

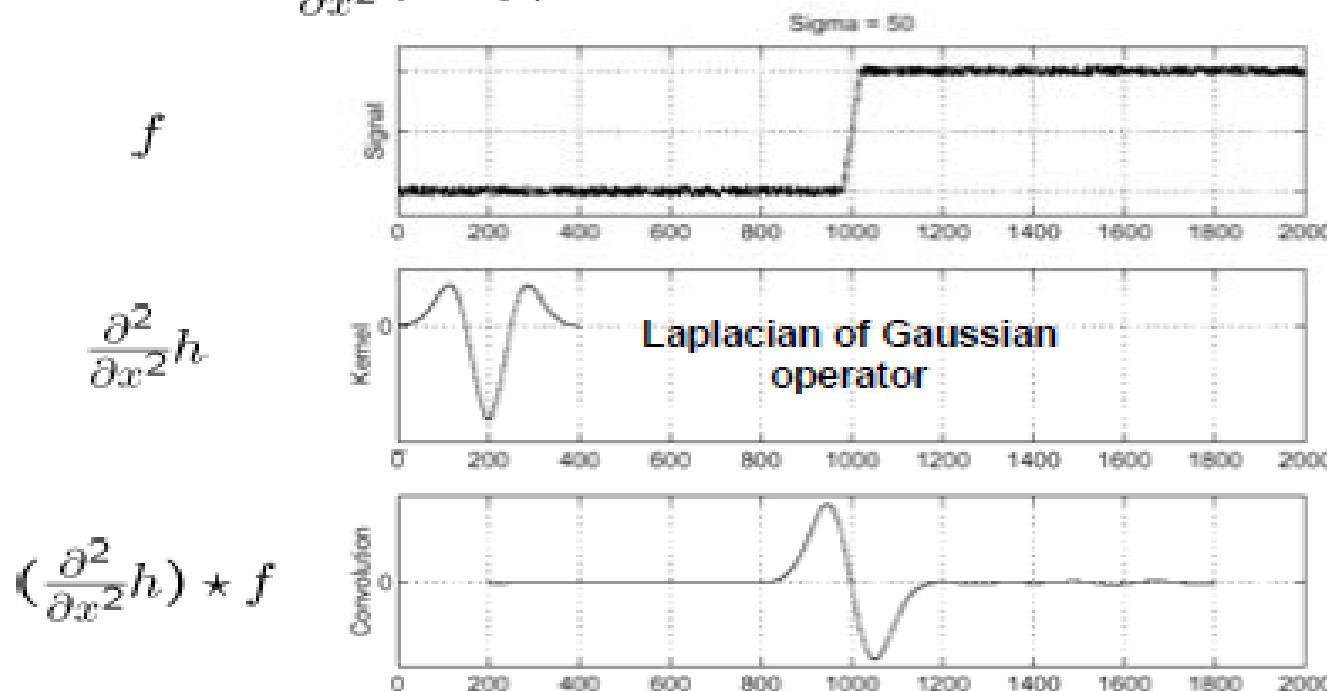
# Line fitting, Corner Extraction

# Today's Lecture

- Recap Canny edge detector
- Line Fitting
- Corner detection

# Edge as the zero-crossing of the second order derivative using Laplacian of Gaussian (LoG filter/Marr filter)

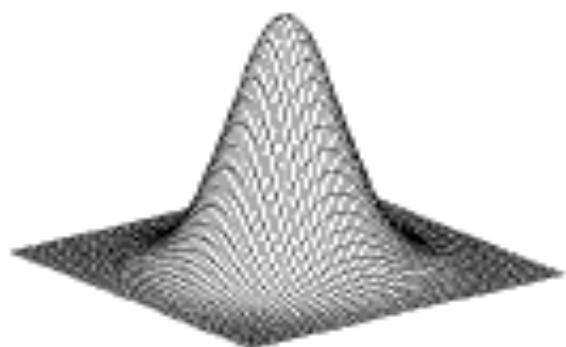
Consider  $\frac{\partial^2}{\partial x^2}(h * f)$



Where is the edge?

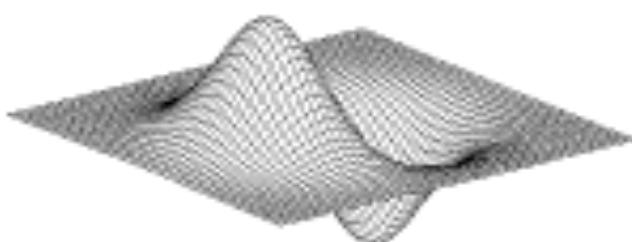
Zero-crossings of bottom graph

## 2D edge detection filters



Gaussian

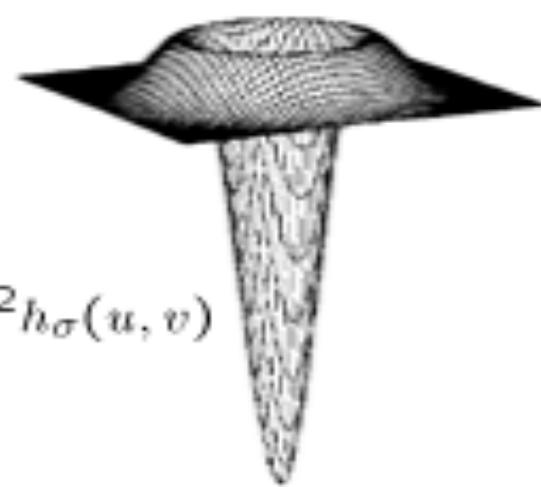
$$h_\sigma(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u, v)$$

Laplacian of Gaussian

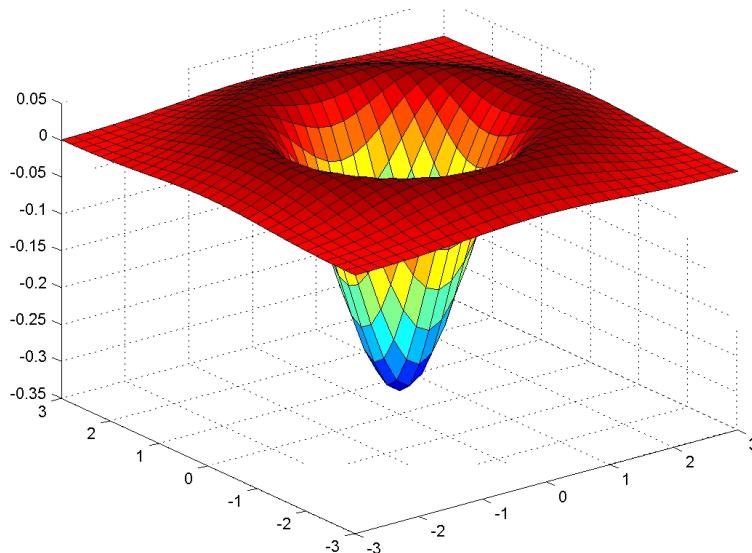


$$\nabla^2 h_\sigma(u, v)$$

- $\nabla^2$  is the Laplacian operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

# Laplacian of Gaussian (LoG): Marr edge detector



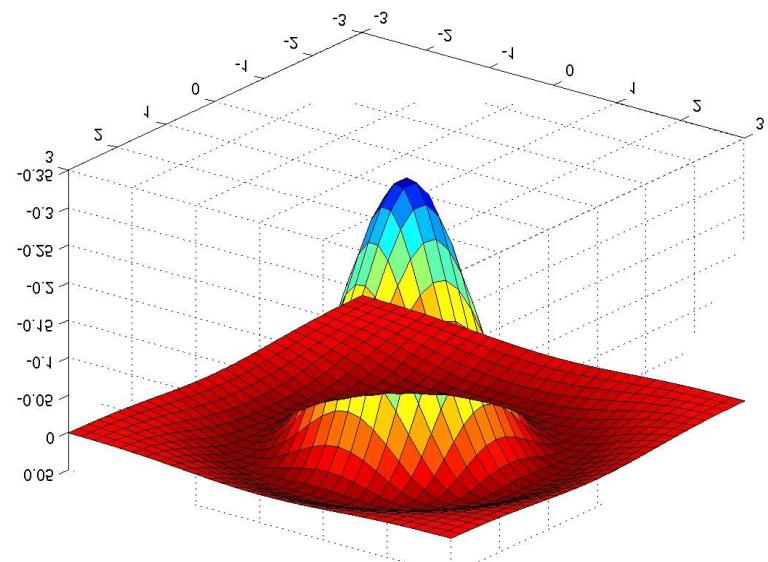
$$LoG_{\sigma} = -\frac{1}{\pi\sigma^4} \left[ 1 - \frac{x^2 + y^2}{2\sigma^2} \right] e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Matlab: `fspecial('log', sigma)`

# LoG kernel — Mexican hat

An example of an  $11 \times 11$  mask approximating the Laplacian of Gaussian with  $\sigma^2 = 2$  is given below:

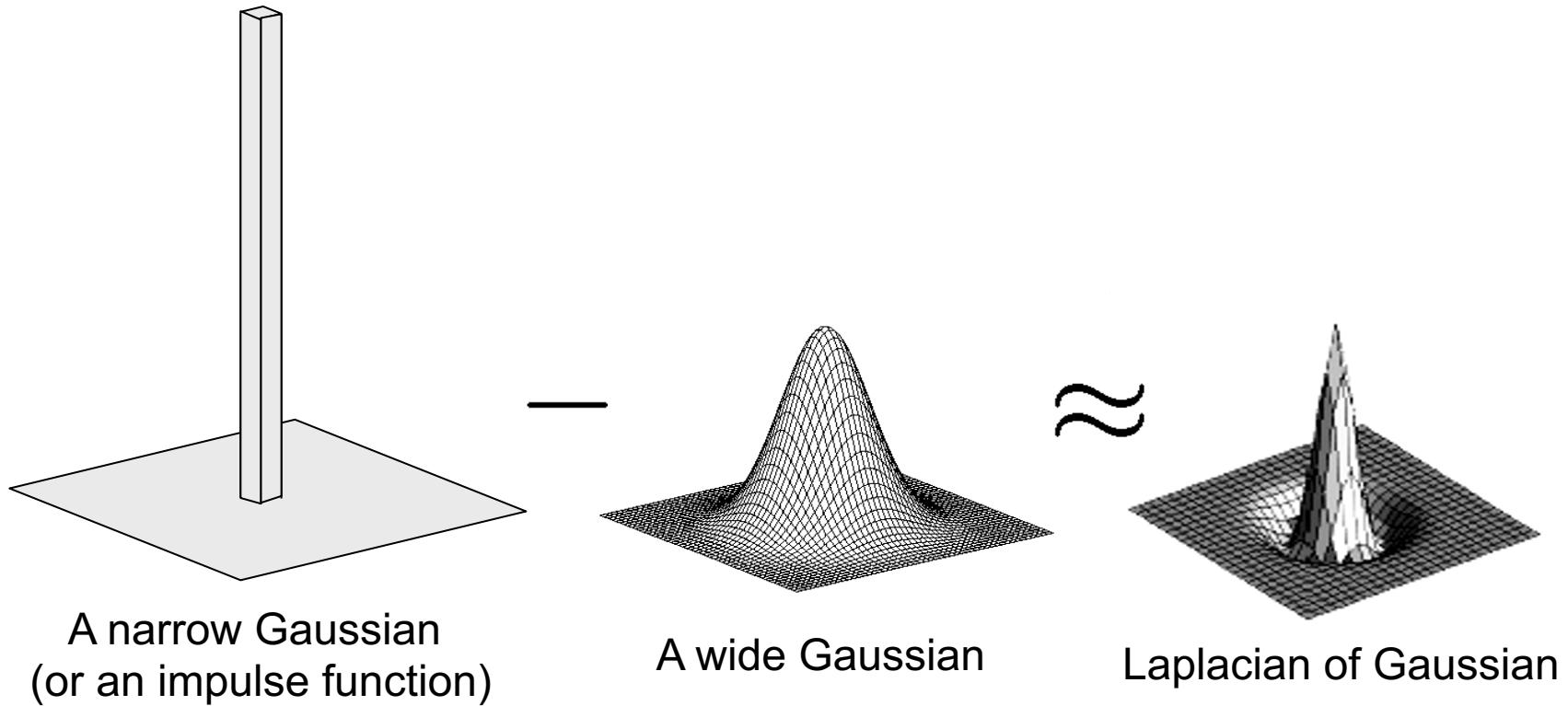
$$\begin{matrix} 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -2 & -9 & -23 & -1 & 103 & 180 & 103 & -1 & -23 & -9 & -2 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & \\ 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & \end{matrix}$$



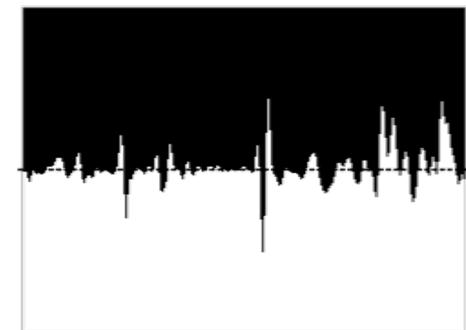
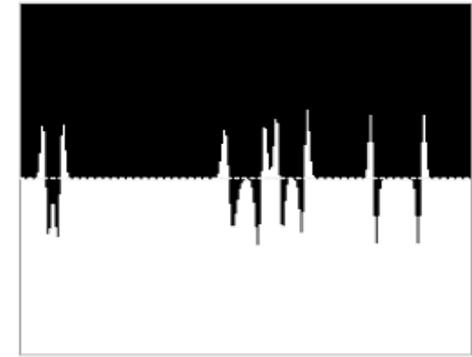
# Detecting zero-crossings

- The edge locations occur at points in the LoG-filtered image where the value passes through zero. i.e., where there is a sign change.
- The number of zero crossings is influenced by the size of the Gaussian; the greater the smoothing, the smaller the number of zero crossings.
- A simple algorithm is to declare a pixel  $p$  to be zero crossing if any one of its 8-neighbors is of opposite sign. To restrict the detected zero crossings to the more significant ones, we can impose a threshold  $T$ .

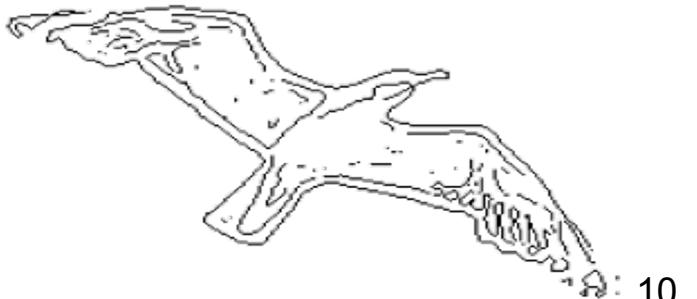
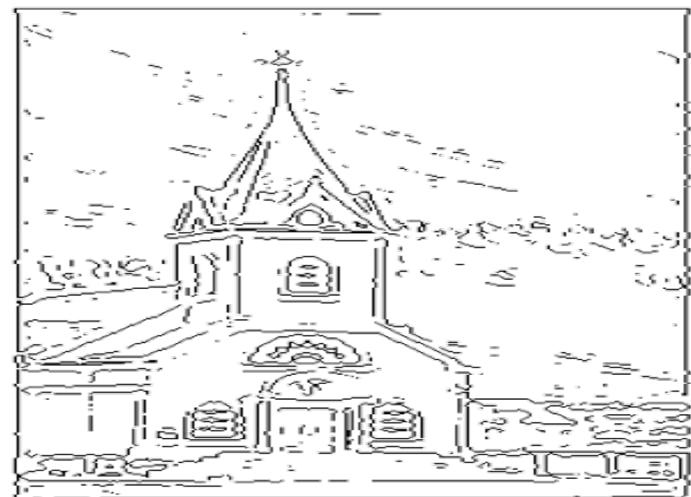
“Laplacian of Gaussian” (LoG) can be approximately computed as “Difference of Gaussians (DoG).



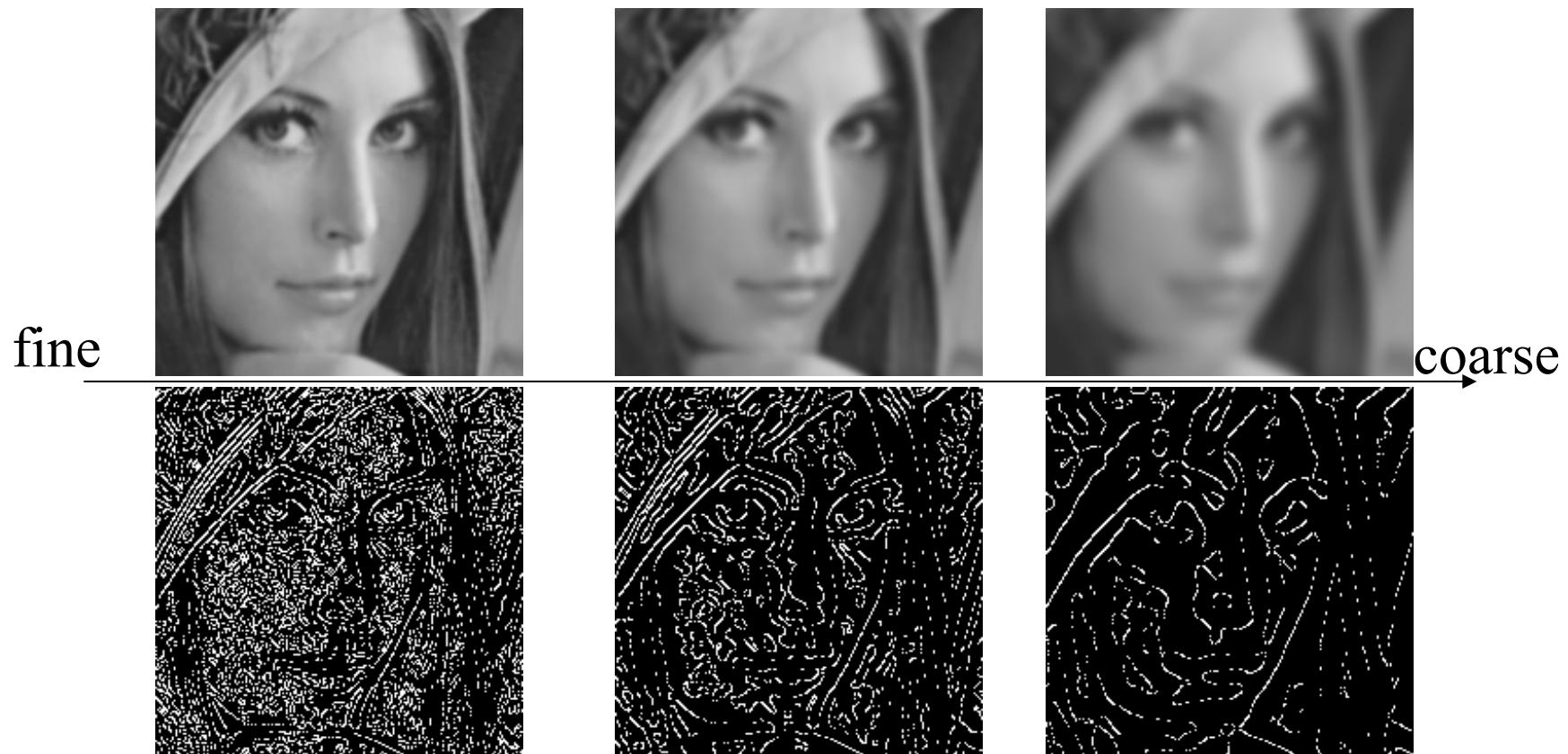
# Example: 2-D LoG filtering



# Example: LoG/Marr edge detector



# Change of Scale (sigma in Gaussian/LoG)





# Sobel vs. LoG

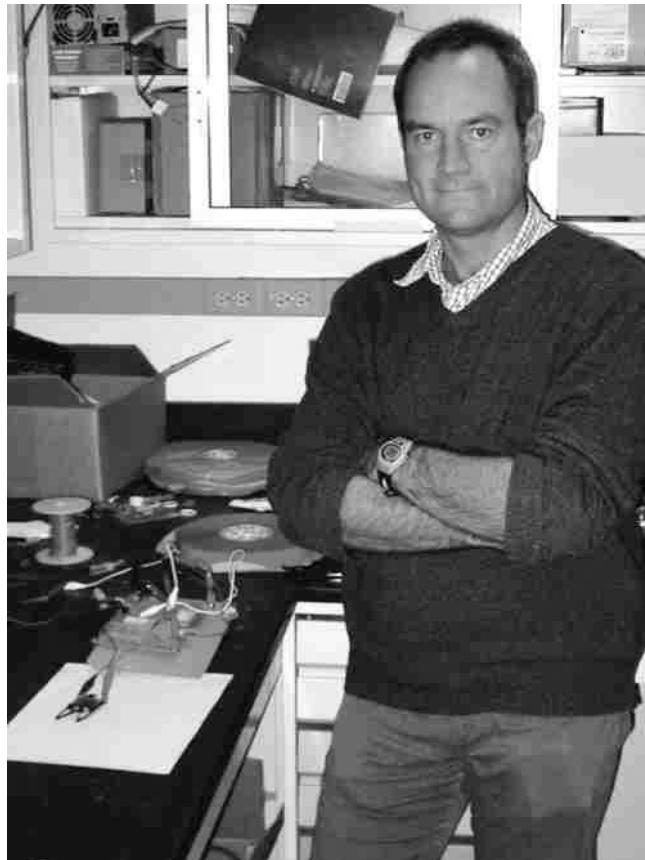


Sobel



LoG

# Canny Edge Detector



[John Canny](#)



