book)

What's going on here?

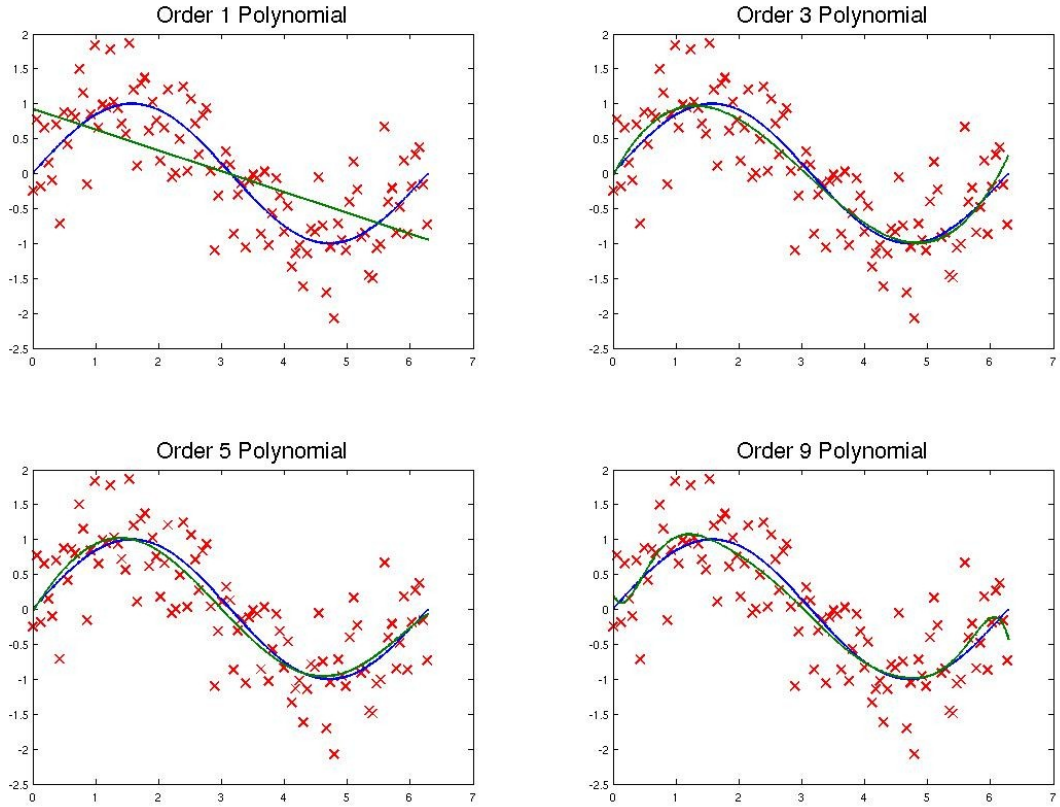
- This is classic over-fitting
- Not enough data, too much flexibility in classifier



(Similar to Example from Bishop's Book)

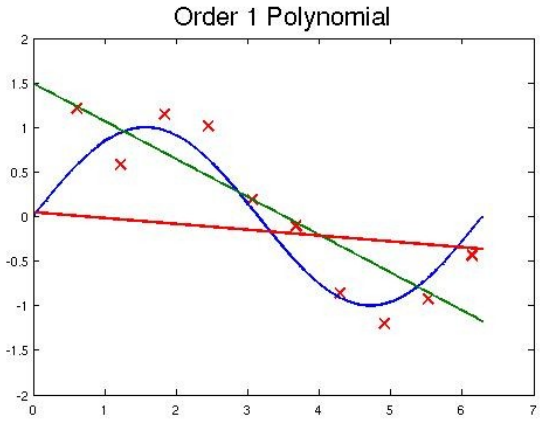
Two Solutions

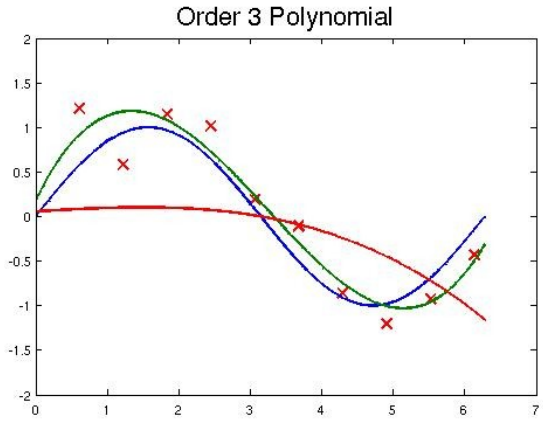
- Solution 1: Model Selection
- We want the simplest model that matches the data well

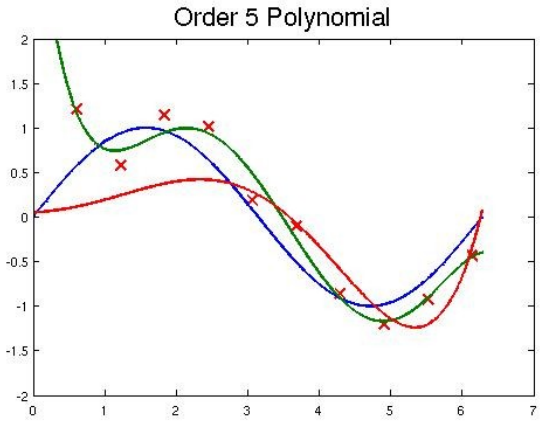


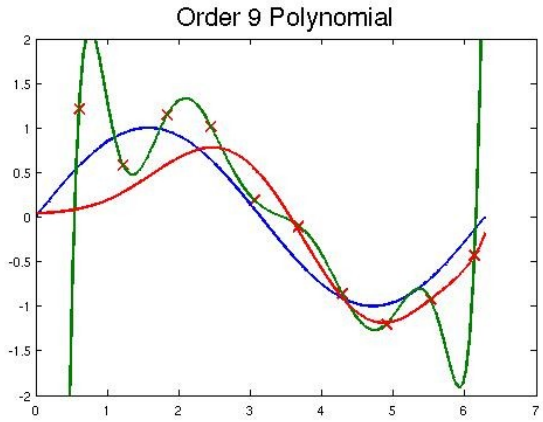
Solution #2: Regularization

- Smooths out bumps in the curve









Notice that we have oversmoothed the Lower-order polynomials!

Mathematically

- We will change the training criterion from

$$L(m,b) = \sum_{i=1}^{N} (mx_i + b - y_i)^2$$

- To

$$L(m,b) = \left[\sum_{i=1}^{N} (mx_i + b - y_i)^2 \right] + \lambda(m^2 + b^2)$$

Least squares (global) optimization

Good

- Clearly specified objective
- Optimization is easy

Bad

- May not be what you want to optimize
- Sensitive to outliers
  - Bad matches, extra points
- Doesn’t allow you to get multiple good fits
  - Detecting multiple objects, lines, etc.

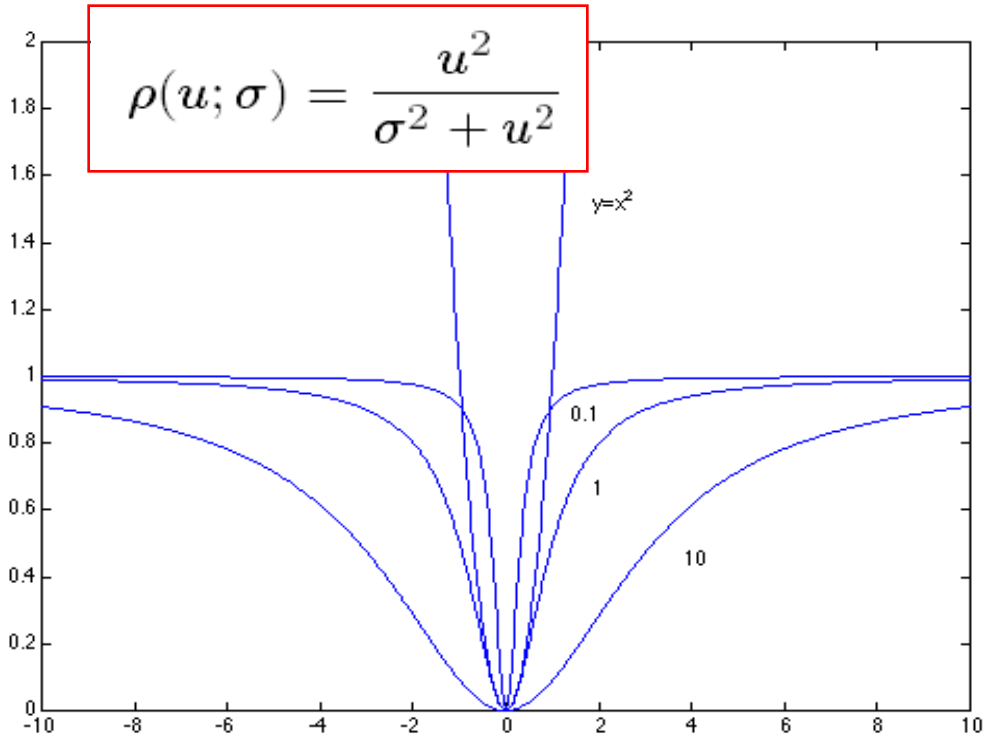
Robust least squares (to deal with outliers)

General approach:

minimize $\sum_i \rho(u_i(x_i, \theta); \sigma)$ $\quad u^2 = \sum_{i=1}^{n} (y_i - mx_i - b)^2$

$u_i(x_i, \theta)$ – residual of i[th] point w.r.t. model parameters ϑ

ρ – robust function with scale parameter σ



The robust function ρ

- Favors a configuration with small residuals
- Constant penalty for large residuals

Slide from S. Savarese

Robust Estimator

1. Initialize: e.g., choose θ by least squares fit and

   $$\sigma = 1.5 \cdot \text{median}(error)$$

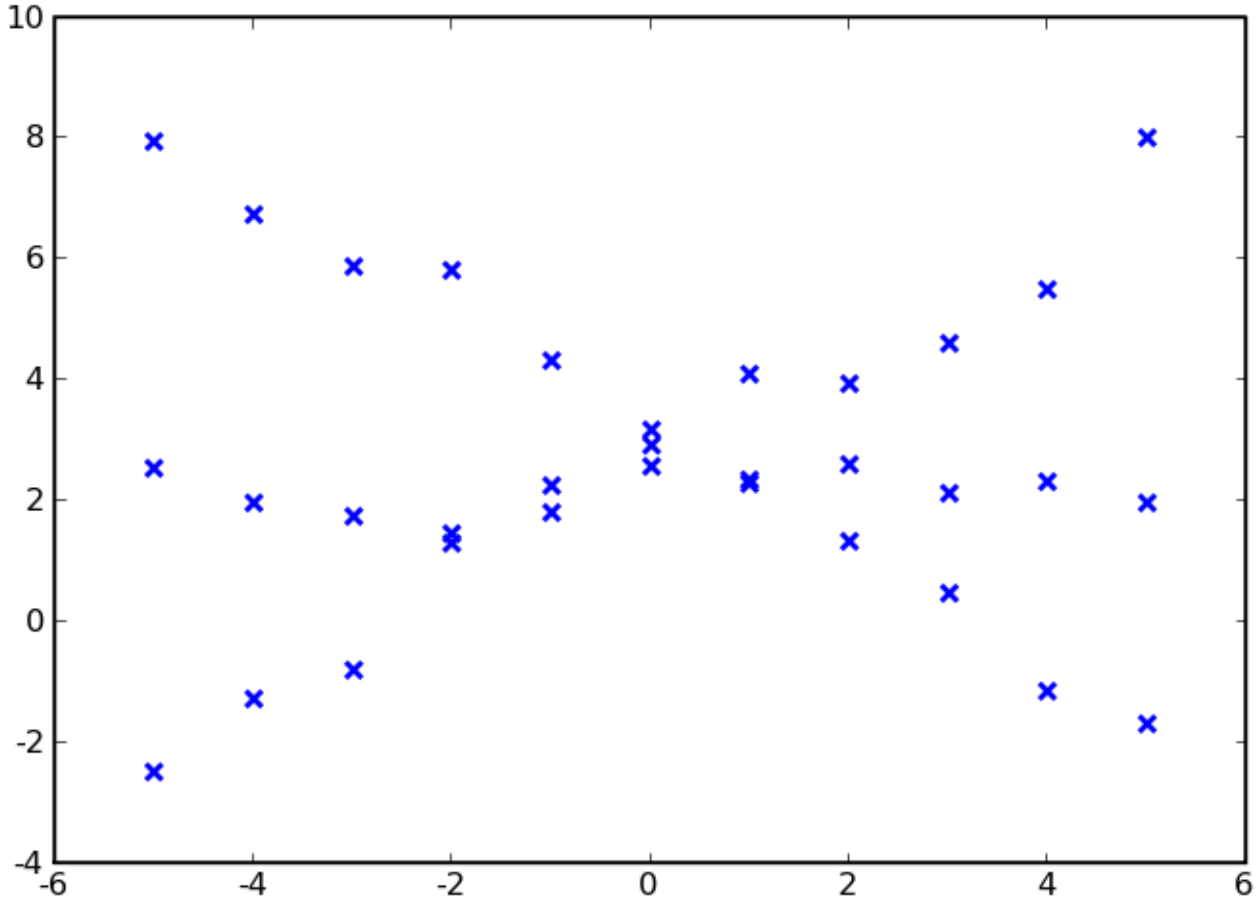
2. Choose params to minimize:
   - E.g., numerical optimization

   $$\sum_i \frac{error(\theta, data_i)^2}{\sigma^2 + error(\theta, data_i)^2}$$

3. Compute new $\sigma = 1.5 \cdot \text{median}(error)$

4. Repeat (2) and (3) until convergence

What if we have multiple lines?



We'll Need to Fit Multiple Lines

- Two Problems:
- Problem 1: How many lines?
  - We'll assume we know this
- Problem 2: Which points go to which lines?
  - We'll figure this out as part of the optimization process

Mathematical Formulation

- Again, we will define a cost function

$$L = \sum_{i=1}^{N} \min_{j} (y_i - \mathbf{x}_i \cdot \theta_j)^2$$

- Note that we are still minimizing a squared error, but we have included a min term
- This says, “I only count the minimum error”
- Practical effect- each line only has to cover part of the data set

Optimizing This

Note, this will be very difficult to optimize

$$L = \sum_{i=1}^{N} \min_{j} (y_i - \mathbf{x}_i \cdot \theta_j)^2$$

So we will introduce a second set of variables

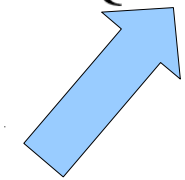
$$L = \sum_{i=1}^{N} \sum_{j=1}^{N_C} \mathbf{z}_{i,j} (y_i - \mathbf{x}_i \cdot \theta_j)^2$$
$$\mathbf{z}_{i,j} \in \{0,1\} \forall i,j, \quad \sum_{j=1}^{N_c} \mathbf{z}_{i,j} = 1$$

What this means

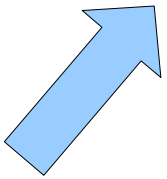
z assigns every point to one line

$$L = \sum_{i=1}^{N} \sum_{j=1}^{N_C} \mathbf{z}_{i,j} (y_i - \mathbf{x}_i \cdot \theta_j)^2$$

$$\mathbf{z}_{i,j} \in \{0, 1\} \forall i, j, \quad \sum_{j=1}^{N_c} \mathbf{z}_{i,j} = 1$$



Every entry is either 0 or 1



Only one entry is equal to one

Optimizing this

- This leads to a natural two-step process
  - 1. Optimize θ for each line –
    - this is the same least-squares formulation that we used before, except we only consider the points assigned to that line
  - *2.* Optimize **z**
    - Set **z** so that the line is assigned to the line with the minimum error