**Canny Edge Detector is one of the most successful edge detection method.**

Ref: John F. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 8, No. 16, pp. 679-698, Nov. 1986. 13

# Canny Edge Detector



*(Matlab automatically set thresholds)*

# Example: Canny edge detector



- original image (Lena)

# The Canny edge detector



- norm of the gradient

# The Canny edge detector



- thresholding

# The Canny edge detector



- thinning
- (non-maximum suppression, edge following)

# Canny Edge Detector

- Make use of Derivative of Gaussian
- Non-maximum suppression
  - Thin multi-pixel wide “ridges” down to single pixel
- Optimal thresholding
  - Low, high edge-strength thresholds
  - Accept all edges over low threshold that are connected to edge over high threshold
- Optimal decision making.
- Matlab: `edge(I, 'canny', sigma)`

# Other Edge Detectors

- Machine learning based edge detector.
- To learn how human label edges of natural images.

## Reference:

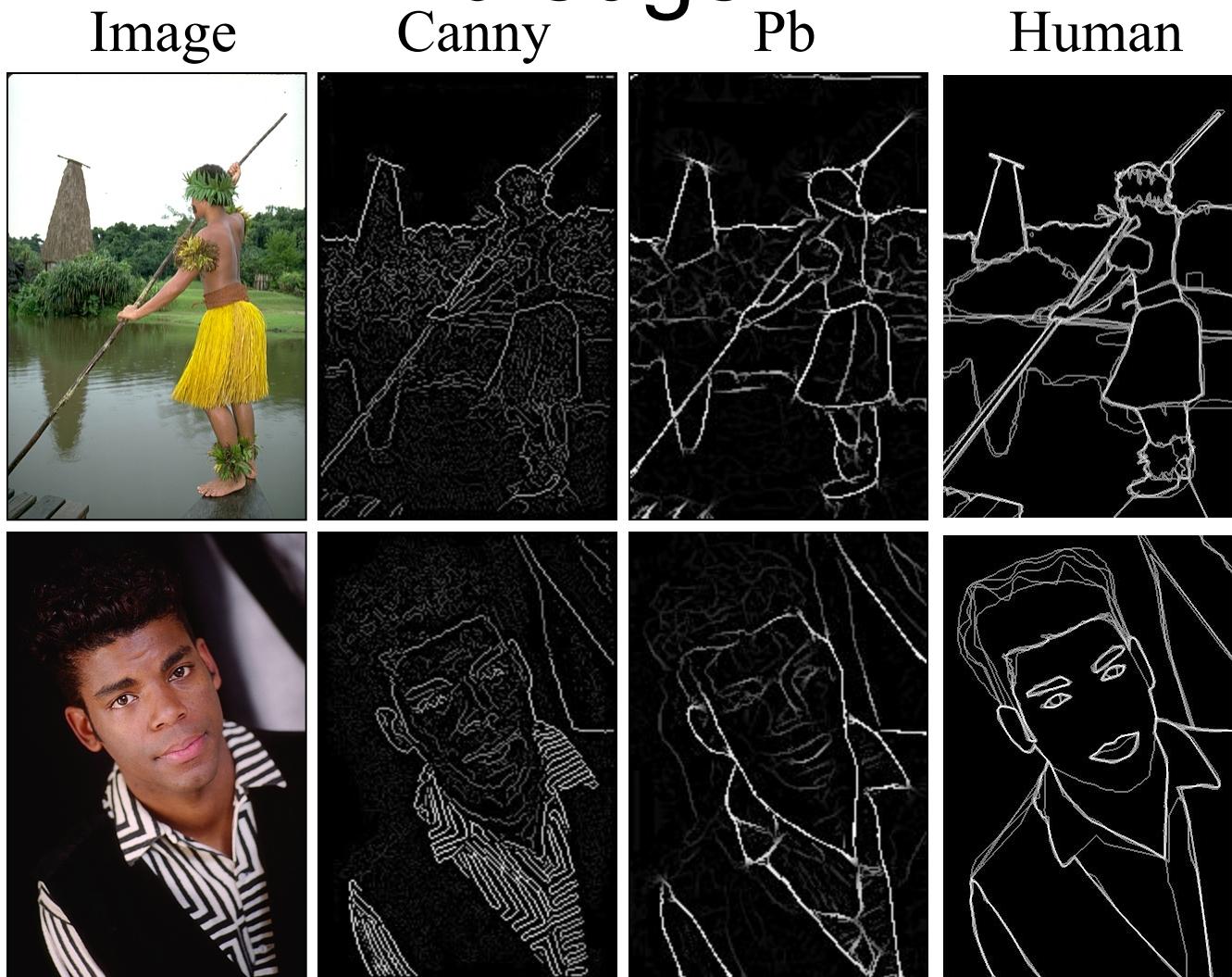
*Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues,*

David R. Martin, Charless C. Fowlkes, and Jitendra Malik, TPAMI, 2004

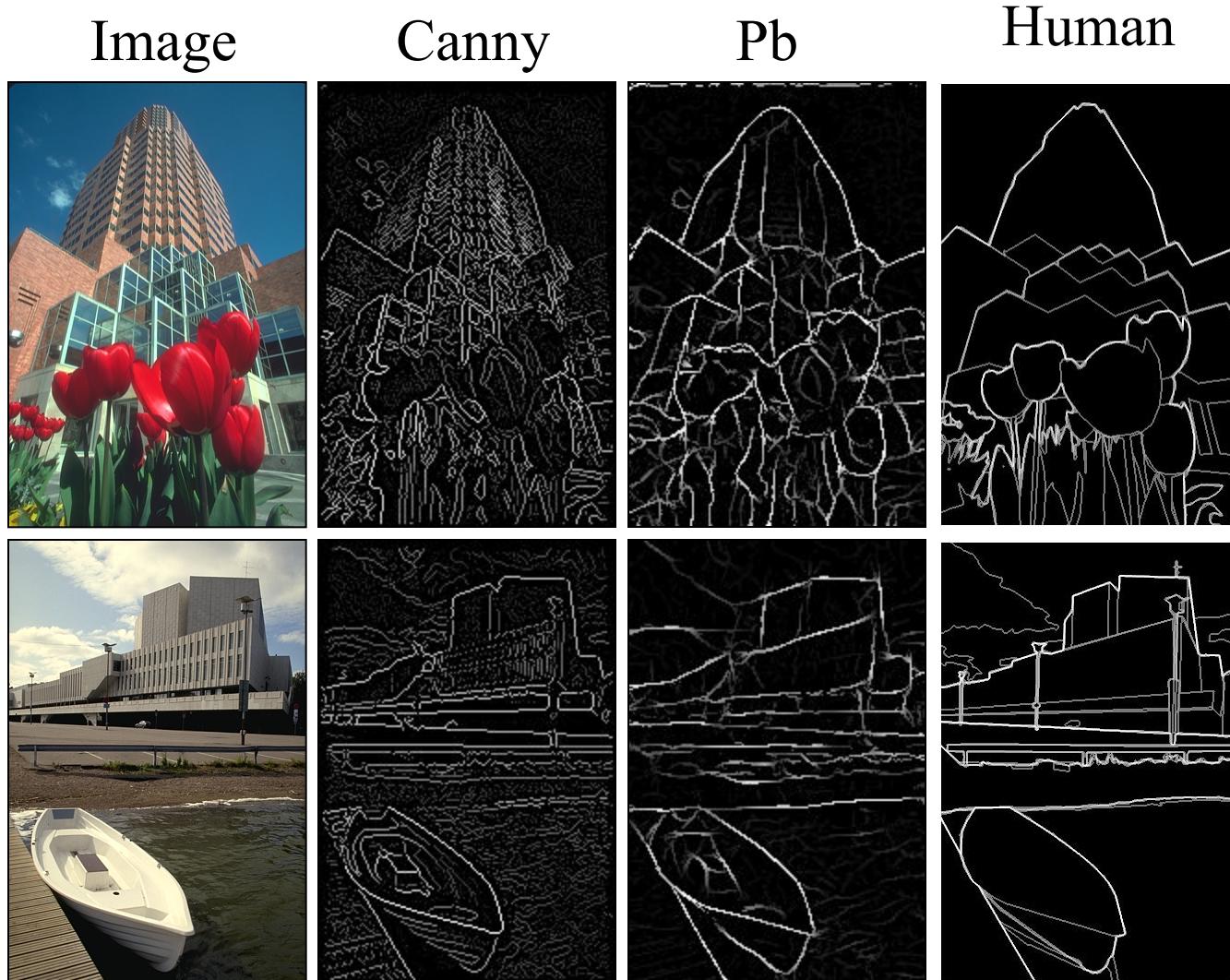
# Sample human-labeled training images and their corresponding edge maps



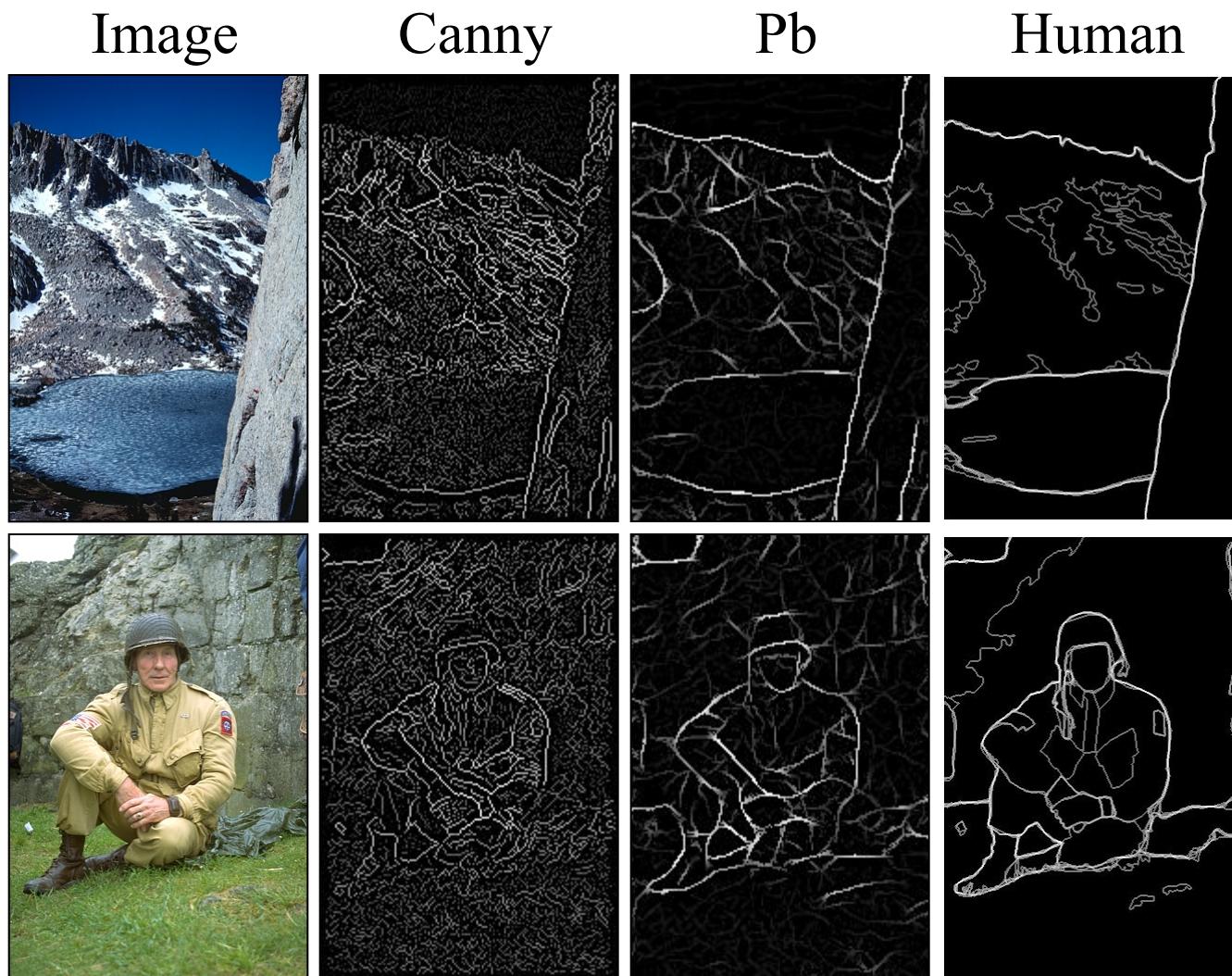
# Learned edge detector: Pb edge I



# Pb result II



# Pb result III



# Line Fitting

# A binary edge map is not enough



# A binary edge map is not enough

- Fragmented edges
- Many broken lines.
- Noise in edge orientations.
- Membership not clear:  
which points on which line ?

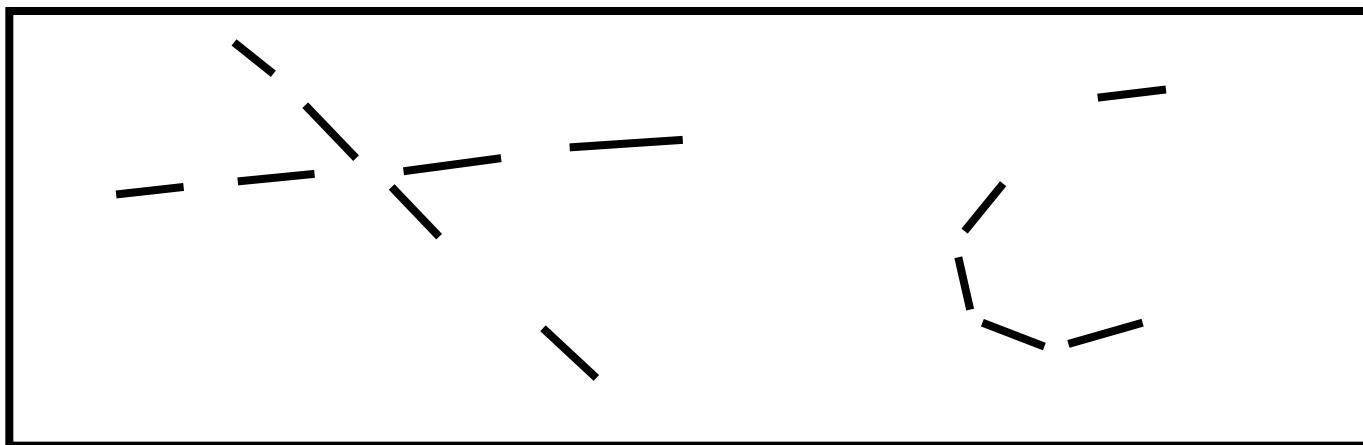


# Vanishing lines (3D Vision)

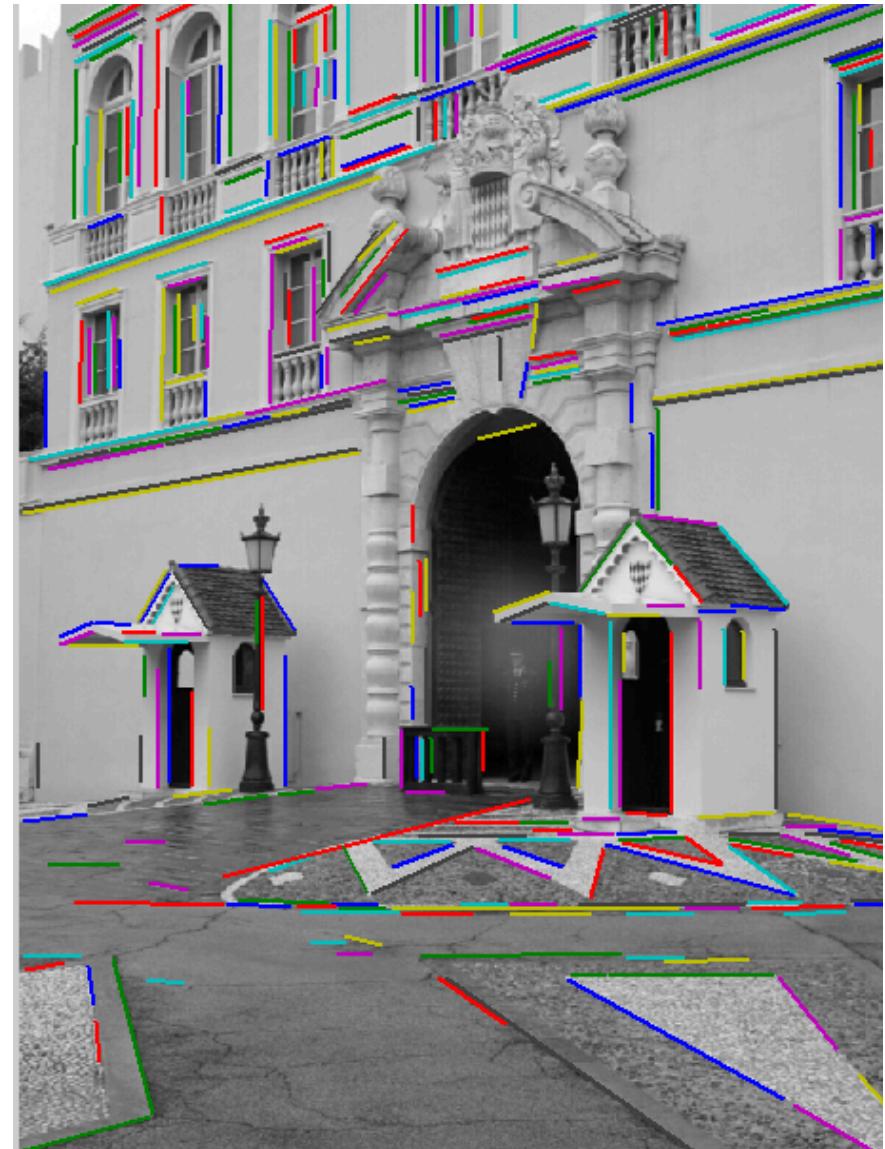


## **Goal of Line Fitting:**

from fragmented edges to straight lines or curves



# Goal of Line Fitting



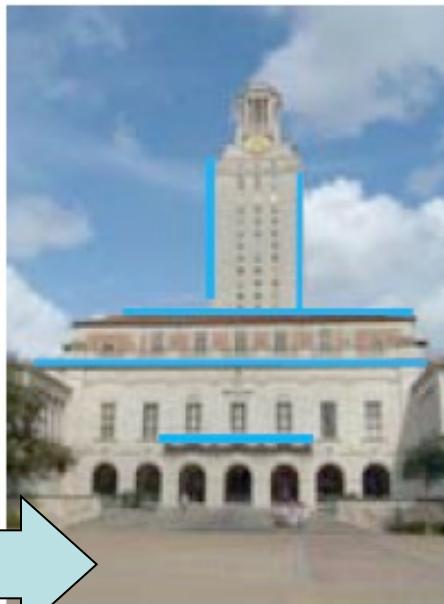
# Content

- Least Squares Line Fitting
- Robust-M Estimator Line Fitting
- RANSAC
- Hough Transform