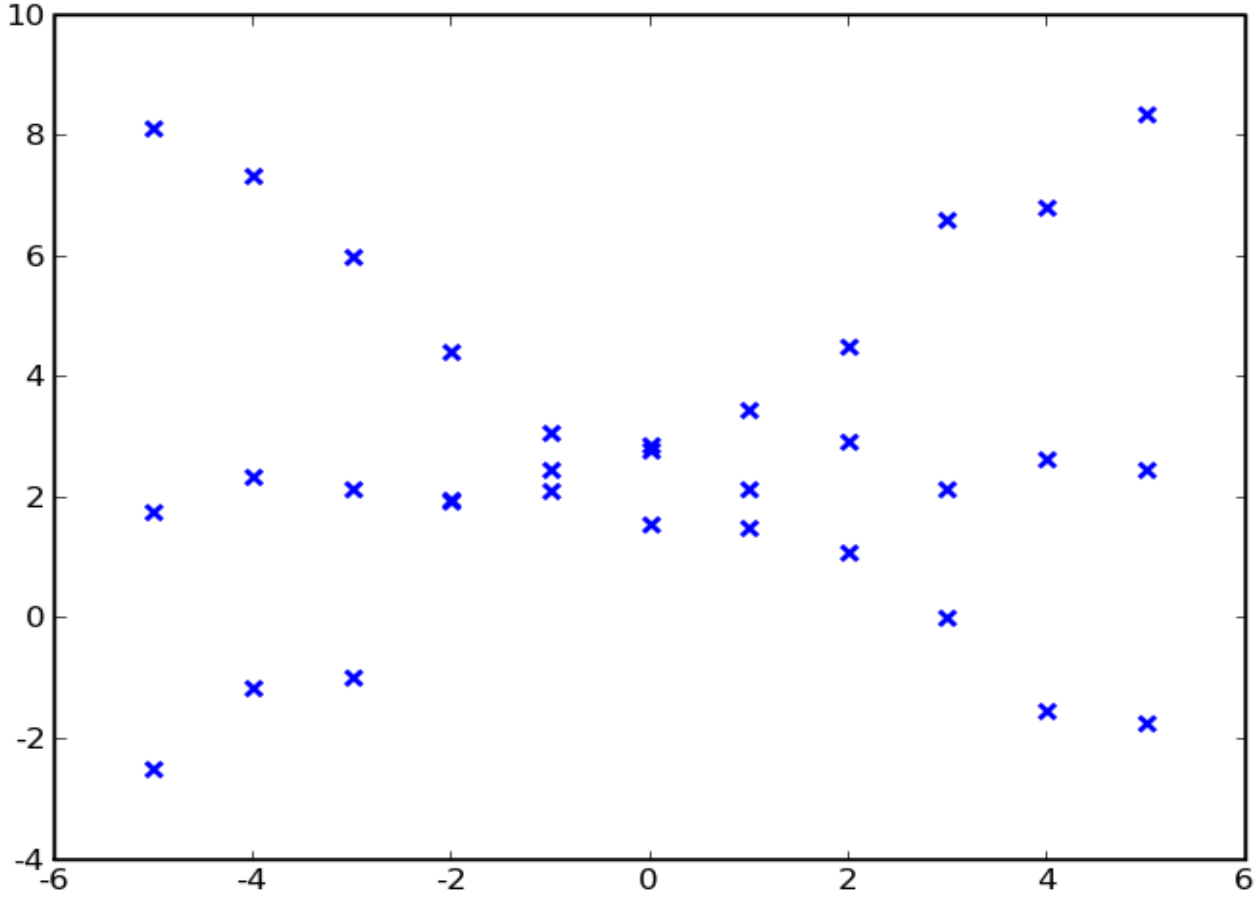
$$L = \sum_{i=1}^{N} \sum_{j=1}^{N_C} \mathbf{z}_{i,j} (y_i - \mathbf{x}_i \cdot \theta_j)^2$$

$$\mathbf{z}_{i,j} \in \{0,1\} \forall i,j, \quad \sum_{j=1}^{N_c} \mathbf{z}_{i,j} = 1$$

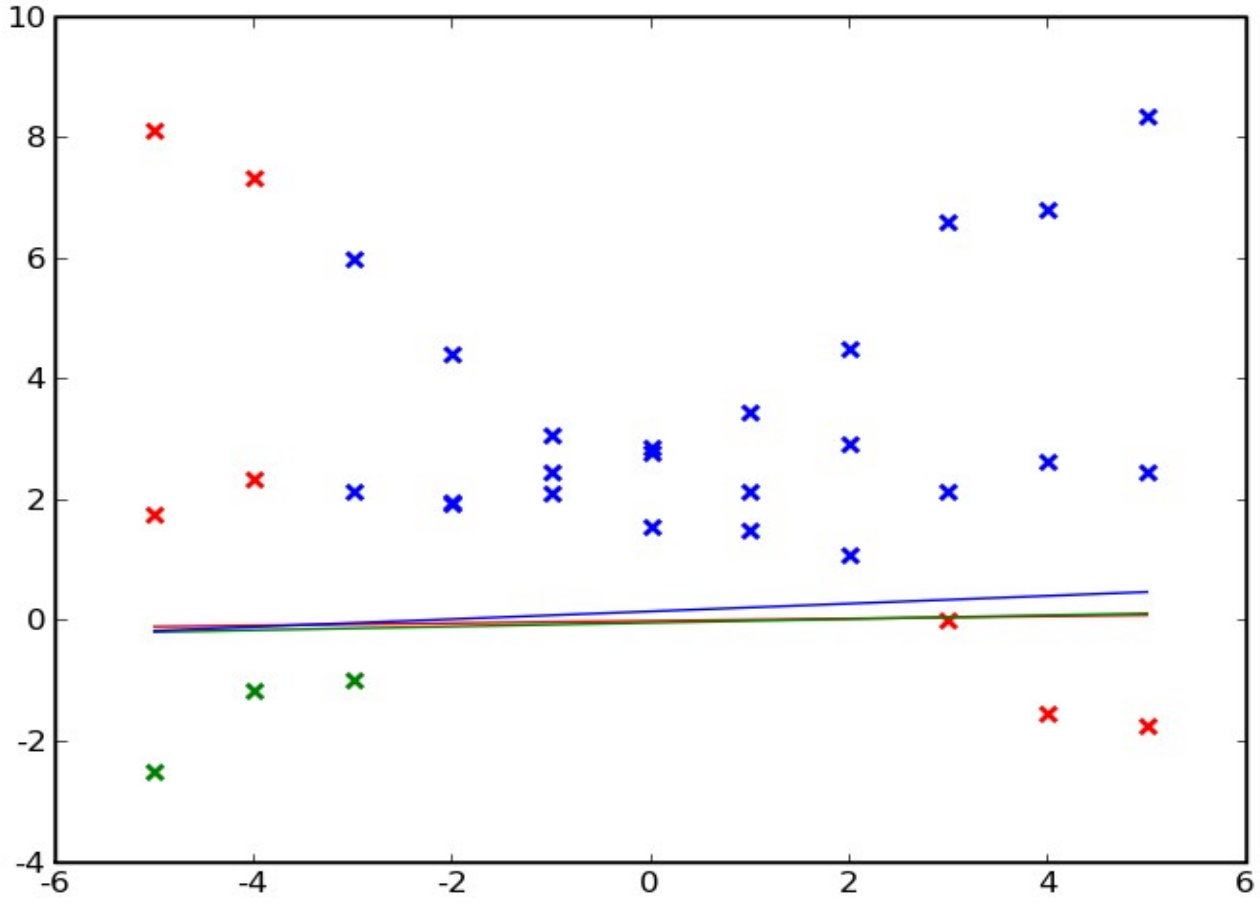
An Example

We start with the points unassigned

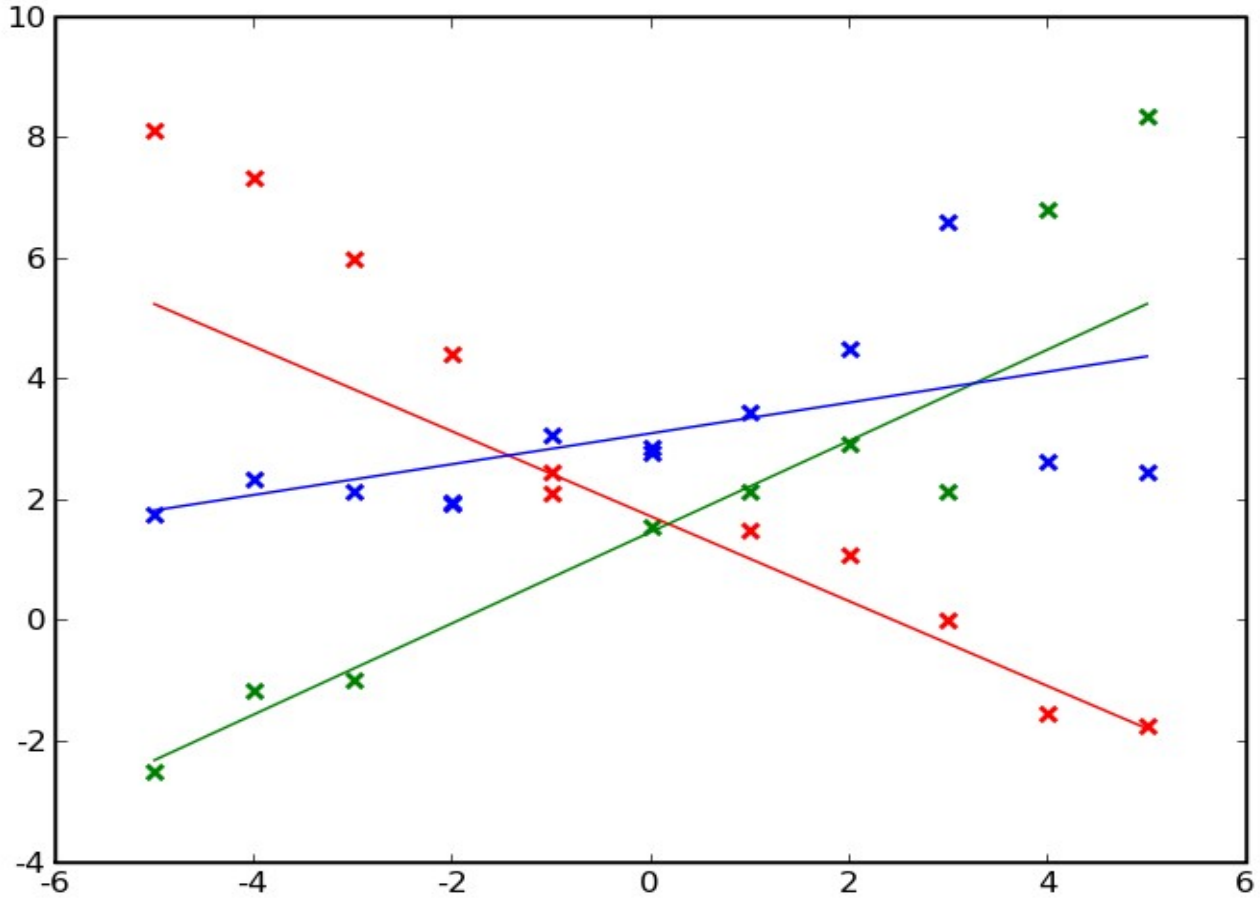
Choose lines randomly



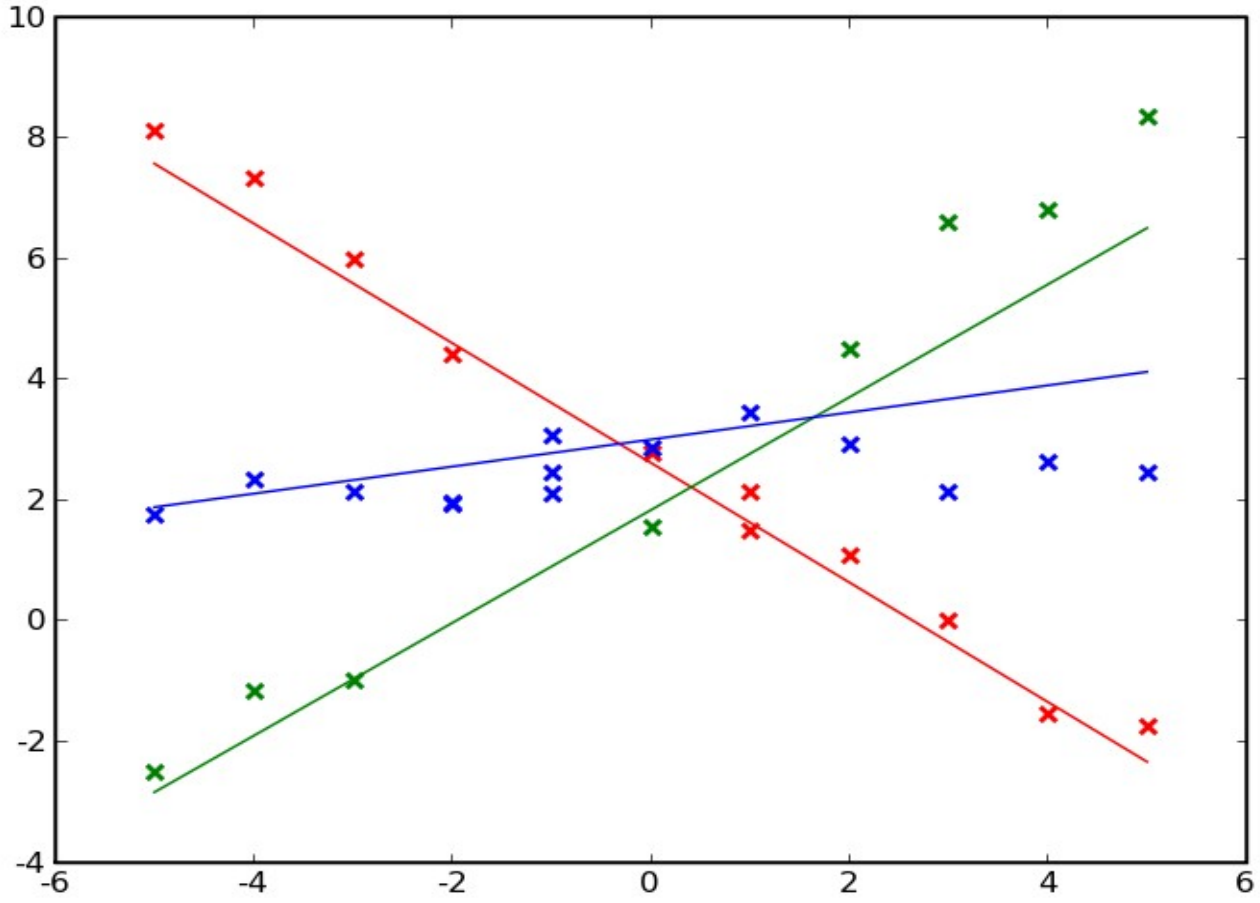
An Example



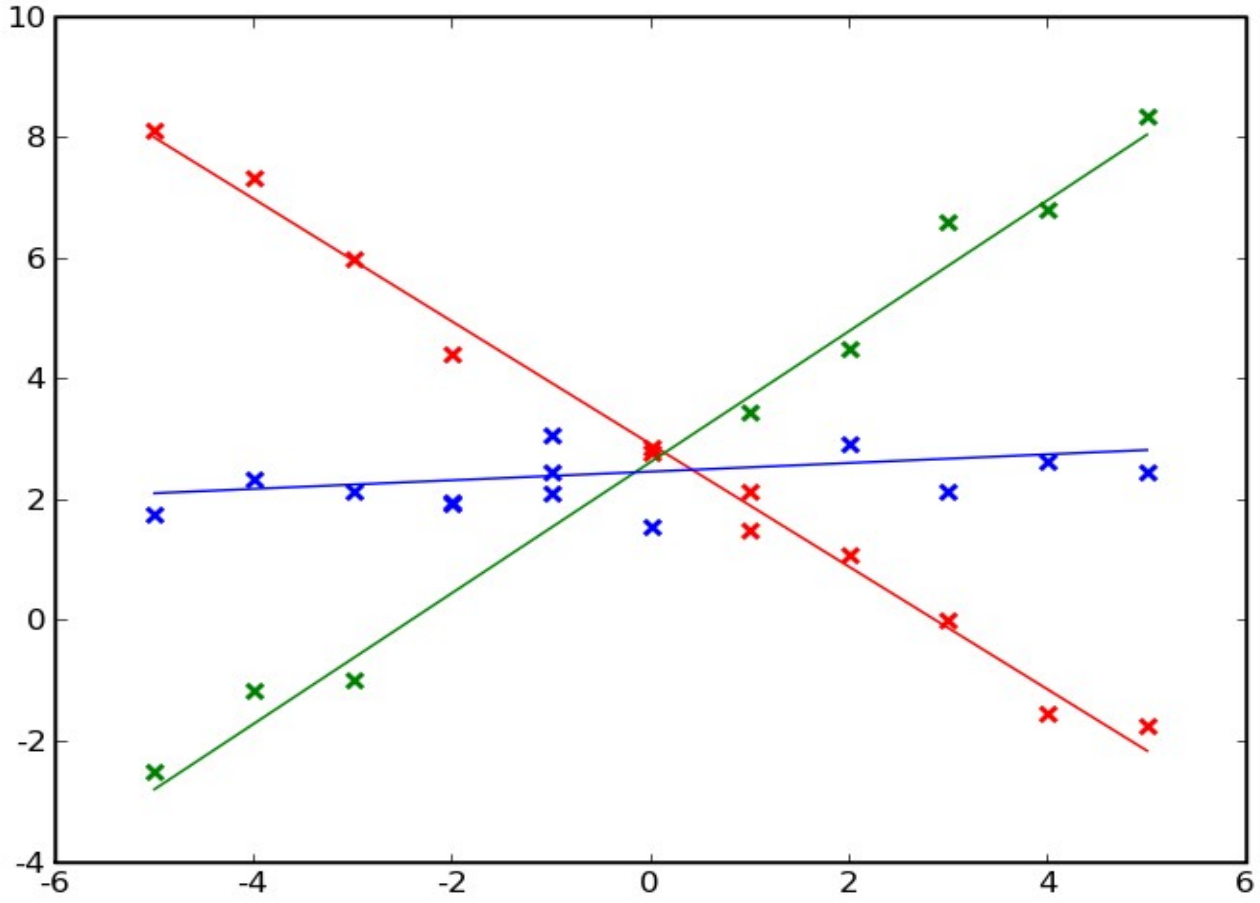
An Example



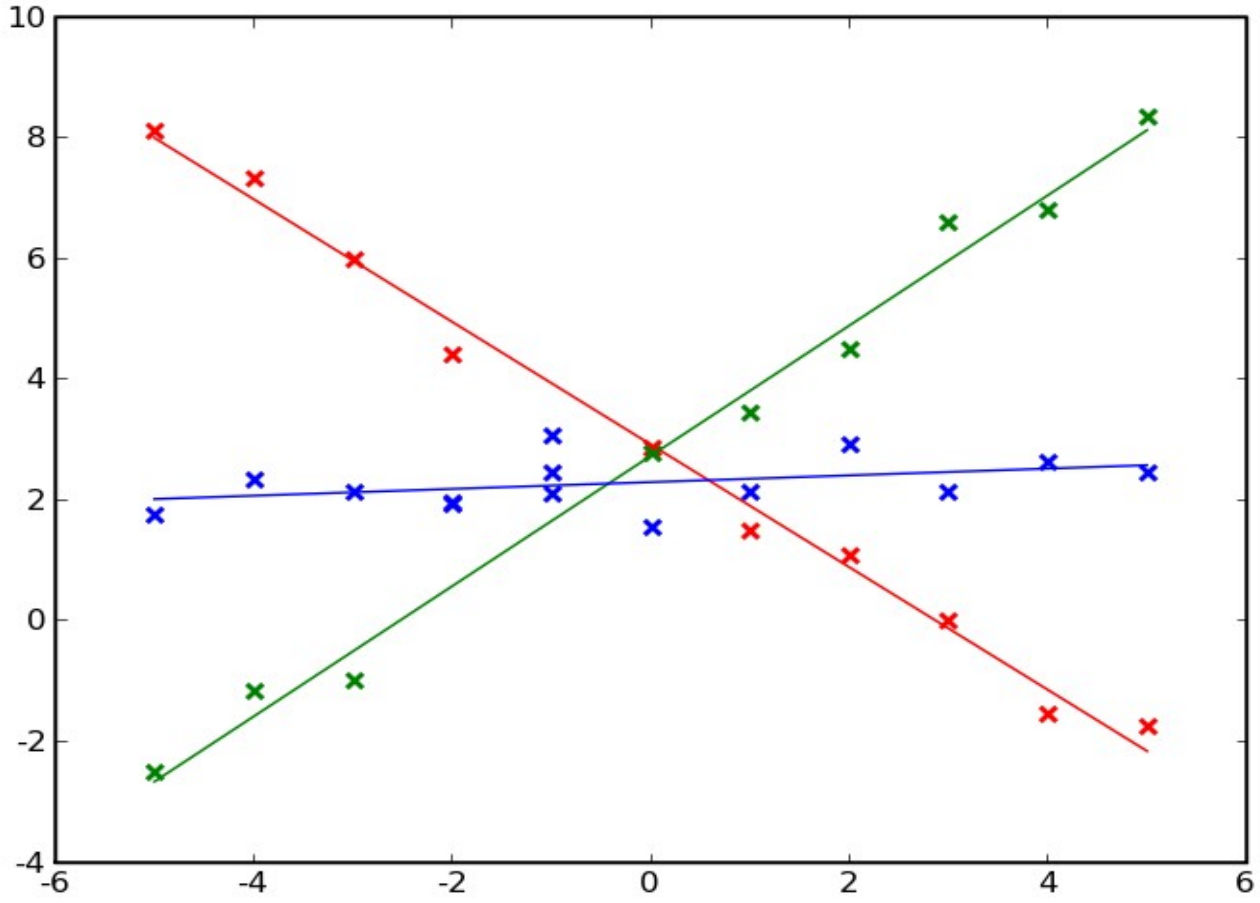