# “Fitting” in general context

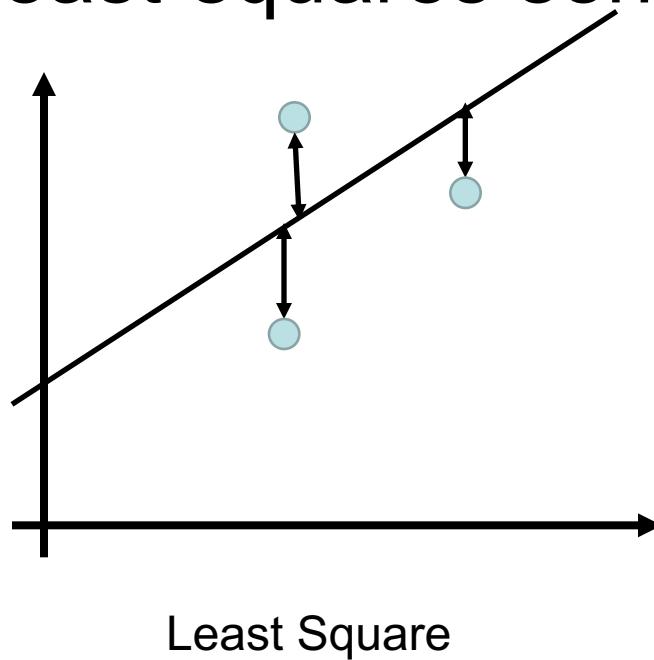
- Choose a parametric model to represent a set of features.
- Membership determination
  - Decide which feature belongs to which model.
  - Not a local decision; must look at the full image globally.
- Key questions
  - What is the best parametric model
  - How many models (e.g. lines) are there
  - Which point belongs to which line
  - Remove noisy points, wrong edge points.

# Line fitting examples



# Basic Line Fitting-Least Square

- Given a set of points *belong to* a straight line, find the equation of the line.
- Find the line parameters that yield minimal error in a least-squares sense.



# Least squares line fitting

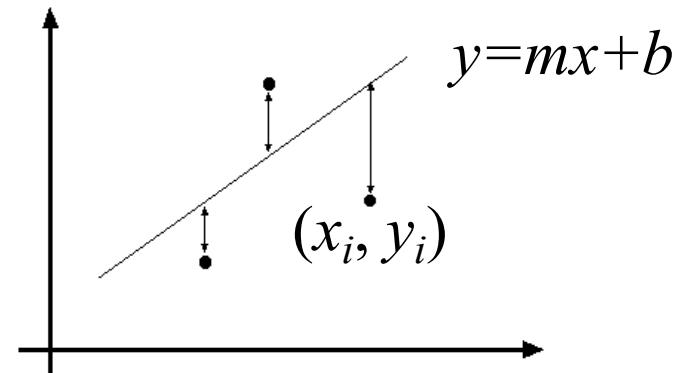
- Data:  $(x_1, y_1), \dots, (x_n, y_n)$

- Line equation:

- $y_i = m x_i + b$

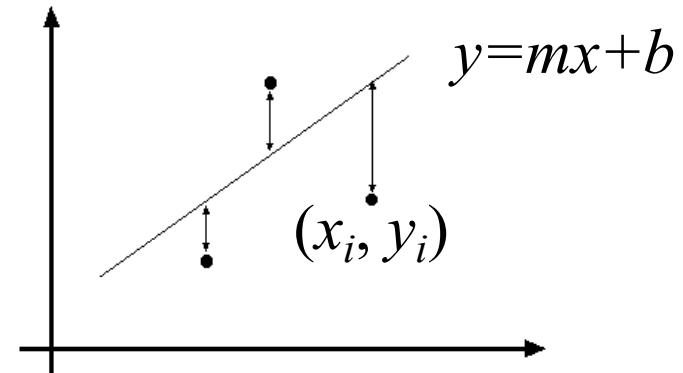
- Find  $(m, b)$  to minimize

$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$



# Least squares line fitting

$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$



**Solution:**

Step 1: Define fitting error for each point as

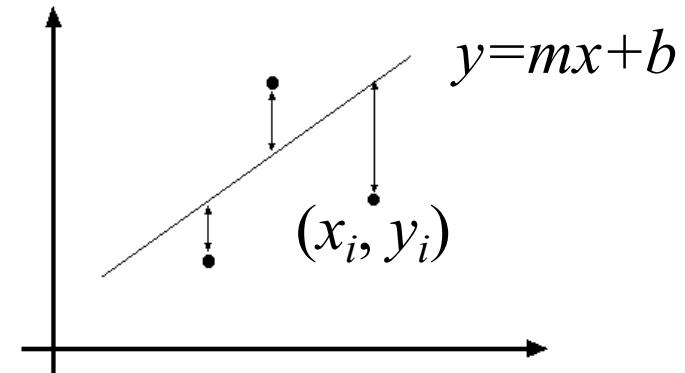
$$e_i = [x_i \ 1] \begin{bmatrix} m \\ b \end{bmatrix} - y_i$$

Step 2: The total fitting error for all the points are

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} m \\ b \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

# Least squares line fitting

$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$



Solution:

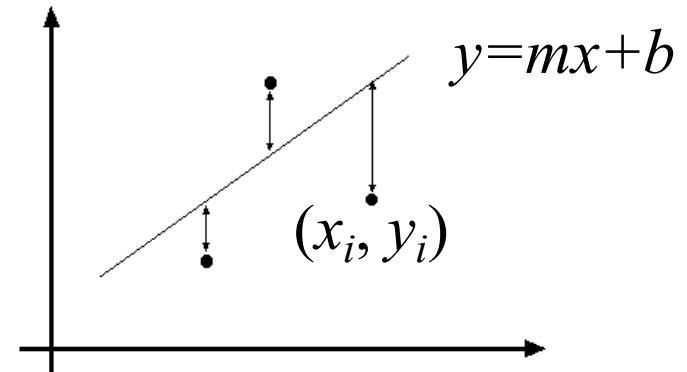
Step 2: The total fitting error for all the points are

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \mathbf{Ap} - \mathbf{y}$$

$$\text{Where, } \mathbf{A} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}, \mathbf{p} = \begin{bmatrix} m \\ b \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

# Least squares line fitting

$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$



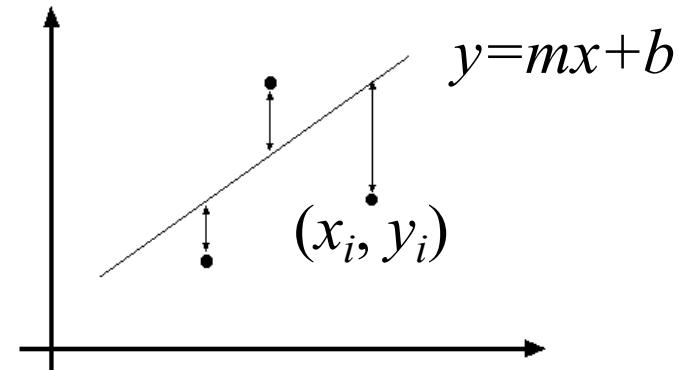
Solution:

Step 3: The errors for all points are

$$\begin{aligned} E &= \sum_{i=1}^n (mx_i + b - y_i)^2 = \mathbf{e}^T \mathbf{e} = (\mathbf{Ap} - \mathbf{y})^T (\mathbf{Ap} - \mathbf{y}) \\ &= \mathbf{y}^T \mathbf{y} - 2(\mathbf{Ap})^T \mathbf{y} + (\mathbf{Ap})^T \mathbf{Ap} \end{aligned}$$

# Least squares line fitting

$$E = \sum_{i=1}^n (y_i - mx_i - b)^2$$



**Solution:**

Step 3: find solution to  $\mathbf{p}$ , where  $\mathbf{p} = \begin{bmatrix} m \\ b \end{bmatrix}$

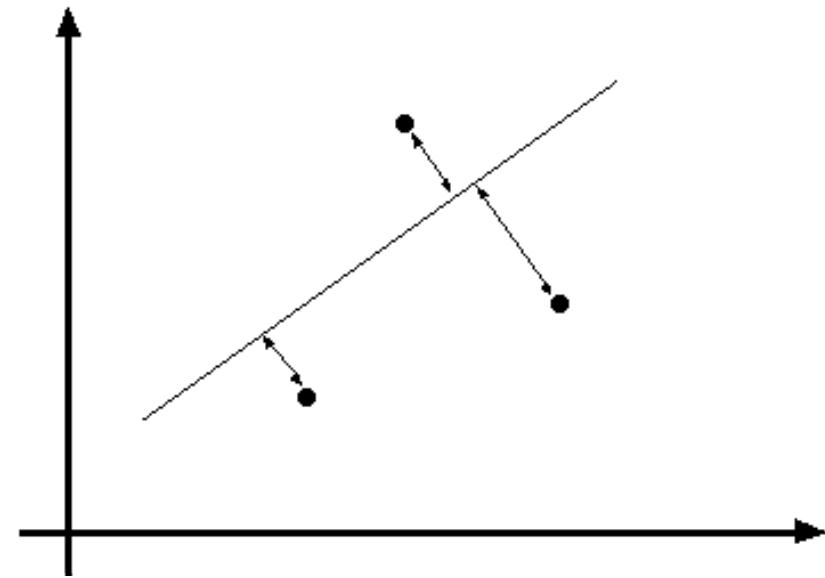
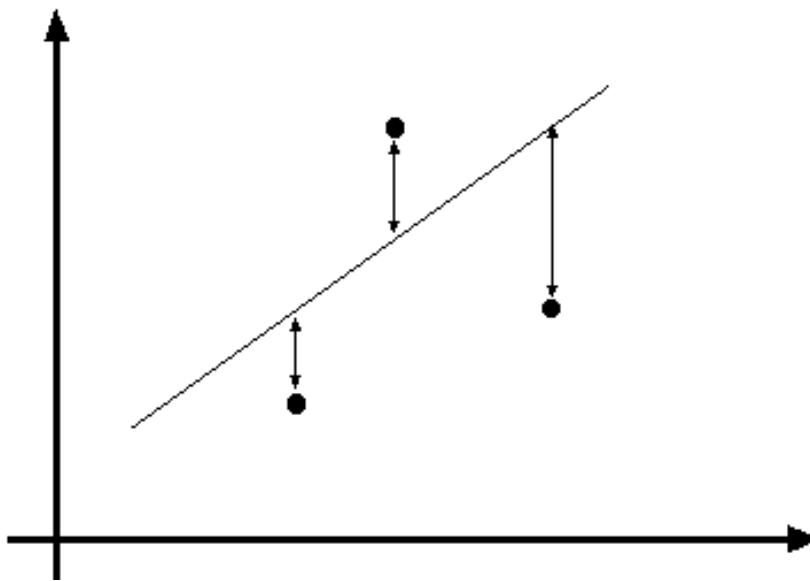
$$\frac{dE}{d\mathbf{p}} = 2\mathbf{A}^T \mathbf{A}\mathbf{p} - 2\mathbf{A}^T \mathbf{y} = 0$$

$$\rightarrow \mathbf{p} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$$

<https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>

# Issues with LS fitting

- Not rotation-invariant
- Fails completely for vertical lines
- Modified least squares using a perpendicular distance



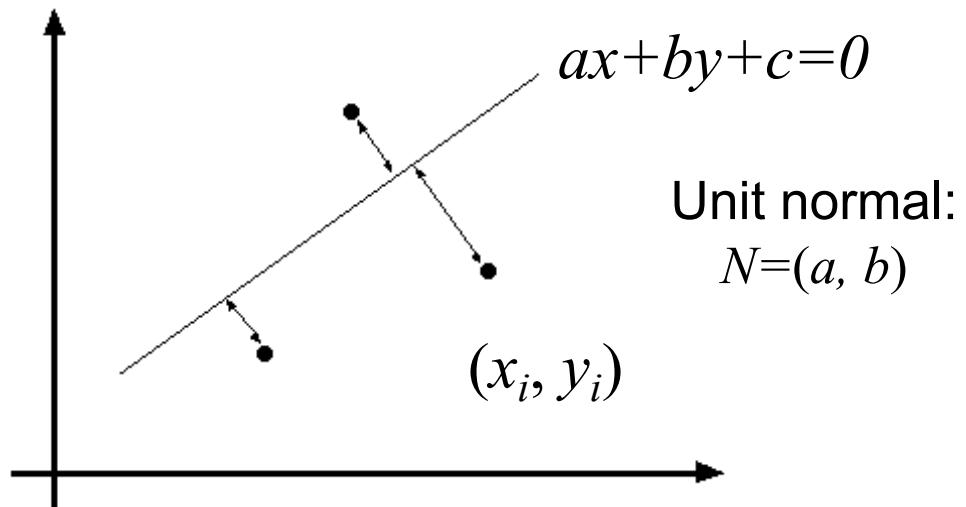
# Total least squares

Point to line: Perpendicular Distance

$$d = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}}$$

If ( $a^2+b^2=1$ ) then the distance between point  $(x_i, y_i)$  to line is

$$|ax_i + by_i + c|$$



Unit normal:  
 $N=(a, b)$

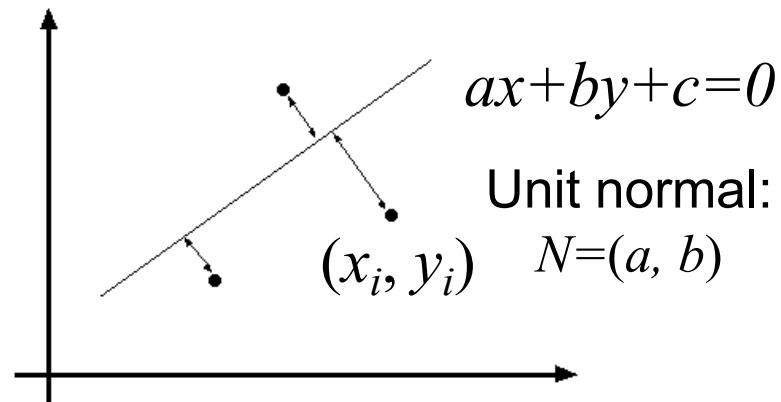
# Total least squares

If  $(a^2+b^2=1)$  then the distance between point  $(x_i, y_i)$  to the line is

$$|ax_i + by_i + c|$$

Find  $(a, b, c)$  to minimize the sum of squared perpendicular distances

$$E = \sum_{i=1}^n (ax_i + by_i + c)^2$$

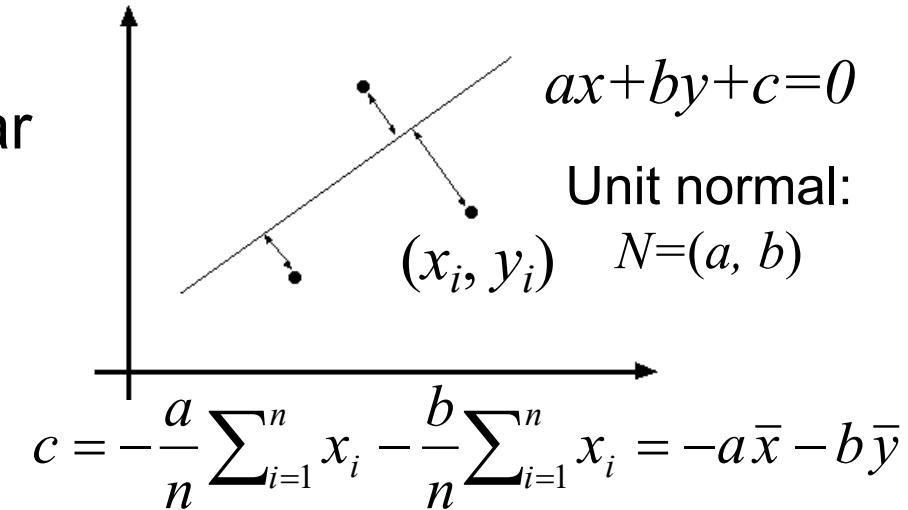


# Total least squares

Find  $(a, b, c)$  to minimize the sum of squared perpendicular distances

$$E = \sum_{i=1}^n (ax_i + by_i + c)^2$$

$$\frac{\partial E}{\partial c} = \sum_{i=1}^n -2(ax_i + by_i + c) = 0$$



$$E = \sum_{i=1}^n (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 = \mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p}$$

$$\text{minimize } \mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p} \quad \text{s.t. } \mathbf{p}^T \mathbf{p} = 1 \quad \Rightarrow \quad \text{minimize } \frac{\mathbf{p}^T \mathbf{A}^T \mathbf{A} \mathbf{p}}{\mathbf{p}^T \mathbf{p}}$$

**Solution is the eigenvector corresponding to the smallest eigenvalue of  $\mathbf{A}^T \mathbf{A}$ .**

# Recap: Two Common Optimization Problems

## LS Problem statement

$$\text{minimize } \|\mathbf{Ax} - \mathbf{b}\|^2$$

least squares solution to  $\mathbf{Ax} = \mathbf{b}$

## Solution

$$\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$$

$\mathbf{x} = \mathbf{A} \setminus \mathbf{b}$  (matlab)

## TLS Problem statement

$$\text{minimize } \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} \text{ s.t. } \mathbf{x}^T \mathbf{x} = 1$$

$$\text{minimize } \frac{\mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$$

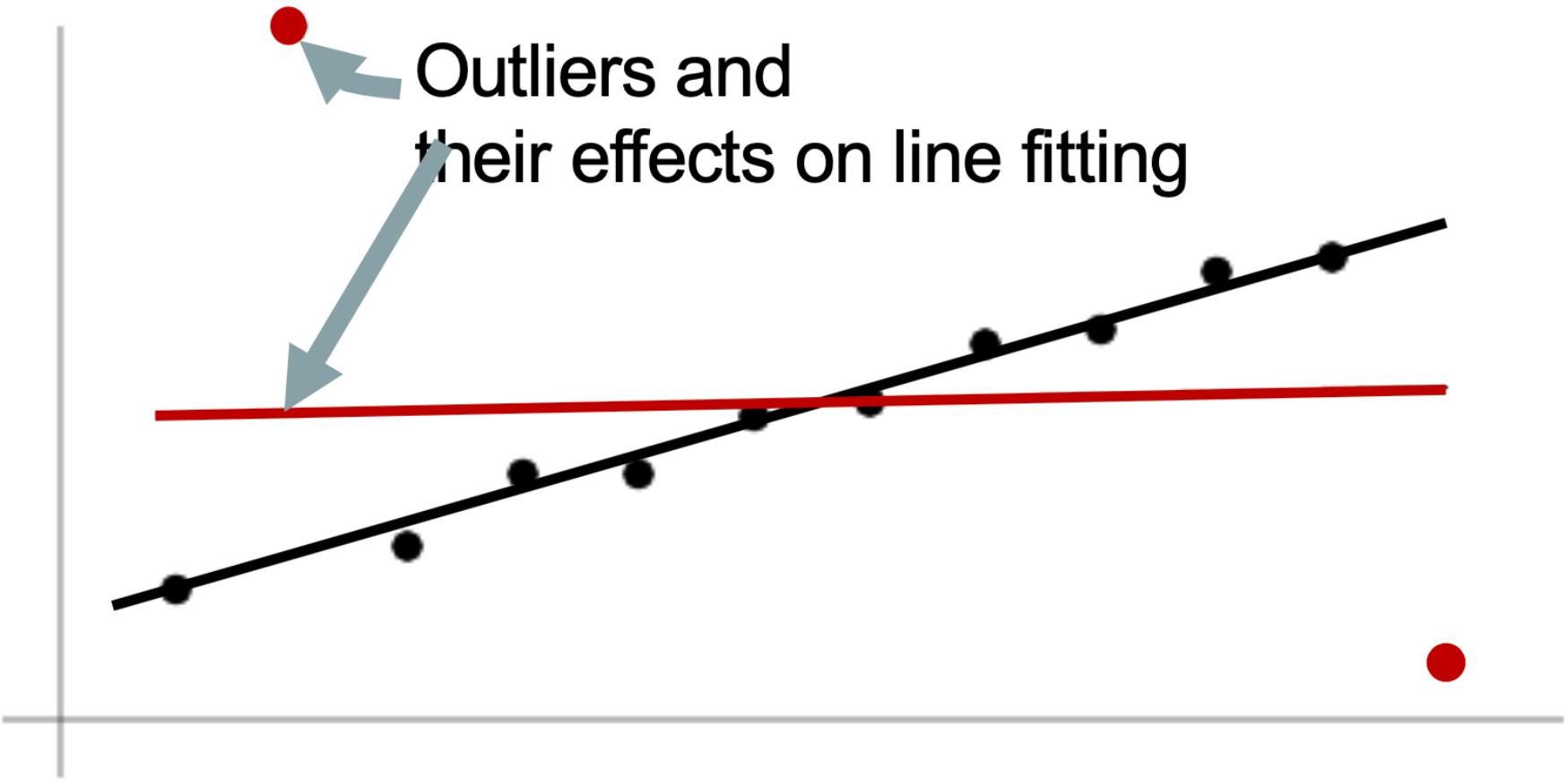
non-trivial TLS solution to  $\mathbf{A} \mathbf{x} = 0$

## Solution

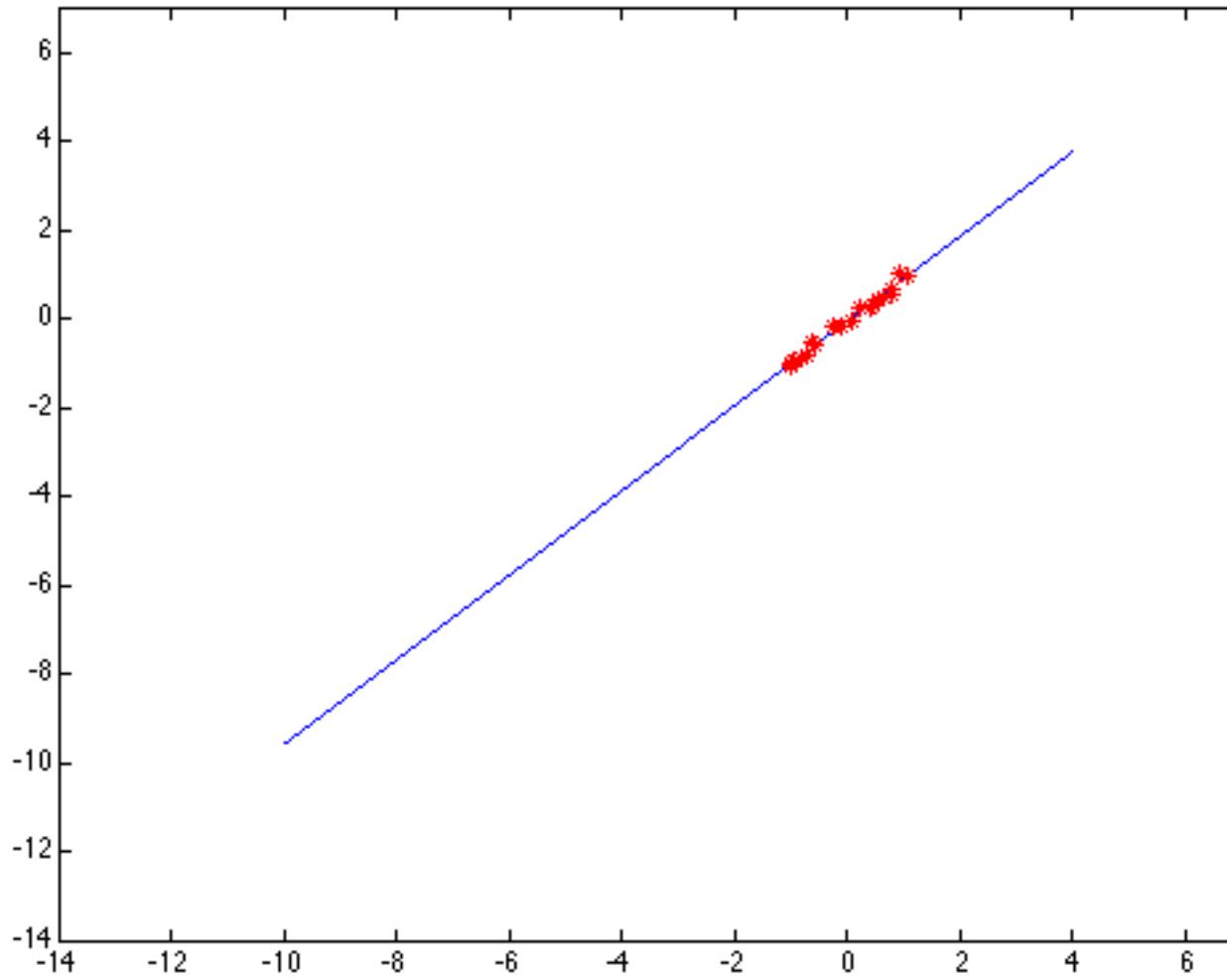
$$[\mathbf{v}, \lambda] = \text{eig}(\mathbf{A}^T \mathbf{A})$$

$$\lambda_1 < \lambda_{2..n} : \mathbf{x} = \mathbf{v}_1$$

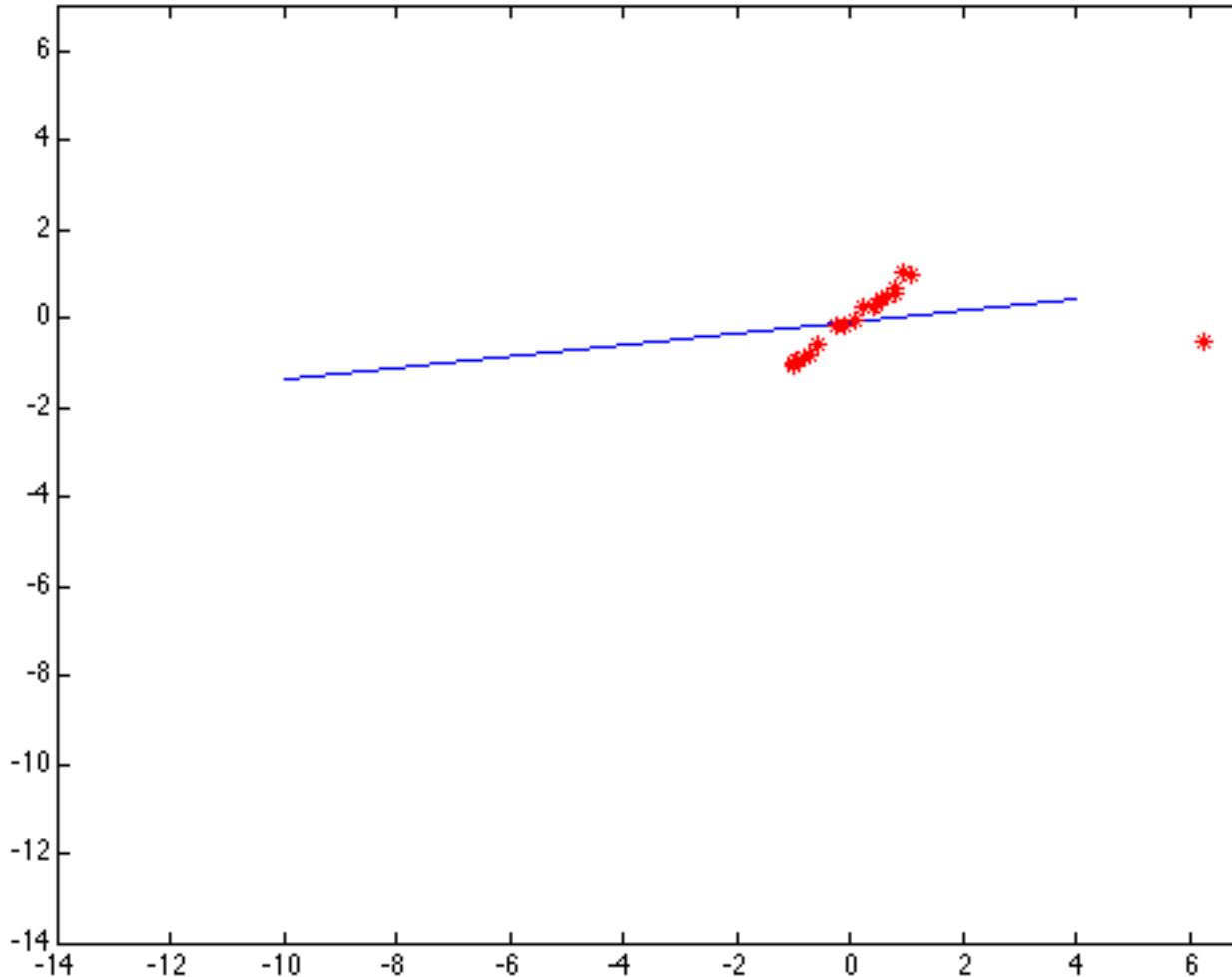
# Basic line fitting: Not Robust to Outliers



# Least Squares: Robustness to noise



# Least Squares: Robustness to noise



Problem: squared error heavily penalizes outliers 53

# Robust M-estimators

## m-estimators

Least-squares:

$$\text{minimize} \quad \|Ax - b\|^2$$

M-estimator

$$\text{minimize} \quad h(\|Ax - b\|)$$

or given  $f_i : \mathbb{R}^m \rightarrow \mathbb{R}$  for  $i = 1, \dots, n$ :

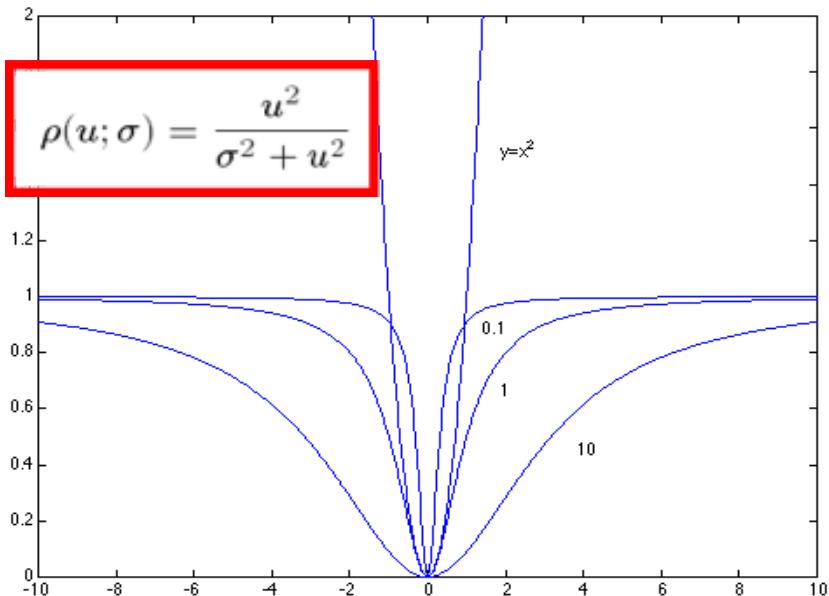
$$\text{minimize} \quad \sum_{i=1}^n h \circ f_i(\mathbf{x})$$

# Robust M-estimator

- General approach: find model parameters  $\theta$  that minimize

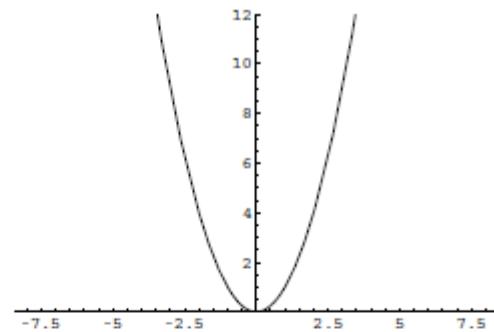
$$\sum_i \rho(r_i(x_i, \theta), \alpha)$$

$r_i(x_i, \theta)$  – residual of  $i$ th point w.r.t. model parameters  $\theta$ ,  $\rho$  – robust function with scale parameter  $\sigma$

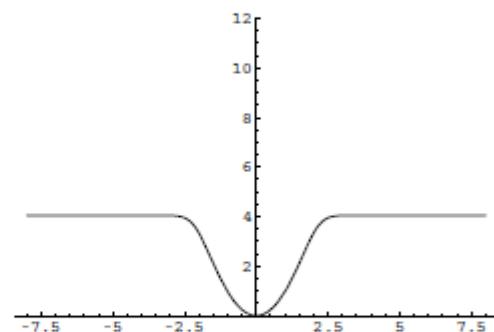


The robust function  $\rho$  behaves like squared distance for small values of the residual  $u$  but saturates for larger values of  $u$