An Example



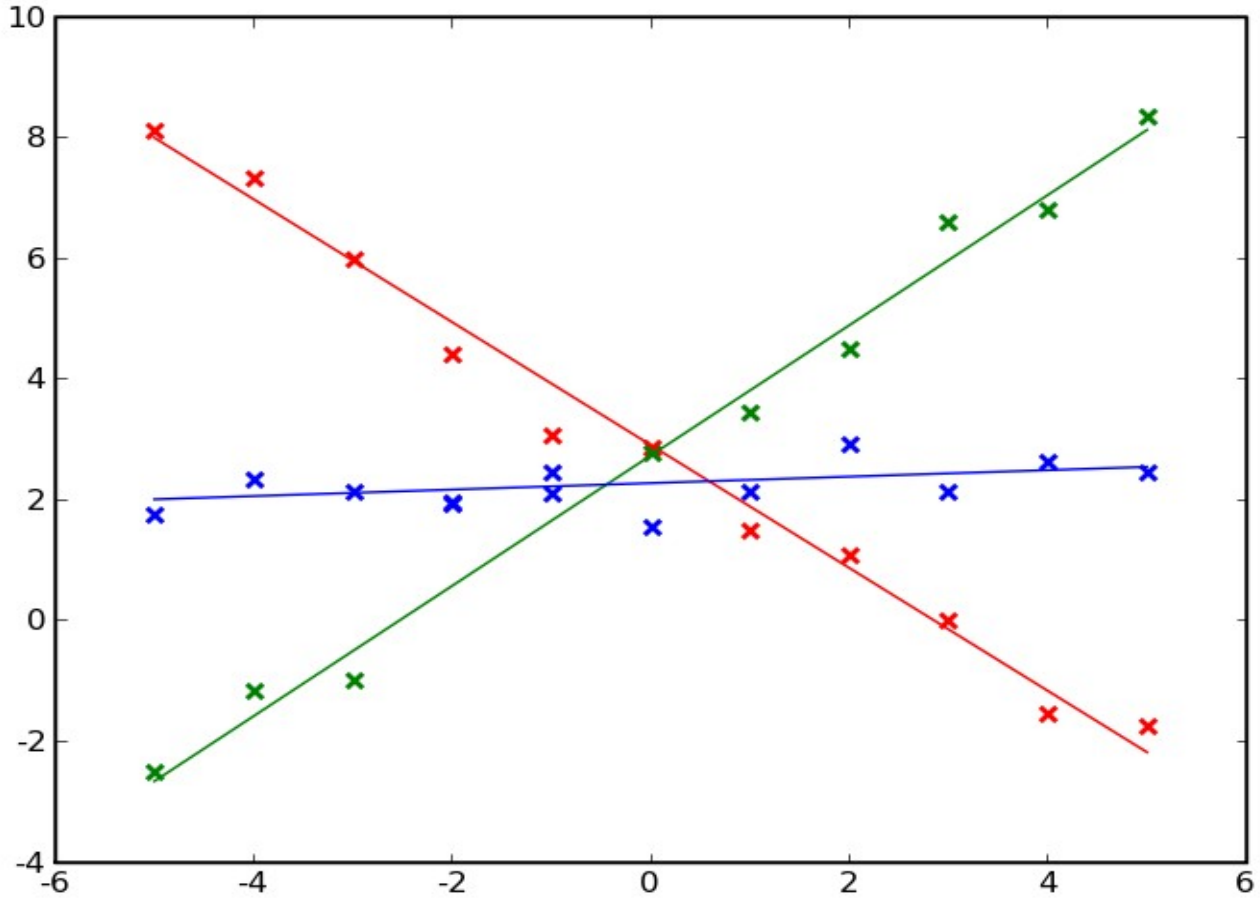
An Example



An Example



An Example



Fitting Multiple Lines

Note that we did have to know how many lines we had

This is not guaranteed to converge to a global minimum

Similar to the *k*-means algorithm that we will study soon.