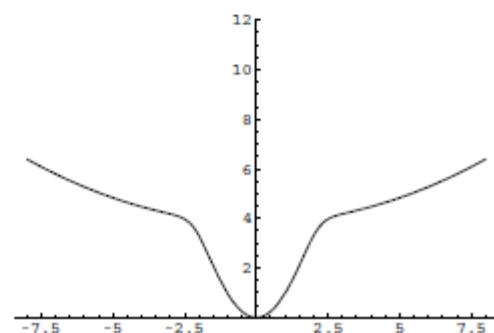
Squared-error



Blake-Zisserman

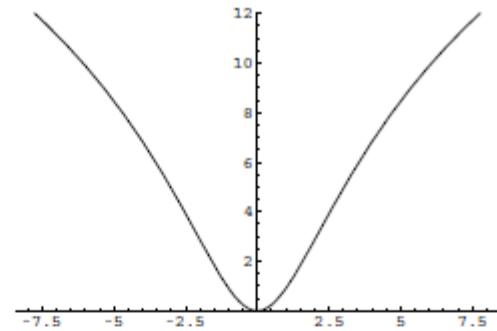


corrupted Gaussian

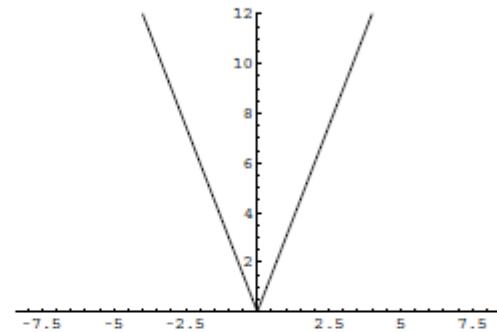


## Various robust measures

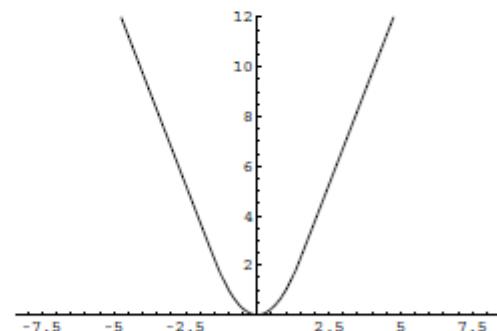
Cauchy



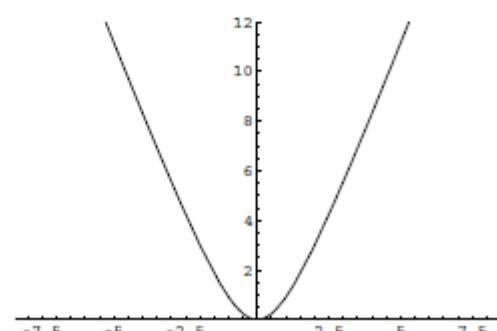
L1



Huber



pseudo-Huber



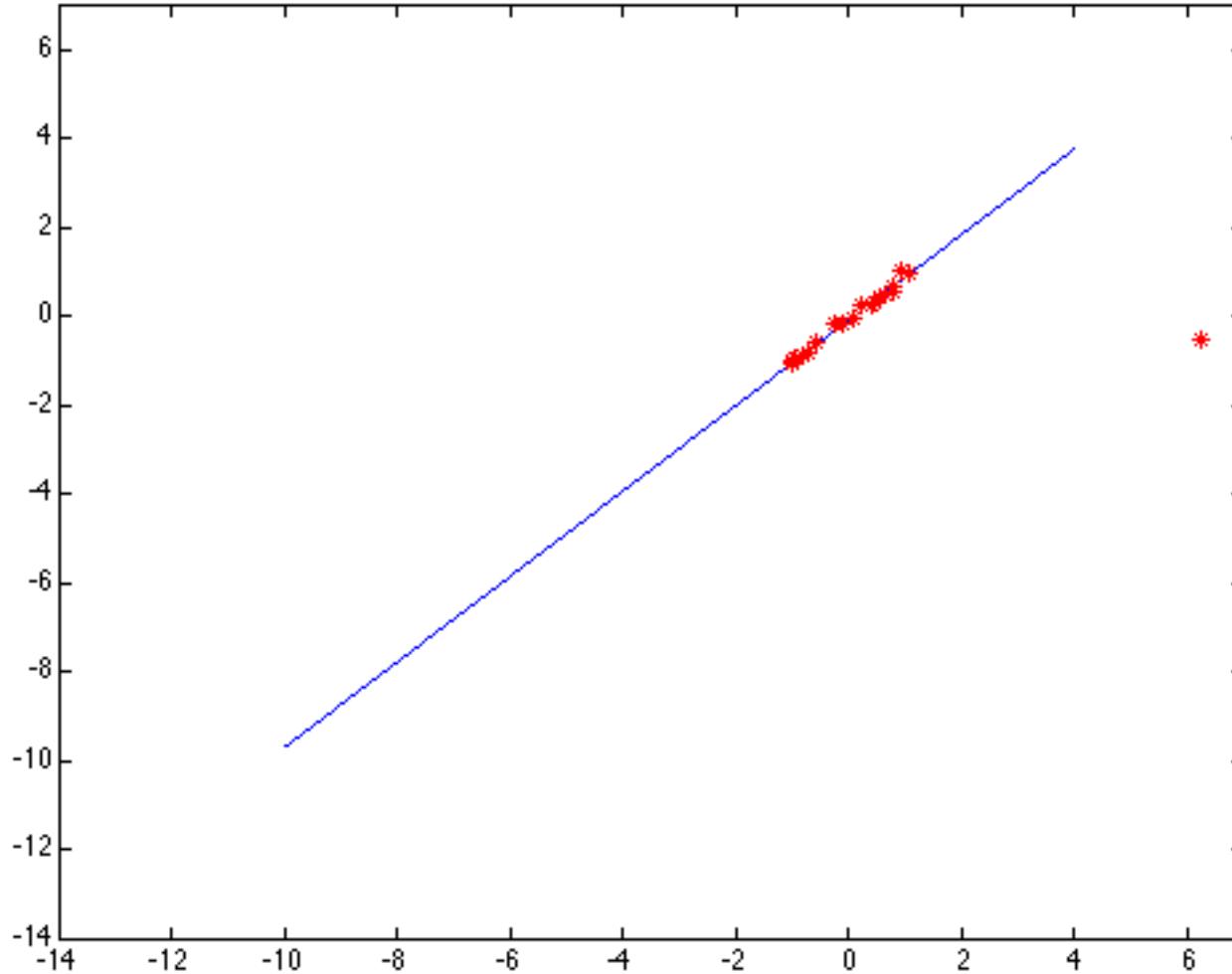
# Robust M-estimator

- General approach: find model parameters  $\theta$  that minimize

$r_i(x_i, \theta)$  – residual of  $i^{\text{th}}$  point w.r.t. model parameters  $\theta$ ,  $\rho$  – robust function with scale parameter  $\sigma$

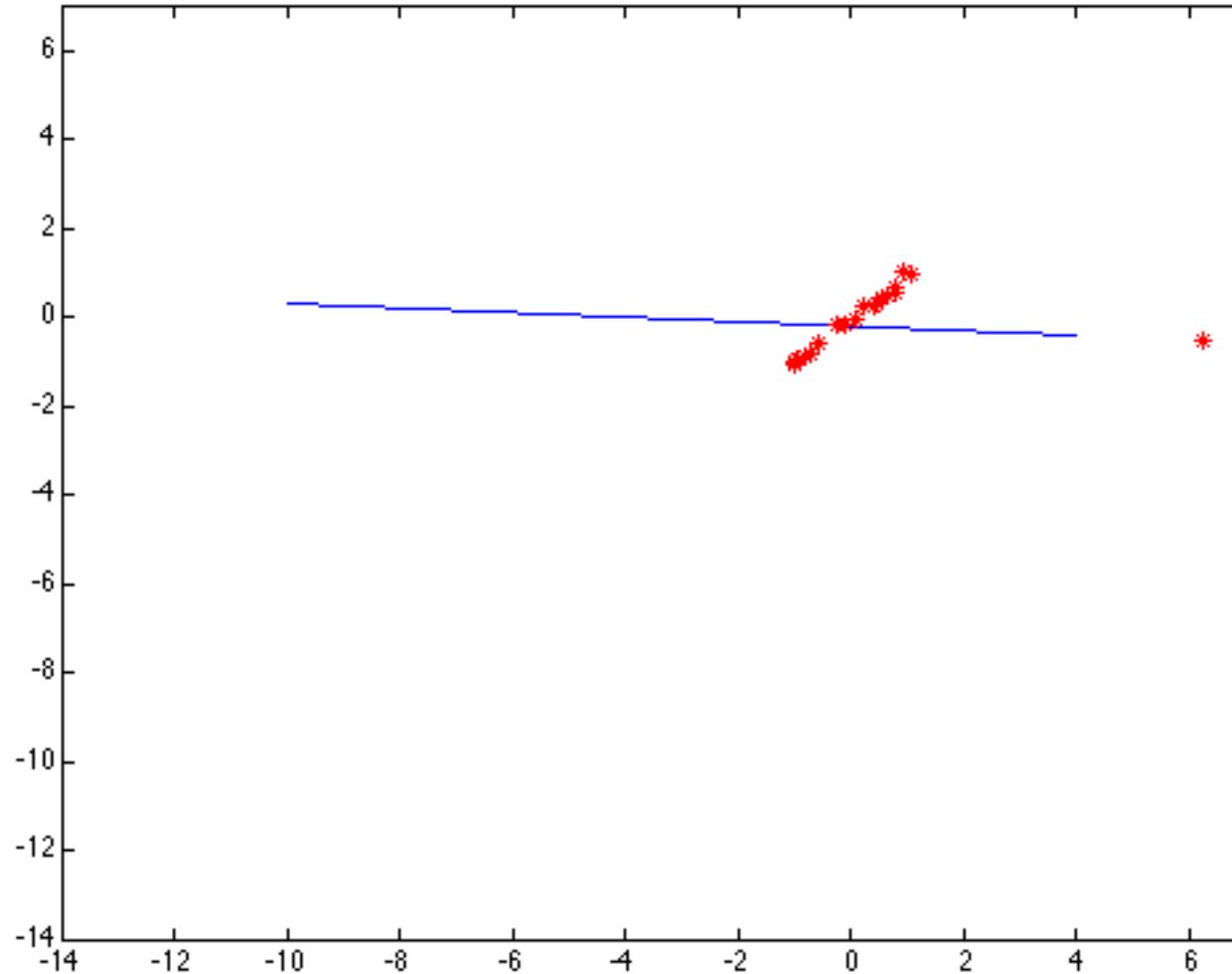
- Robust fitting is a nonlinear optimization problem that must be solved iteratively
- Least squares solution can be used for initialization
- Scale of robust function should be chosen carefully

# Choosing the scale: Just right



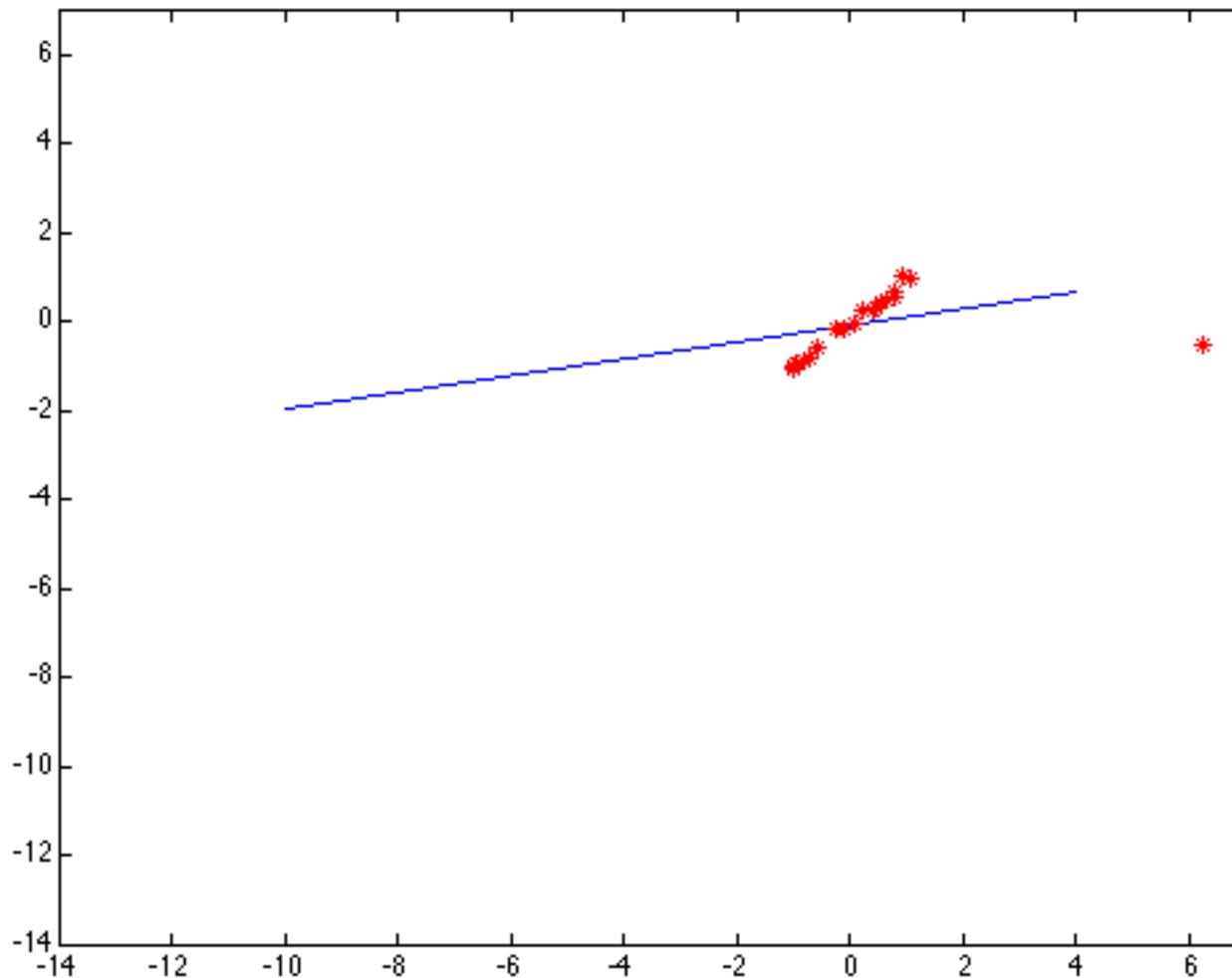
The effect of the outlier is minimized

# Choosing the scale: Too small



The error value is almost the same for every point and the fit is very poor

# Choosing the scale: Too large



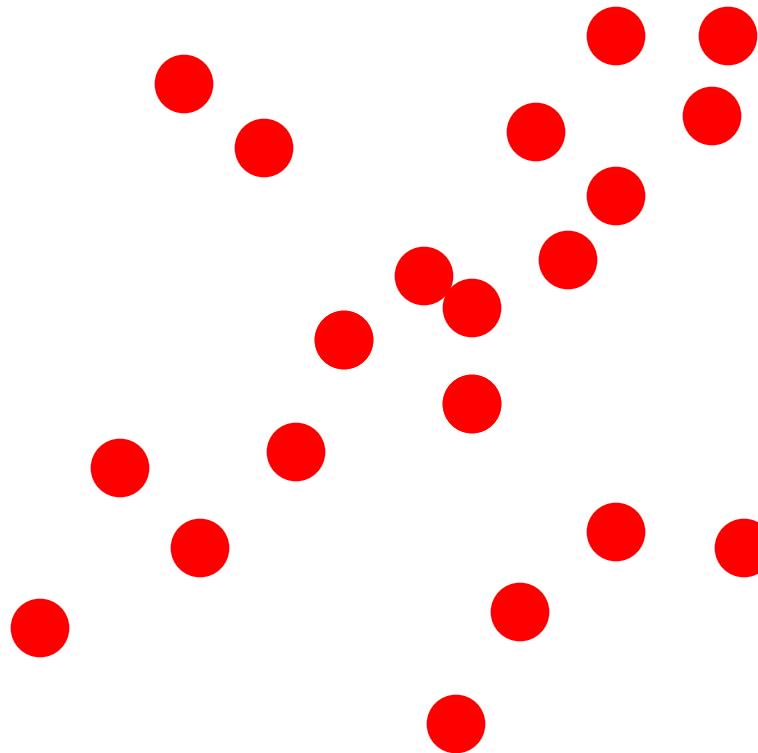
Behaves much the same as least squares

# RANSAC Line fitting

# RANSAC

(RANdom SAmples Consensus) :

Fischler & Bolles in '81.



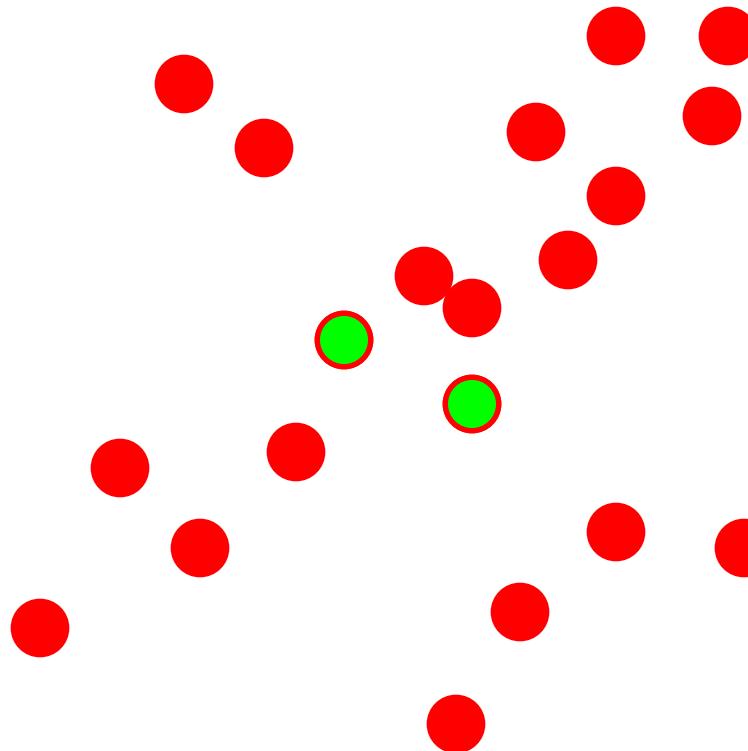
## Algorithm:

1. Randomly **sample** the **minimum** number of points required to fit the model
2. **Solve** for model parameters using these minimum set of samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

# RANSAC

Line fitting example



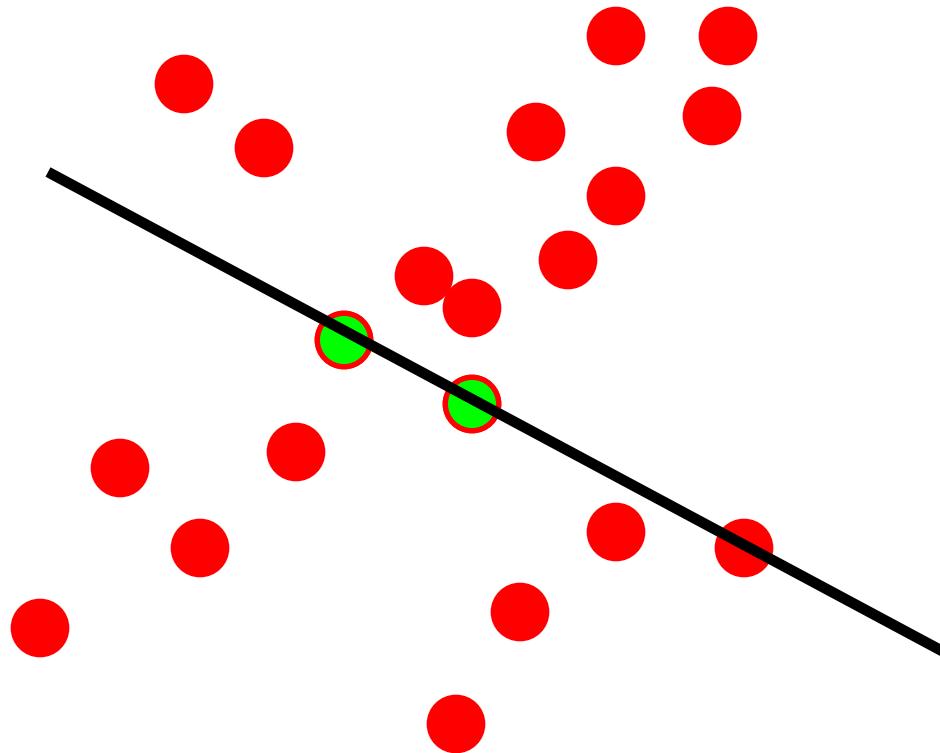
Algorithm:

1. **Randomly sample** the minimum number of points required to fit the model (#=2)
2. **Solve** for model parameters using the minimum set of samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

# RANSAC

Line fitting example



Algorithm:

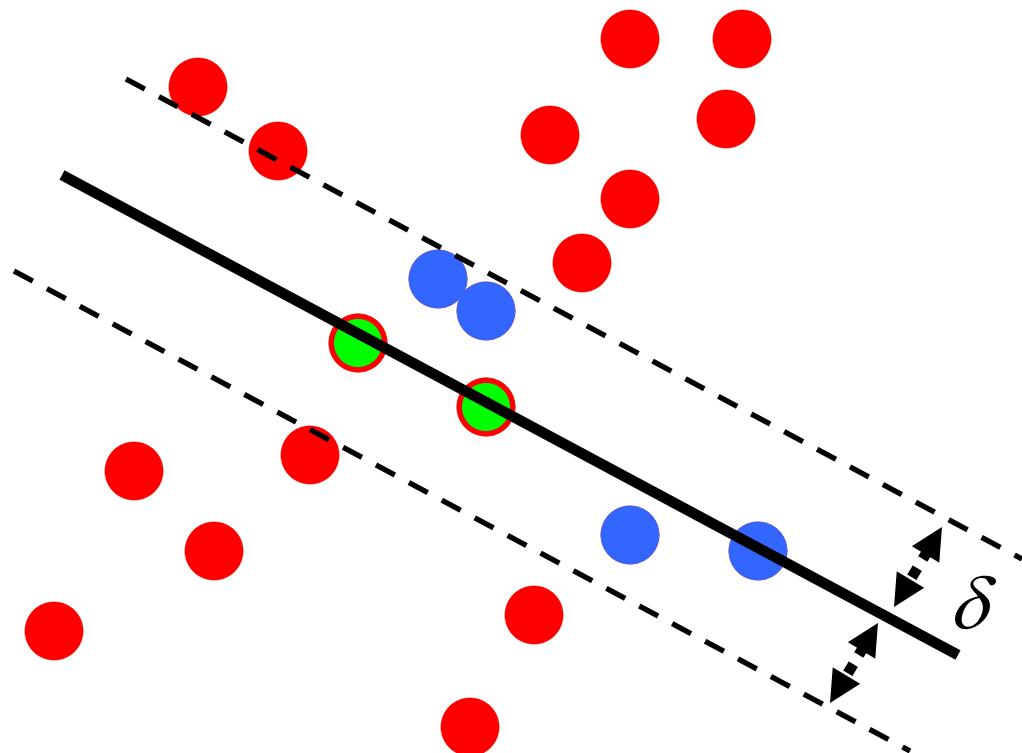
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

# RANSAC

Line fitting example

$$N_I = 6$$

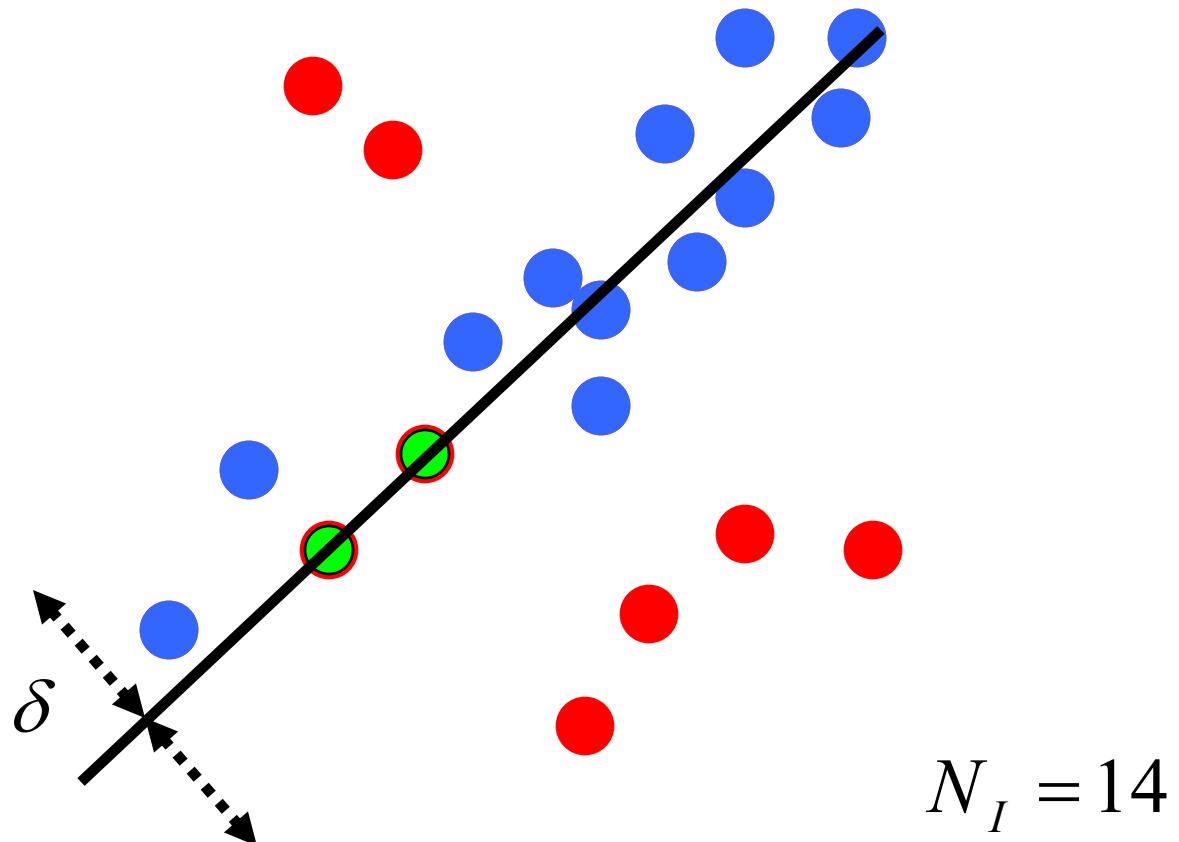


Algorithm:

1. **Sample** randomly the number of points required to fit the model (#=2)
2. **Solve** for model parameters using the minimum set of samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

# RANSAC



Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using the minimum set of samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

# How to choose parameters? (optional)

- Number of samples  $N$ 
  - Choose  $N$  so that, with probability  $p$ , at least one random sample is free from outliers (e.g.  $p=0.99$ ) (outlier ratio:  $e$  )
- Number of sampled points  $s$ 
  - Minimum number needed to fit the model
- Distance threshold  $\delta$ 
  - Choose  $\delta$  so that a good point with noise is likely (e.g., prob=0.95) within threshold

$$N = \log(1 - p) / \log\left(1 - (1 - e)^s\right)$$

# How Many Samples to Choose?

e = probability that a point is an outlier

s = number of points in a sample

N = number of samples (we want to compute this)

p = desired probability that we get a good sample

**Solve the following for N:**

$$1 - (1 - (1 - e)^s)^N = p$$

**Where in the world did that come from? ....**

# How Many Samples to Choose?

$e$  = probability that a point is an outlier

$s$  = number of points in a sample

$N$  = number of samples (we want to compute this)

$p$  = desired probability that we get a good sample

$$1 - (1 - (1 - e)^s)^N = p$$

**Probability that choosing  
one point yields an inlier**

## How Many Samples to Choose?

e = probability that a point is an outlier

s = number of points in a sample

N = number of samples (we want to compute this)

p = desired probability that we get a good sample

$$1 - (1 - (1 - e)^s)^N = p$$

Probability of choosing  
s inliers in a row (sample  
only contains inliers)

# How Many Samples to Choose?

e = probability that a point is an outlier

s = number of points in a sample

N = number of samples (we want to compute this)

p = desired probability that we get a good sample

$$1 - \underbrace{(1 - (1 - e)^s)}^N = p$$

**Probability that N samples  
were contaminated.**

# How Many Samples to Choose?

e = probability that a point is an outlier

s = number of points in a sample

N = number of samples (we want to compute this)

p = desired probability that we get a good sample

$$\underbrace{1 - (1 - (1 - e)^s)^N}_{\text{Probability that at least one sample was not contaminated}} = p$$

**Probability that at least one sample was not contaminated  
(at least one sample of s points is composed of only inliers).**

# How many samples?

Choose  $N$  so that, with probability  $p$ , at least one random sample is free from outliers. e.g.  $p=0.99$

$$(1 - (1 - e)^s)^N = 1 - p$$

$$N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

s	proportion of outliers $e$							
	5%	10%	20%	25%	30%	40%	50%	
2	2	3	5	6	7	11	17	
3	3	4	7	9	11	19	35	
4	3	5	9	13	17	34	72	
5	4	6	12	17	26	57	146	
6	4	7	16	24	37	97	293	
7	4	8	20	33	54	163	588	
8	5	9	26	44	78	272	1177	

# RANSAC pros and cons

## Pros

- Robust to outliers
- Applicable for large number of parameters
- Parameters are easy to choose

## Cons

- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

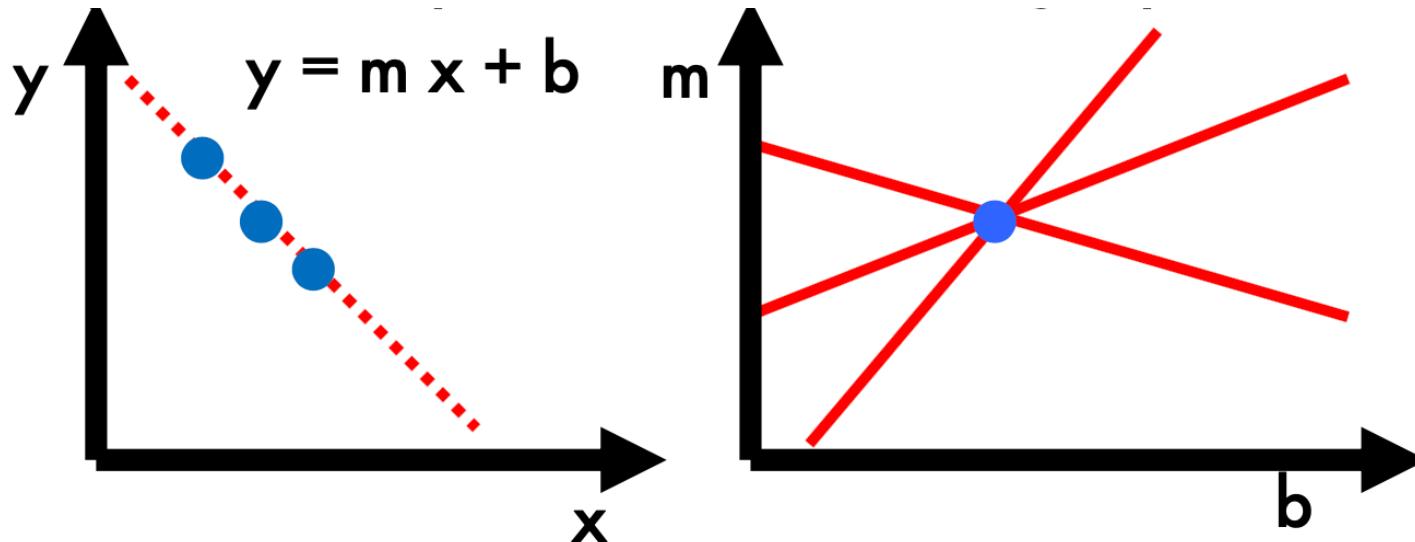
## Common applications (for 3D vision)

- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)

# Hough Transform

# Fitting Multiple Lines using Hough Transform

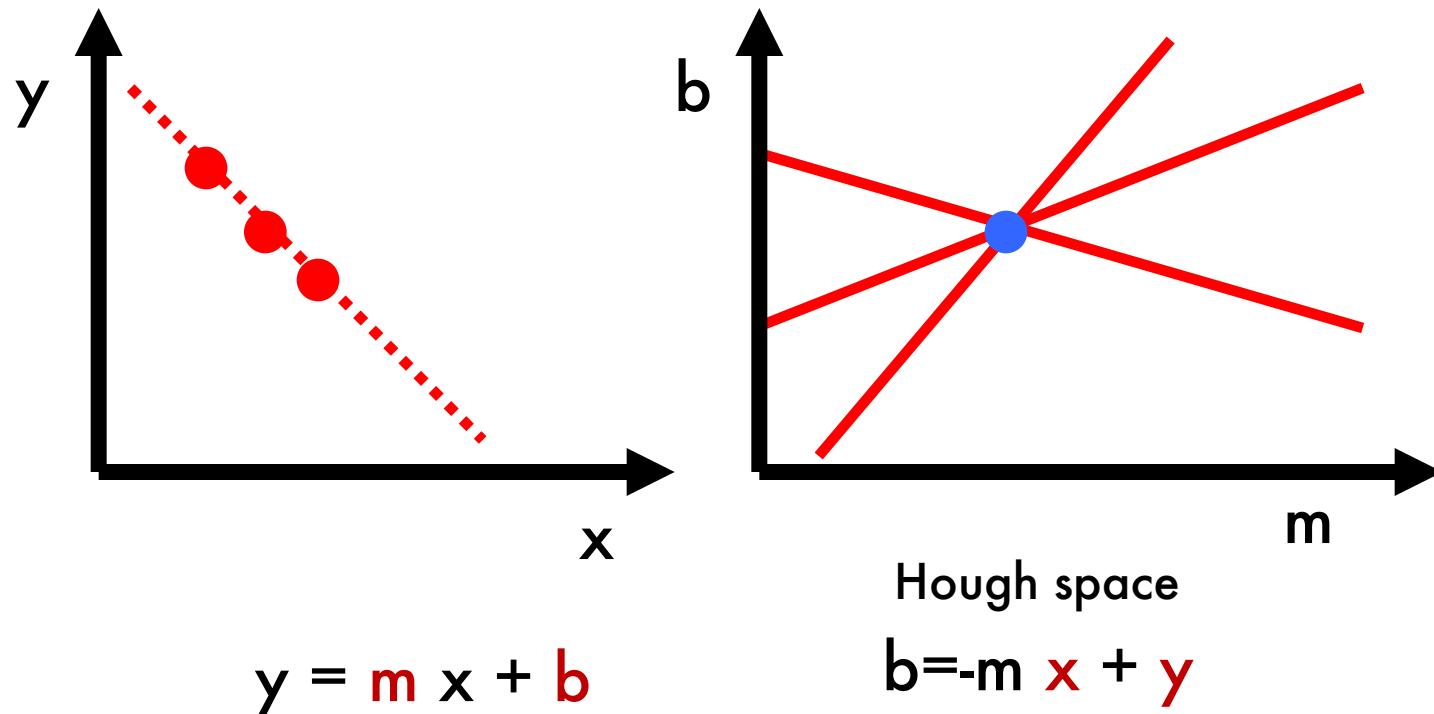
- Given a binary edge image, find the lines (or curves) that explain the data points best in the parameter space.
- This parameter space is called a Hough space



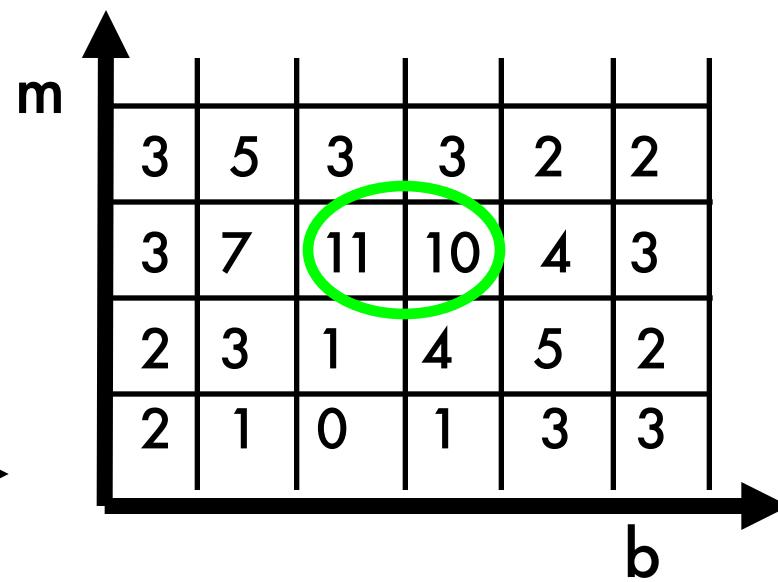
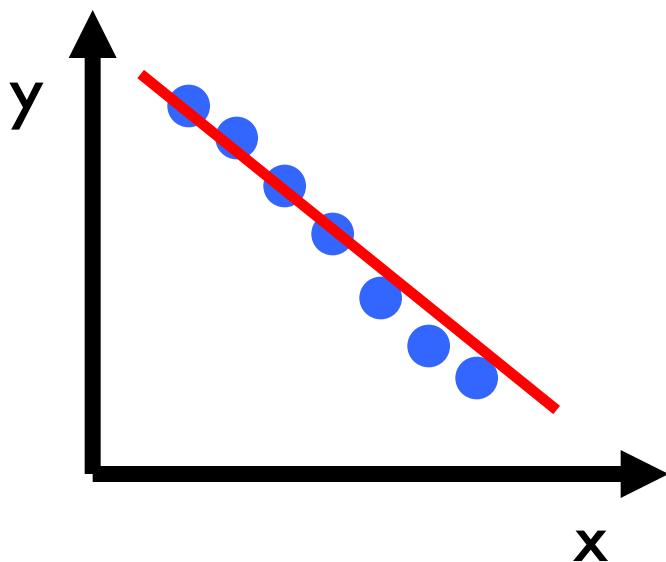
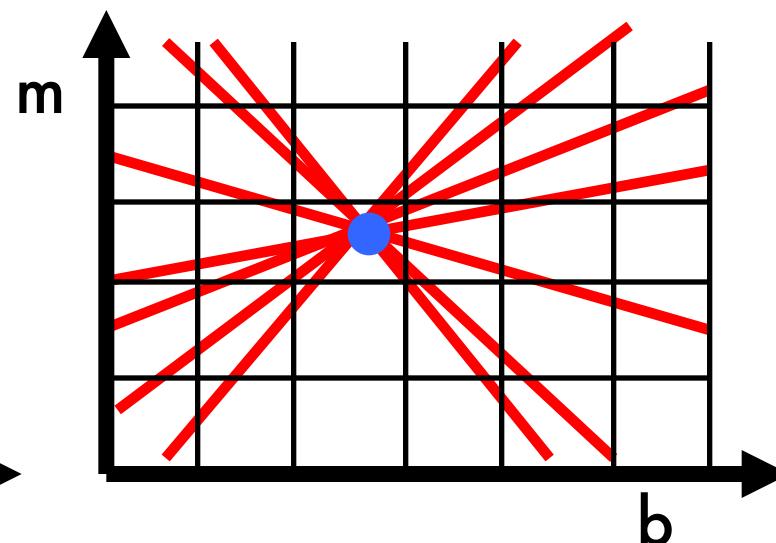
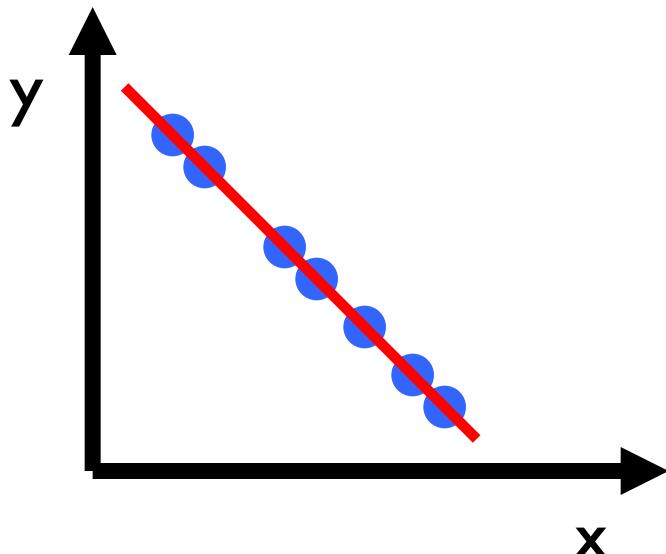
# Hough transform

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures*, Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959. (Patented).