Another Approach

Basic Idea: Let edge points vote for possible lines

Hough Transform

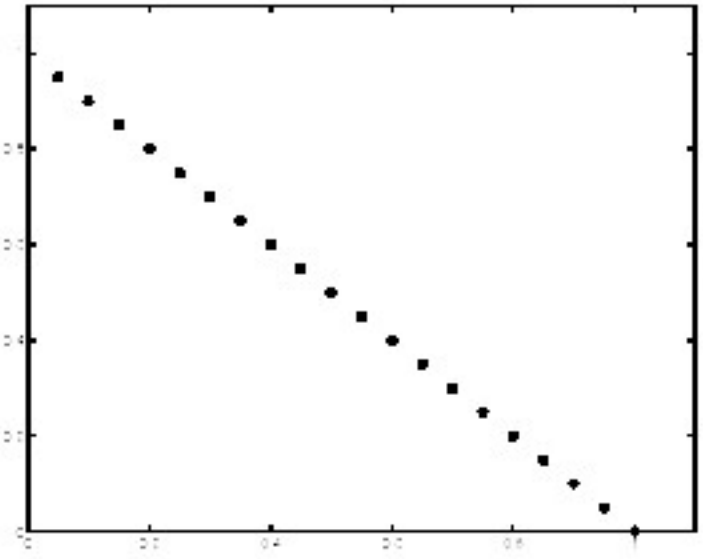
- Parameterize lines as

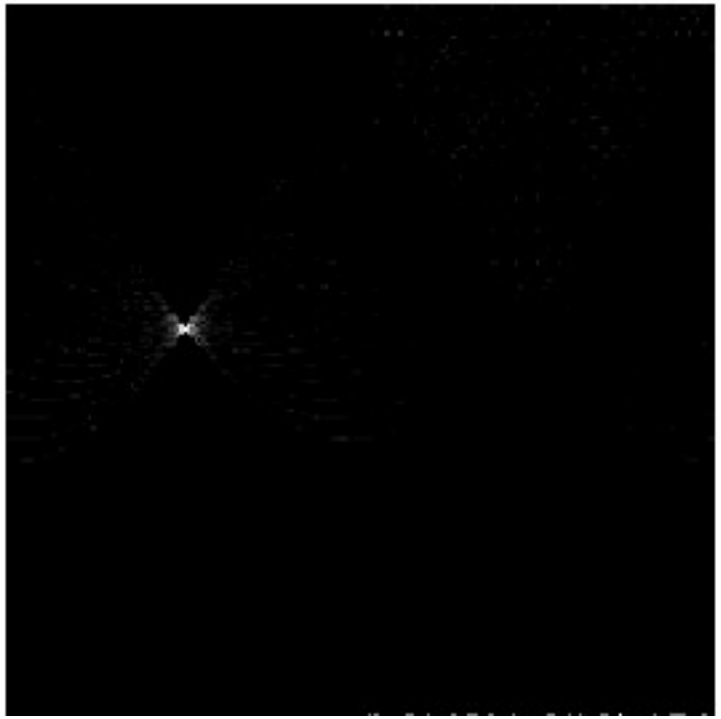
$$x \cos\theta + y \sin\theta + r = 0$$

- Every line can be expressed as a pair of values, θ and r
- To find lines, we'll let each point vote for the possible lines that it could lie on.

Basic Idea

- Discretize the space of θ and r
- Each point “votes” for all (θ, r) combinations that result in a line passing through that point
- Combinations corresponding to lines should receive many votes



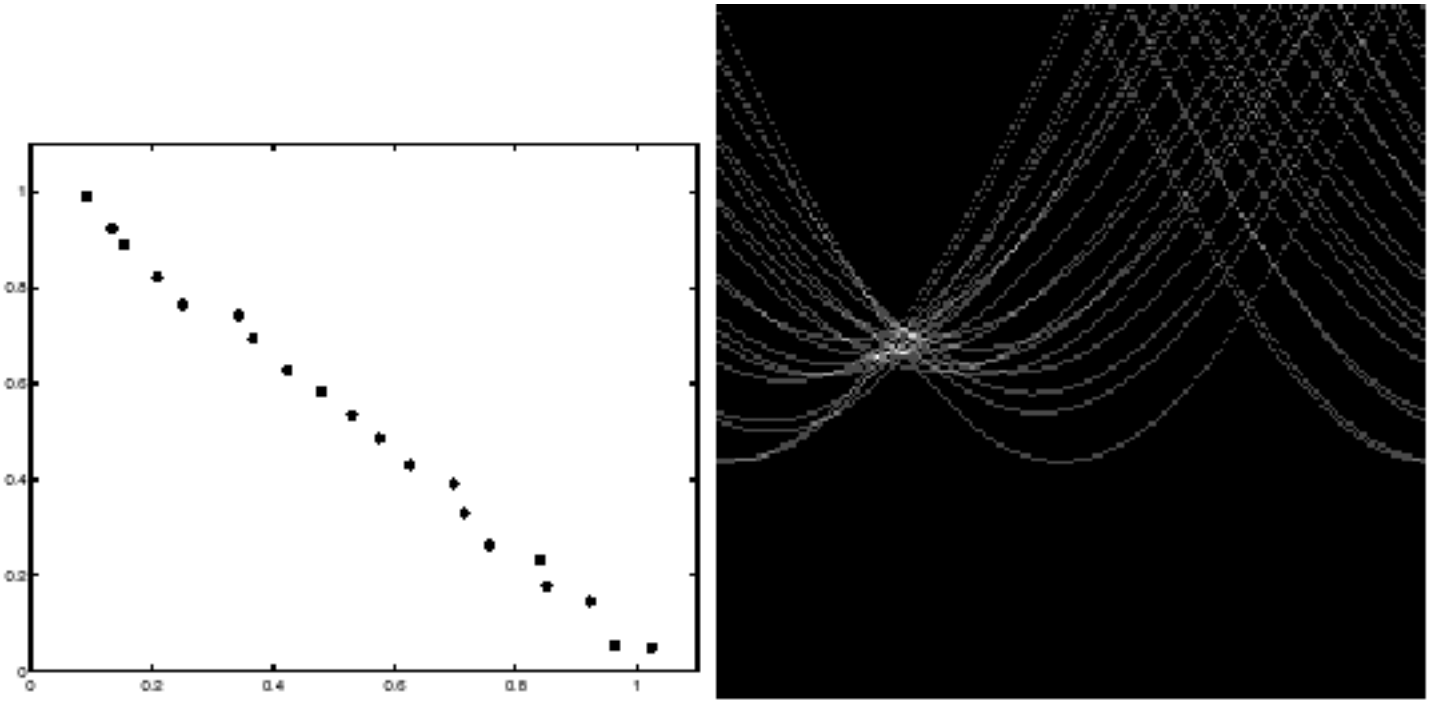


θ

r

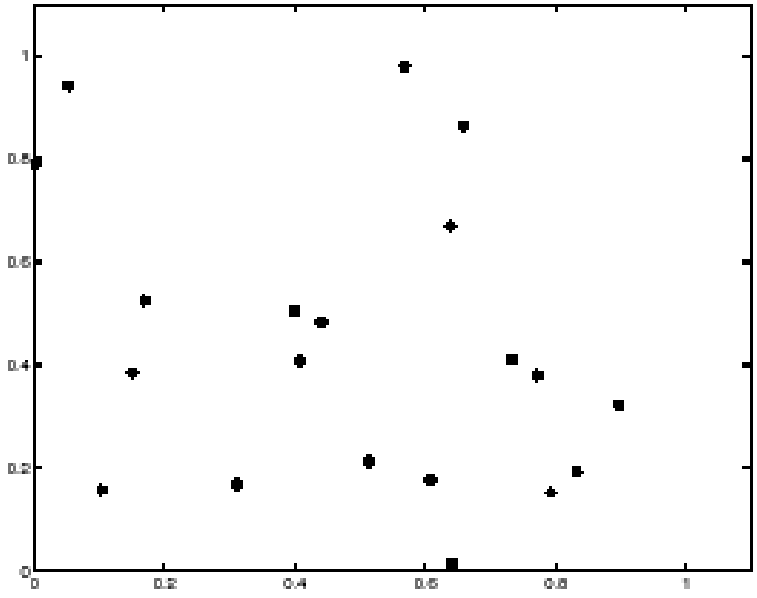
(From Slides by Forsyth)

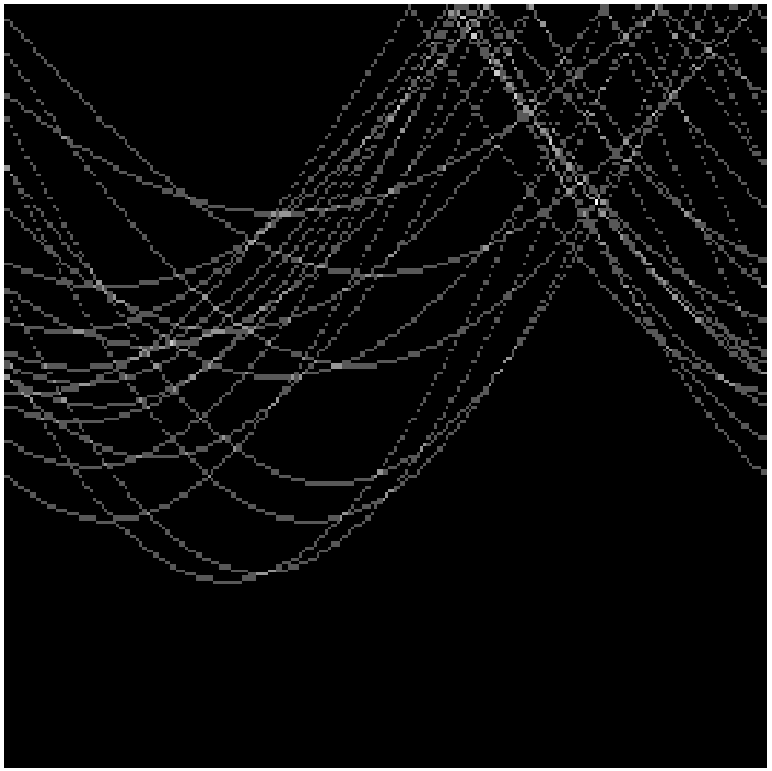
What if there's noise?