Given a set of points, find the line (or curve) that explains the data points best



# Hough transform



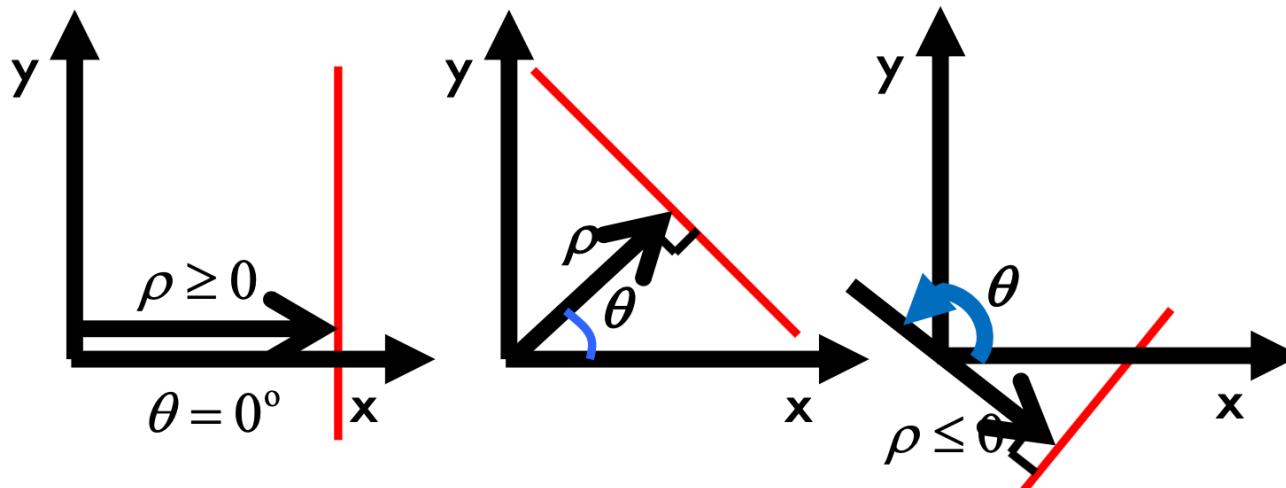
# Hough Transform

- Can detect multiple lines in an image (from multiple local maxima)
- Can be easily extended for circles and ellipses
- Computationally efficient
- Problem:  $(m, b)$  are unbounded. For example, the slope parameter  $m$  can have an infinite value.

# Hough Transform: Using Polar Form

- Use a polar representation for the parameter space

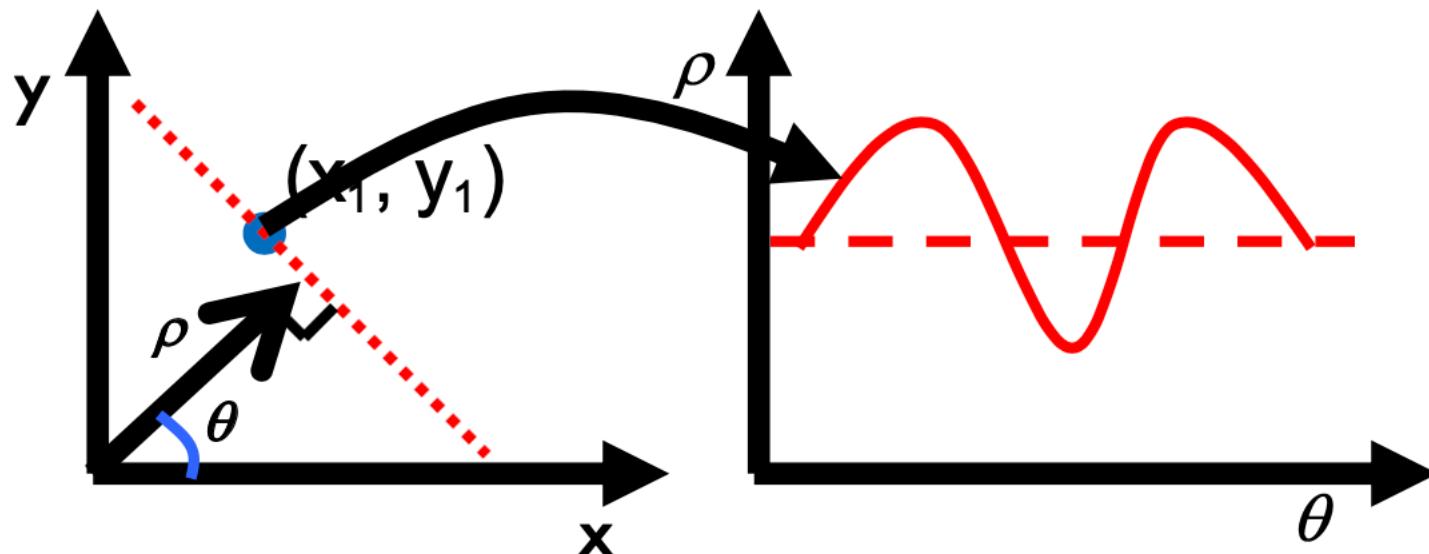
$$x \cos \theta + y \sin \theta = \rho, \quad (0 \leq \theta \leq 180)$$



A point  $(x_1, y_1)$  is mapped to a sinusoid in the polar parameter space

$$x_1 \cos \theta + y_1 \sin \theta = \rho$$

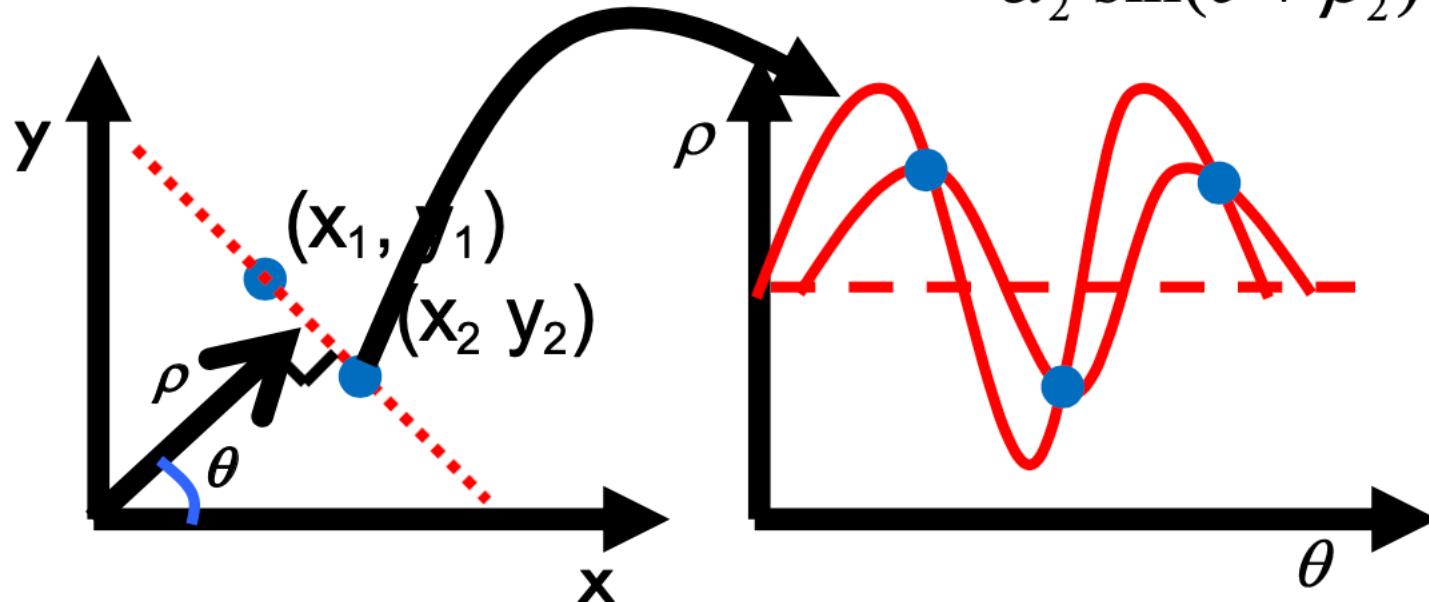
$$\begin{aligned}\rho &= x_1 \cos \theta + y_1 \sin \theta \\ &= \alpha_1 \sin(\theta + \beta_1)\end{aligned}$$



A point  $(x_2, y_2)$  is mapped to a sinusoid in the polar parameter space

$$x_2 \cos \theta + y_2 \sin \theta = \rho$$

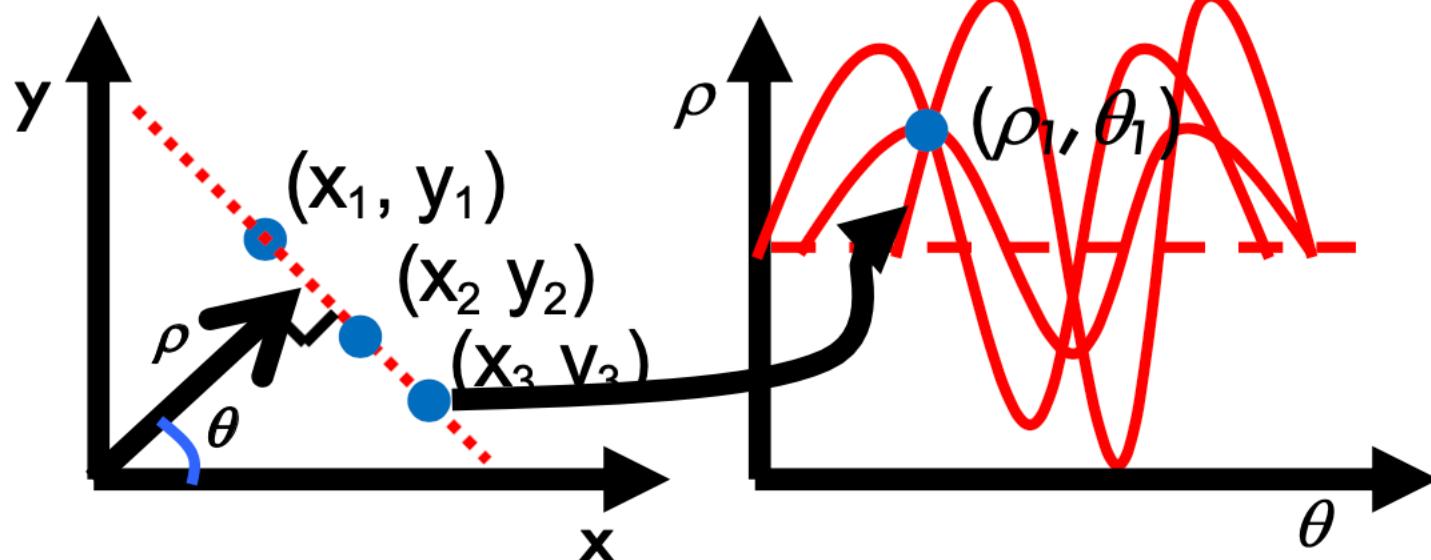
$$\begin{aligned} \rho &= x_2 \cos \theta + y_2 \sin \theta \\ &= \alpha_2 \sin(\theta + \beta_2) \end{aligned}$$



A point  $(x_3, y_3)$  is mapped to a sinusoid in the polar parameter space

$$x_3 \cos \theta + y_3 \sin \theta = \rho$$

$$\begin{aligned}\rho &= x_3 \cos \theta + y_3 \sin \theta \\ &= \alpha_3 \sin(\theta + \beta_3)\end{aligned}$$



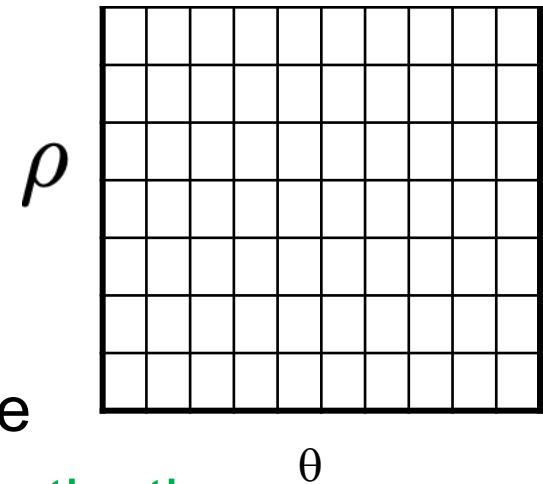
# Hough Transform algorithm:

Using the polar parameterization:

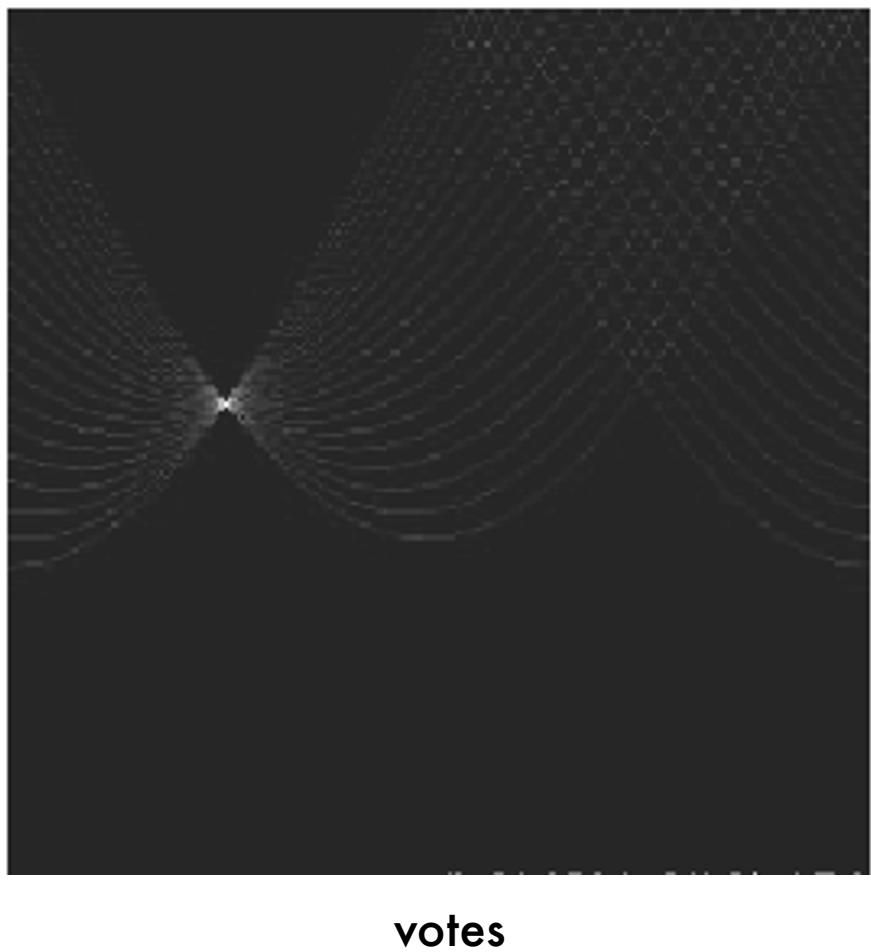
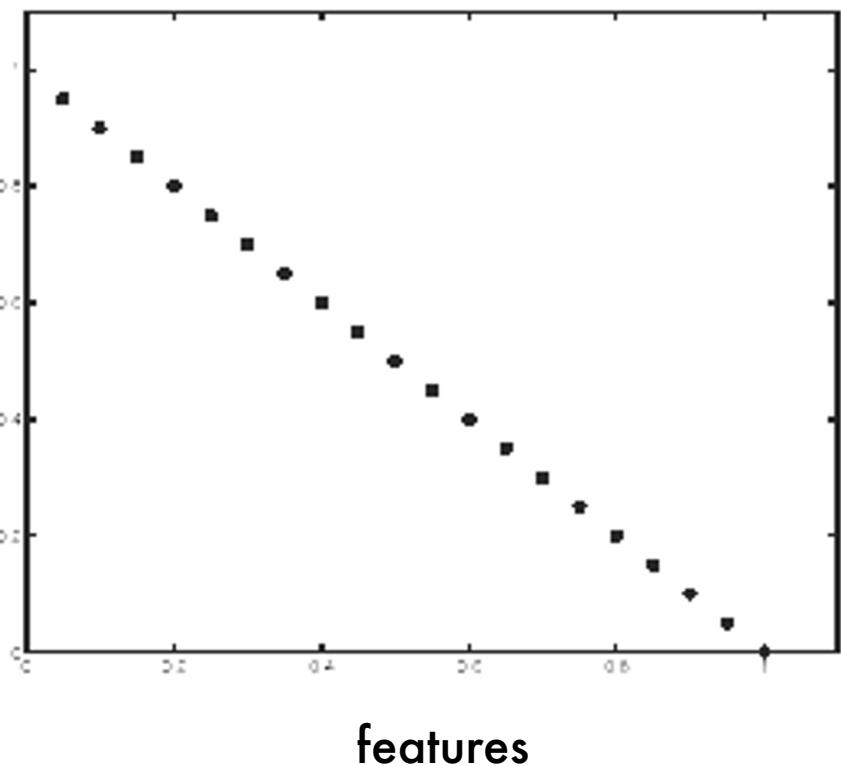
Basic Hough transform algorithm

1. Initialize  $H[\rho, \theta] = 0$
2. for each edge point  $I[x, y]$  in the image
  - for  $\theta = [\theta_{\min} \text{ to } \theta_{\max}]$  // some quantization
$$\rho = x \cos \theta + y \sin \theta$$
$$H[\rho, \theta] += 1$$
3. Find the value(s) of  $(\rho, \theta)$  where  $H[\rho, \theta]$  is maximum
4. The detected line in the image is given by
$$\rho = x \cos \theta + y \sin \theta$$