(From Slides by Forsyth)

What if there are no lines?





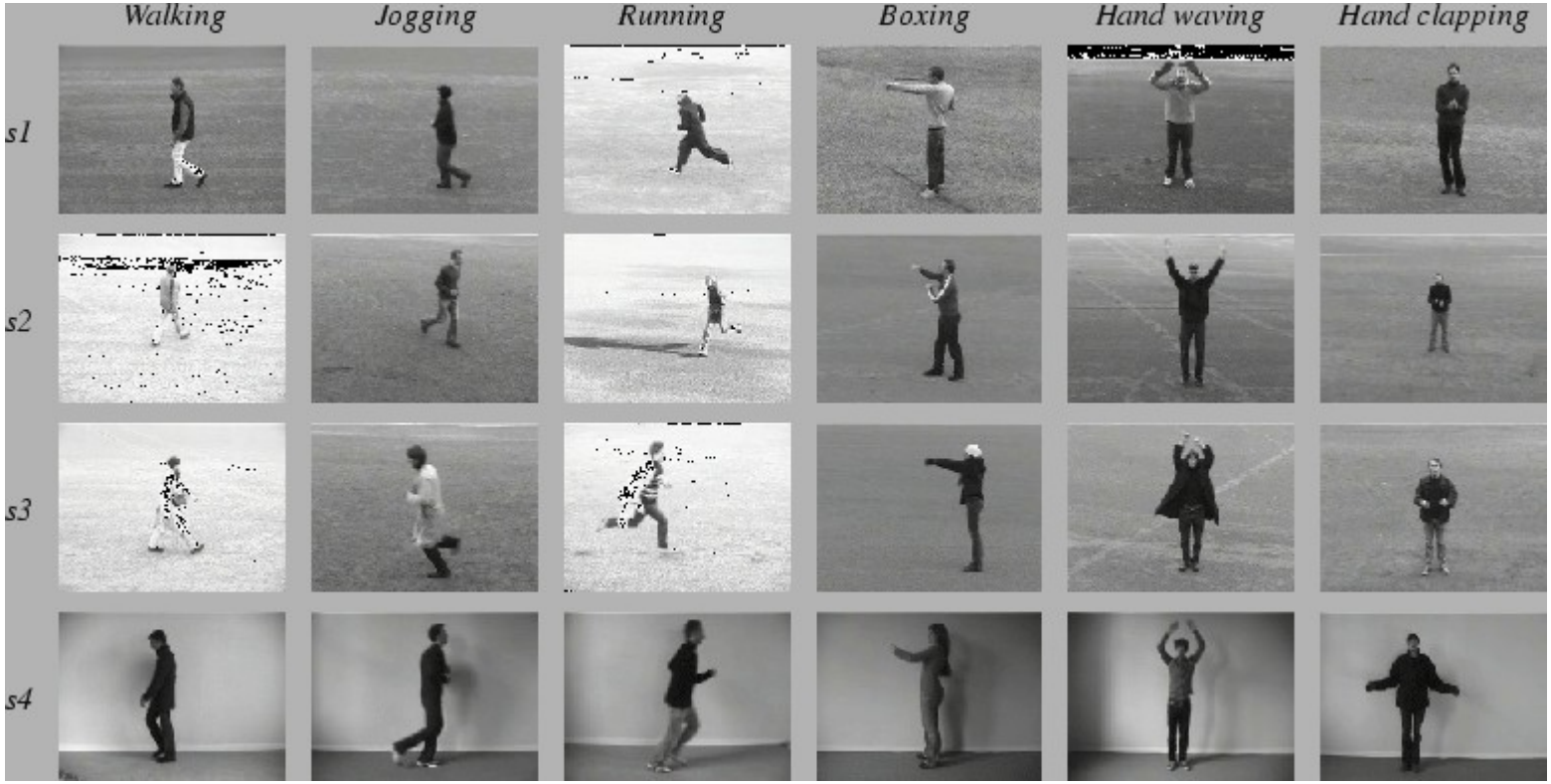
(From Slides by Forsyth)

The Hough Transform

- Advantages:
  - Don’t need to know the number of lines ahead of time
  - Conceptually Simple
  - Can be extended to other types of shapes
- Disadvantages:
  - Noise leads to lots of phantom lines
  - Have to pick the right quantization

Voting Paradigm

- This voting paradigm can be extended to very different problems



Voting

- Can use to find actions

Incremental Action Recognition Using Feature-Tree

Kishore K Reddy
Computer Vision Lab
University of Central Florida
kreddy@cs.ucf.edu

Jingen Liu
Computer Vision Lab
University of Central Florida
liujg@cs.ucf.edu

Mubarak Shah
Computer Vision Lab
University of Central Florida
shah@cs.ucf.edu

Abstract

Action recognition methods suffer from many drawbacks in practice, which include (1)the inability to cope with incremental recognition problems; (2)the requirement of an intensive training stage to obtain good performance; (3) the inability to recognize simultaneous multiple actions; and (4) difficulty in performing recognition frame by frame. bag of words method, or construct new templates using new examples for a template based approach. Performing action recognition on a frame by frame basis is a very important requirement in real world videos. Furthermore, the most challenging task is dealing with videos containing multiple actions happening simultaneously and also having large occlusions. Most methods can only classify videos containing a single action and are not robust to large occlu-

Overall Pipeline

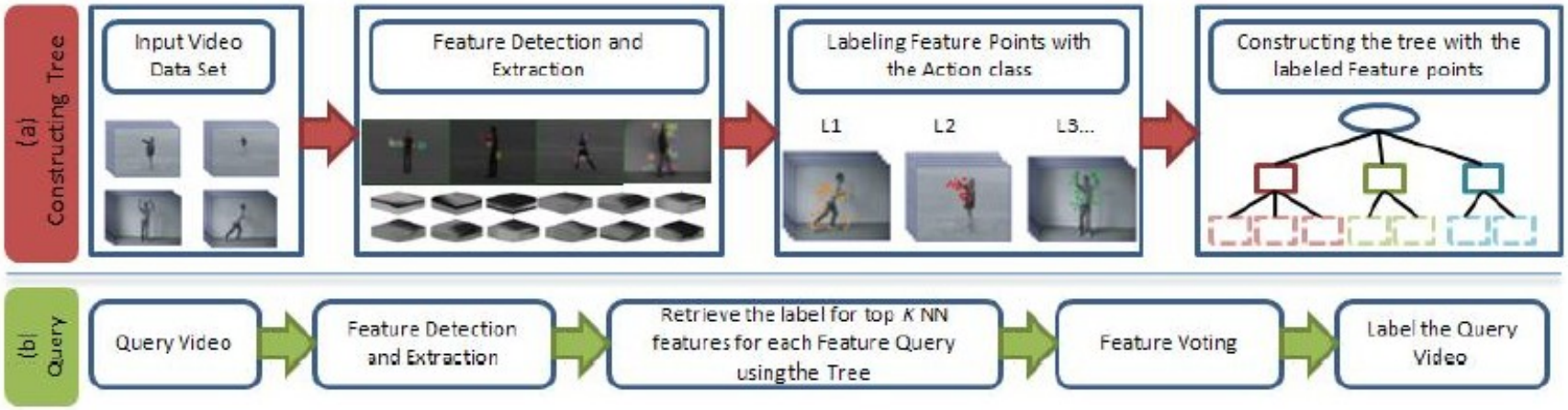
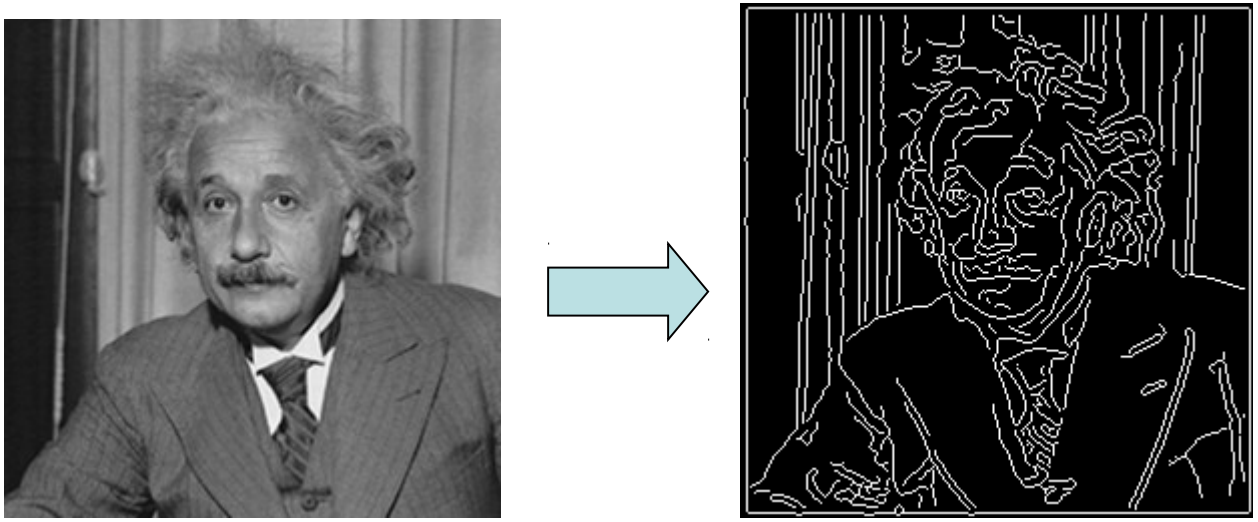


Fig. 1: The proposed framework for action recognition using feature-tree.

Changing Gears

The edge-detection algorithm gives us a binary map.



It would be useful to group these edges into coherent structures.

Connected Components Algorithm

Divides binary image into groups of connected pixels

First, need to define the neighborhood

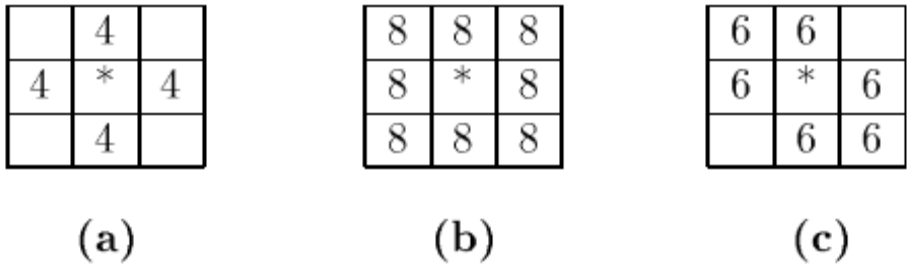


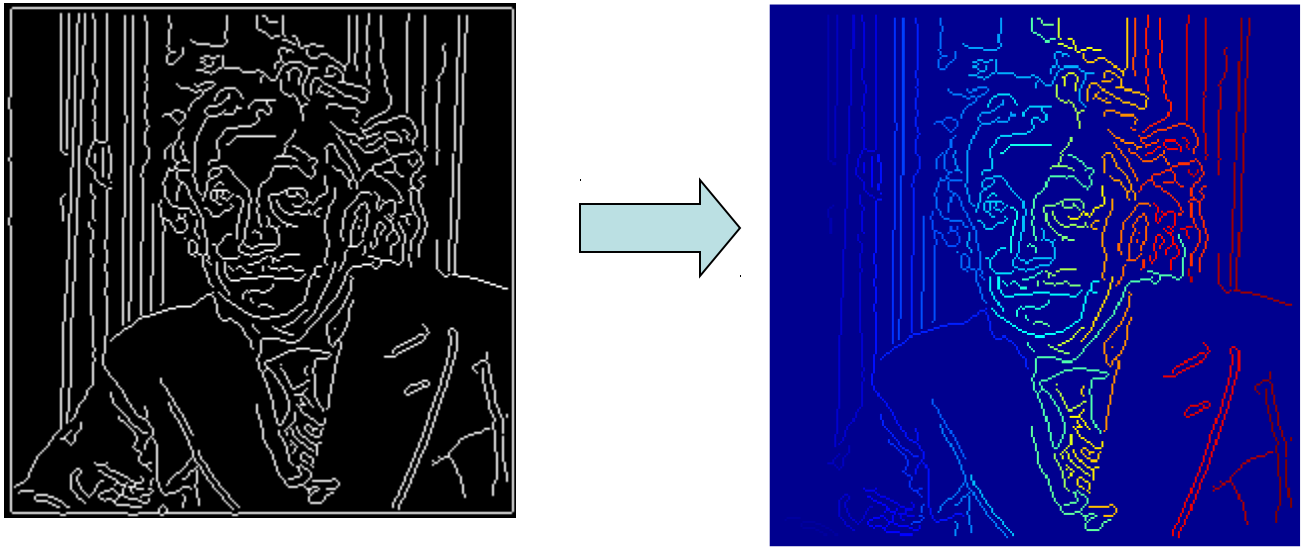
Figure 3.6: Pixel connectedness. (a) 4-connected. (b) 8-connected. (c) 6-connected.

Algorithm is easily explained recursively

1. Scan the binary image left to right, top to bottom.
2. If there is an unlabeled pixel with a value of '1' assign a new label to it.
3. Recursively check the neighbors of the pixel in step 2 and assign the same label if they are unlabeled with a value of '1'.
4. Stop when all the pixels of value '1' have been labeled.

Figure 3.7: Recursive Connected Component Algorithm.

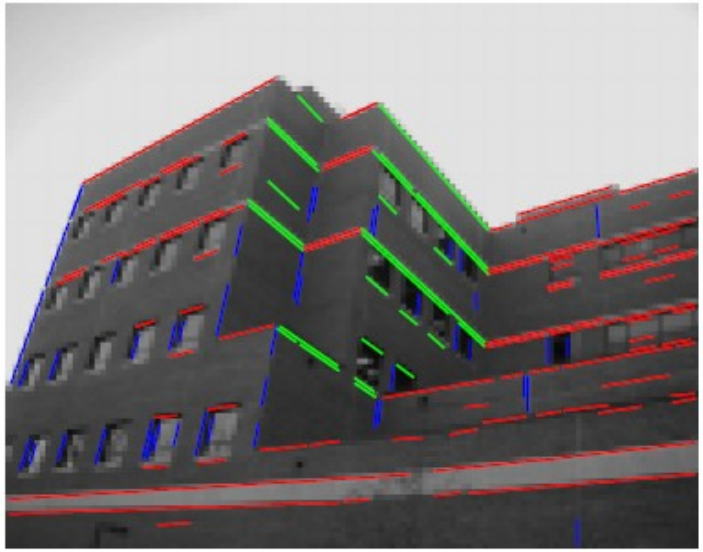
Connected Components

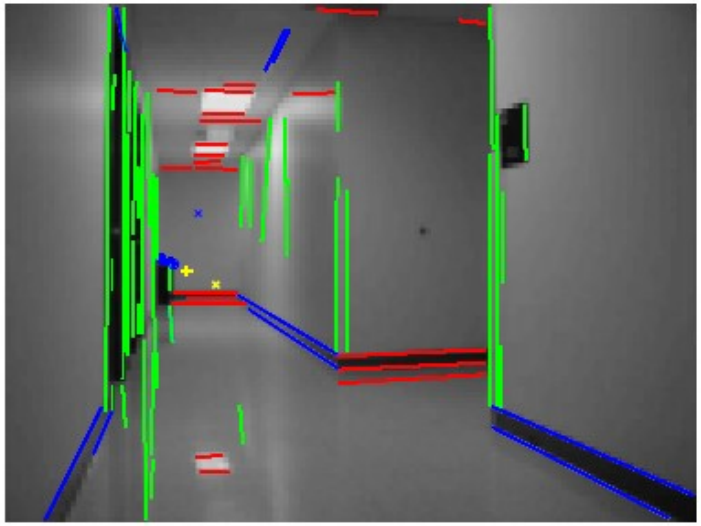


Called bwlabel in MATLAB image processing toolbox

Why do lines matter?

- Lines tell us about the structure of the world and the orientations of the camera
- In this model, we assume that the image holds lines that align with the world axes





(From Kosecka and Zhang - “Video Compass”)