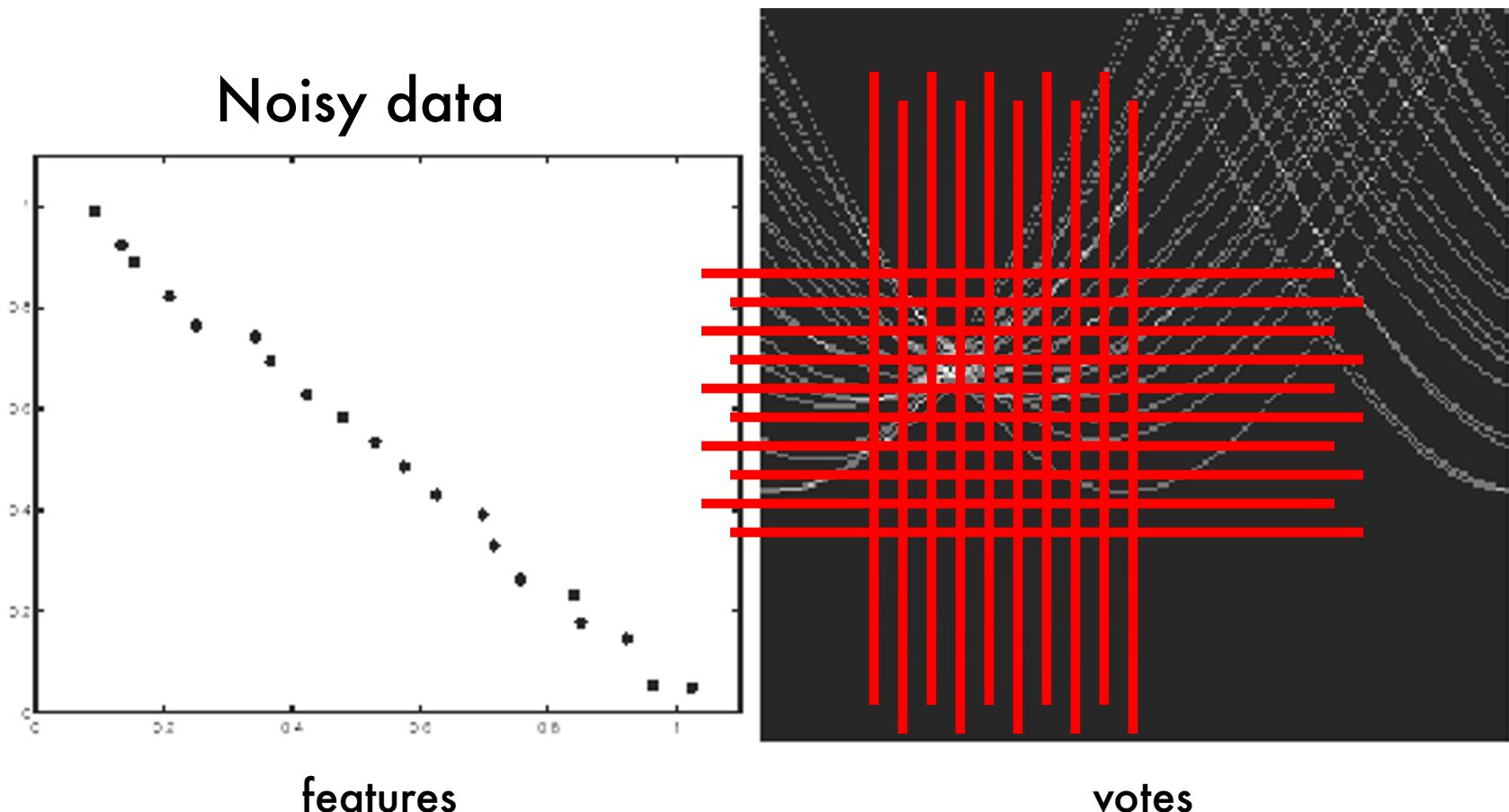
$H$ : accumulator array (votes)



# Hough transform - experiments



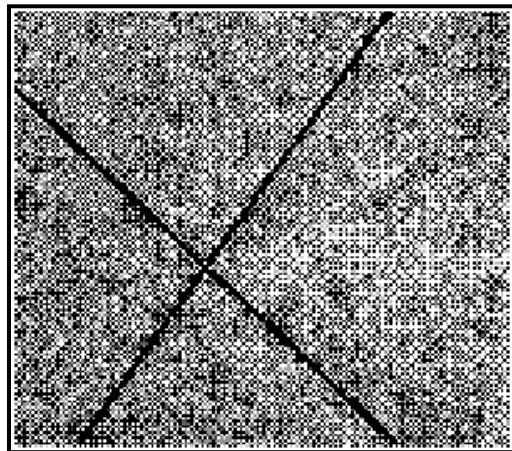
# Hough transform - experiments



Issue: Grid size needs to be adjusted...

95

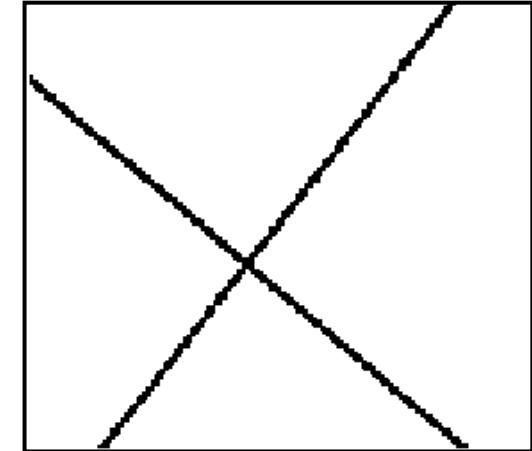
# Example: Hough Transform



Image



Edge detection

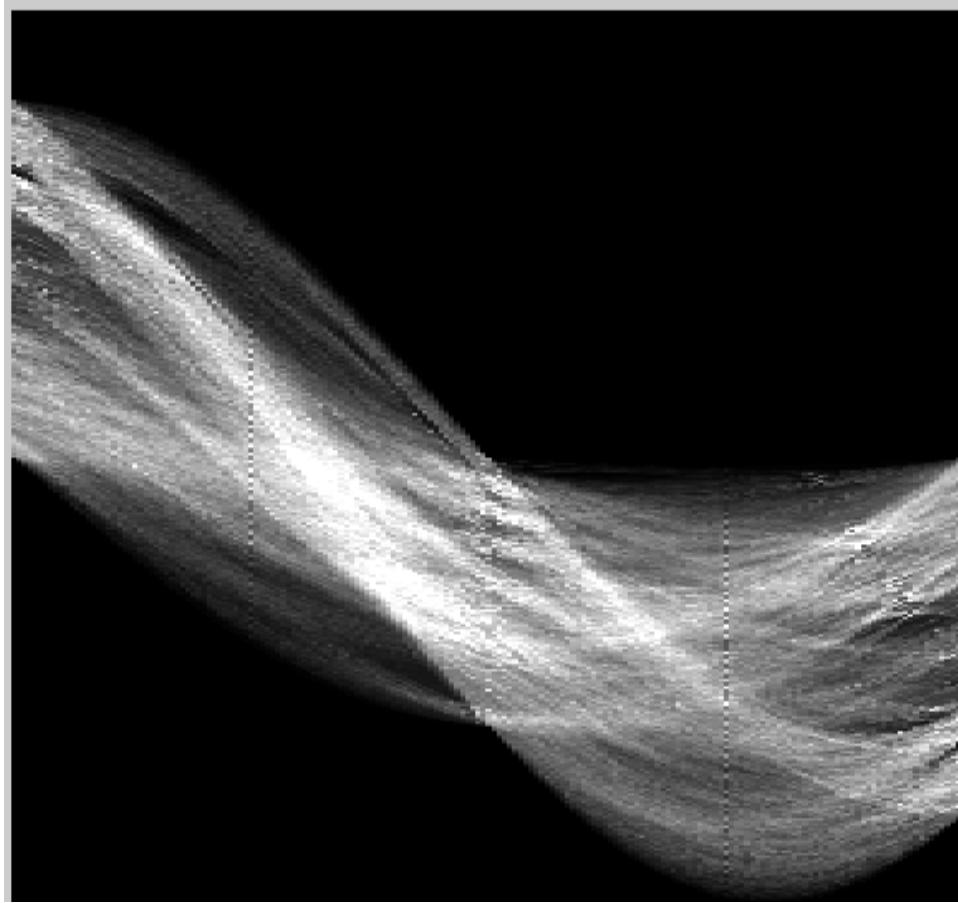


Hough Transform

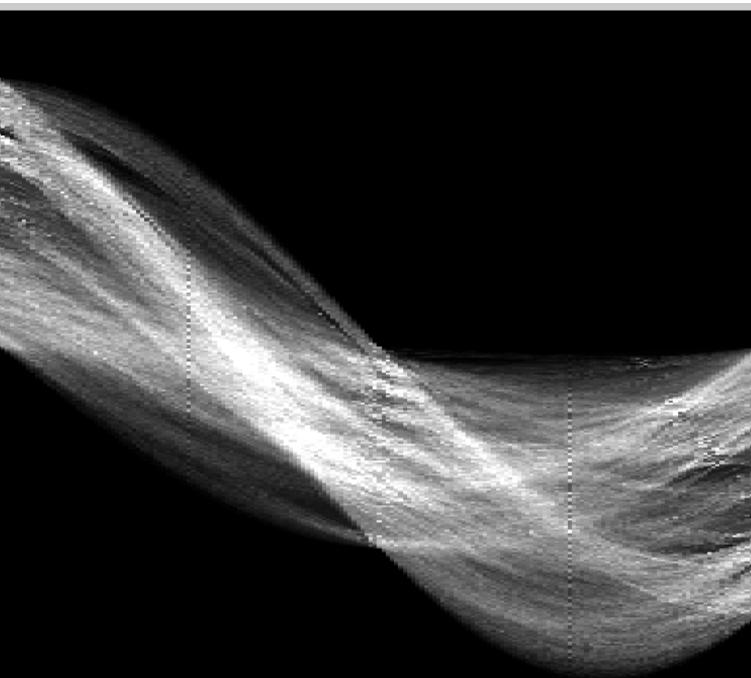
# 1. Image → Canny



## 2. Canny → Hough votes



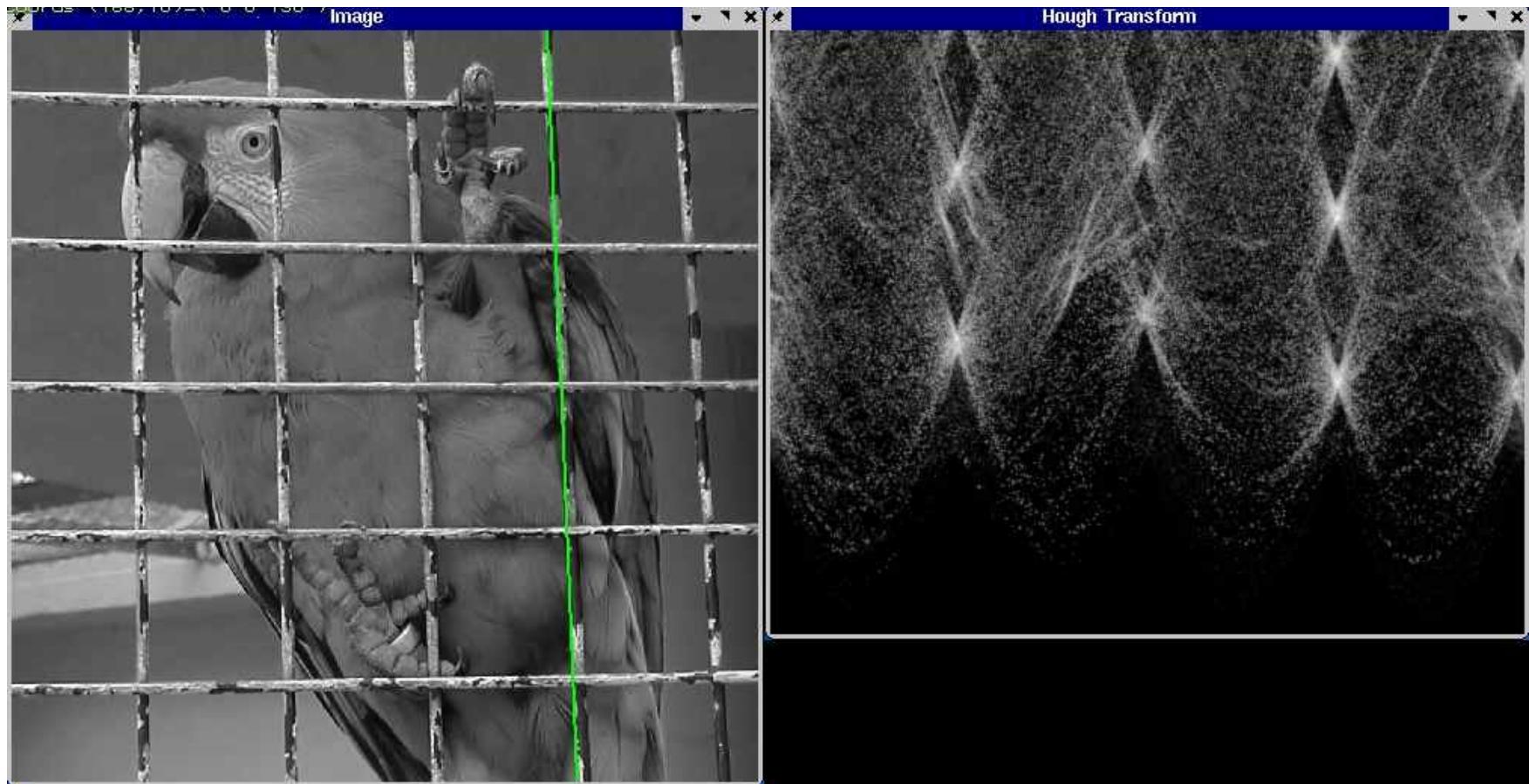
### 3. Hough votes → lines



Find peaks and post-process



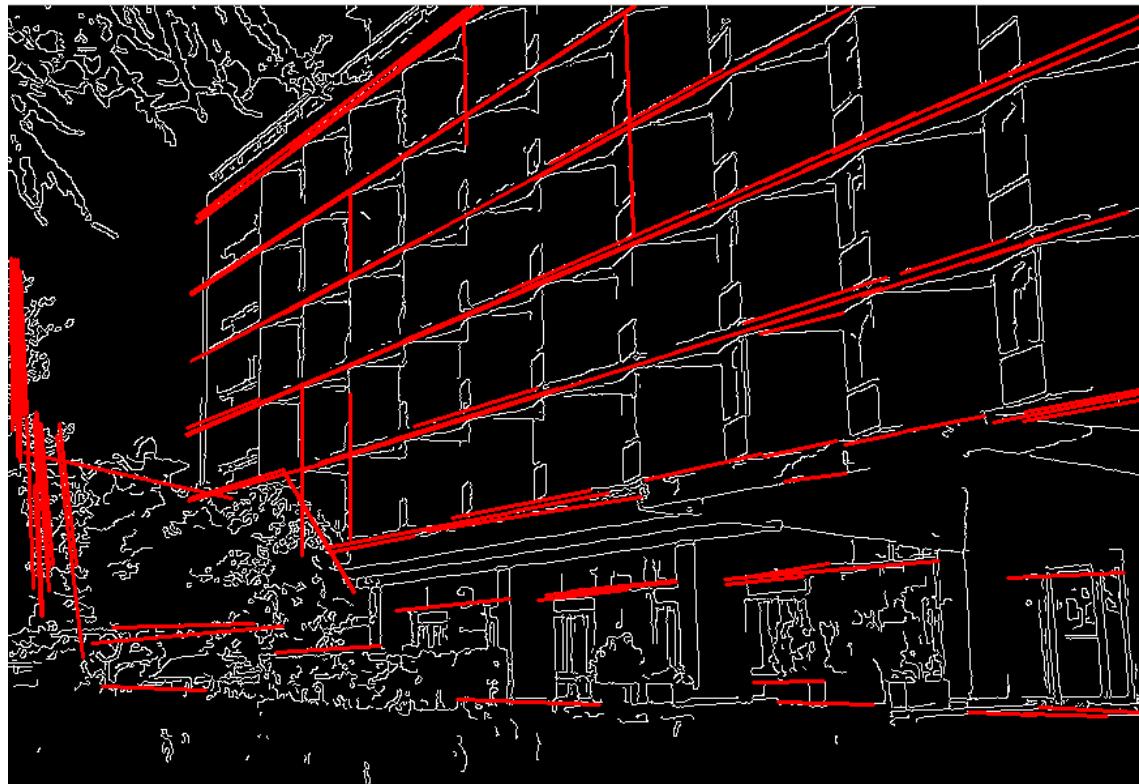
# Hough transform example



100

[http://ostatic.com/files/images/ss\\_hough.jpg](http://ostatic.com/files/images/ss_hough.jpg)

# Hough transform examples



# Hough transform pros and cons

## Pros

- All points are processed independently, so can cope with occlusion
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin
- Can detect multiple instances of a model in a single pass

## Cons

- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- Quantization: hard to pick a good grid size

# Readings

- Chapter 4.3 Lines in
- Book “Computer Vision: Algorithms and Applications”

# Image Feature Extraction

# Feature Extraction

- Edge detection/ Line Fitting
- Interest Point Detection (corner, SIFT)
- Image Region Segmentation (next week)

# Three levels of Vision Processing

- **Low-level vision:**
  - image processing, denoise, filtering, image restoration.
- **Mid-level vision:**
  - image feature detection, image segmentation, edge, contour extraction, perceptual organization,
  - 3D vision reconstruction: Multiview Geometry: 2-1/2D representation, 3D information recovery.
- **High level vision:**
  - visual recognition, classification, object localisation, semantic understanding and labelling, action, activity, event detection.

# Feature Extraction

- Edge detection/ Line Fitting---- 1D
- Interest Point Detection (corner, SIFT)-0D
- Image Region Segmentation----2D

# Content

- Harris Corner Detector
  - The most widely used corner point detector
- SIFT-Scale Invariants Feature Transform

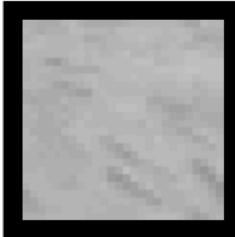
# Expected Learning Outcome

- Understand why and where we need to use interest point detector
- Learn how to implement Harris corner detector and how to use it
- Understand the principle, and why and where to use SIFT etc. (scale-invariant features)

# Motivation

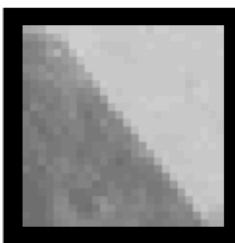
- Find “interesting” parts/pieces inside an image
  - e.g. corners, salient patches
  - Focus of attention, fixation.
  - Speed up computation.
  - Compress/extraction of information.
- Applications of interest points
  - Image Matching, Search
  - Object Detection, Object Recognition
  - Image Alignment & Stitching
  - Stereo
  - Tracking

# Interest Points



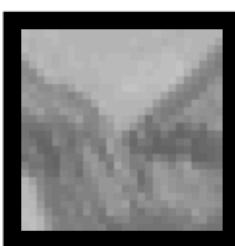
**isotropic structure:** flat region

- not interesting, 0D, not useful for matching



**linear structure:** edges, lines

- edge, can be localized in 1D, subject to the aperture problem

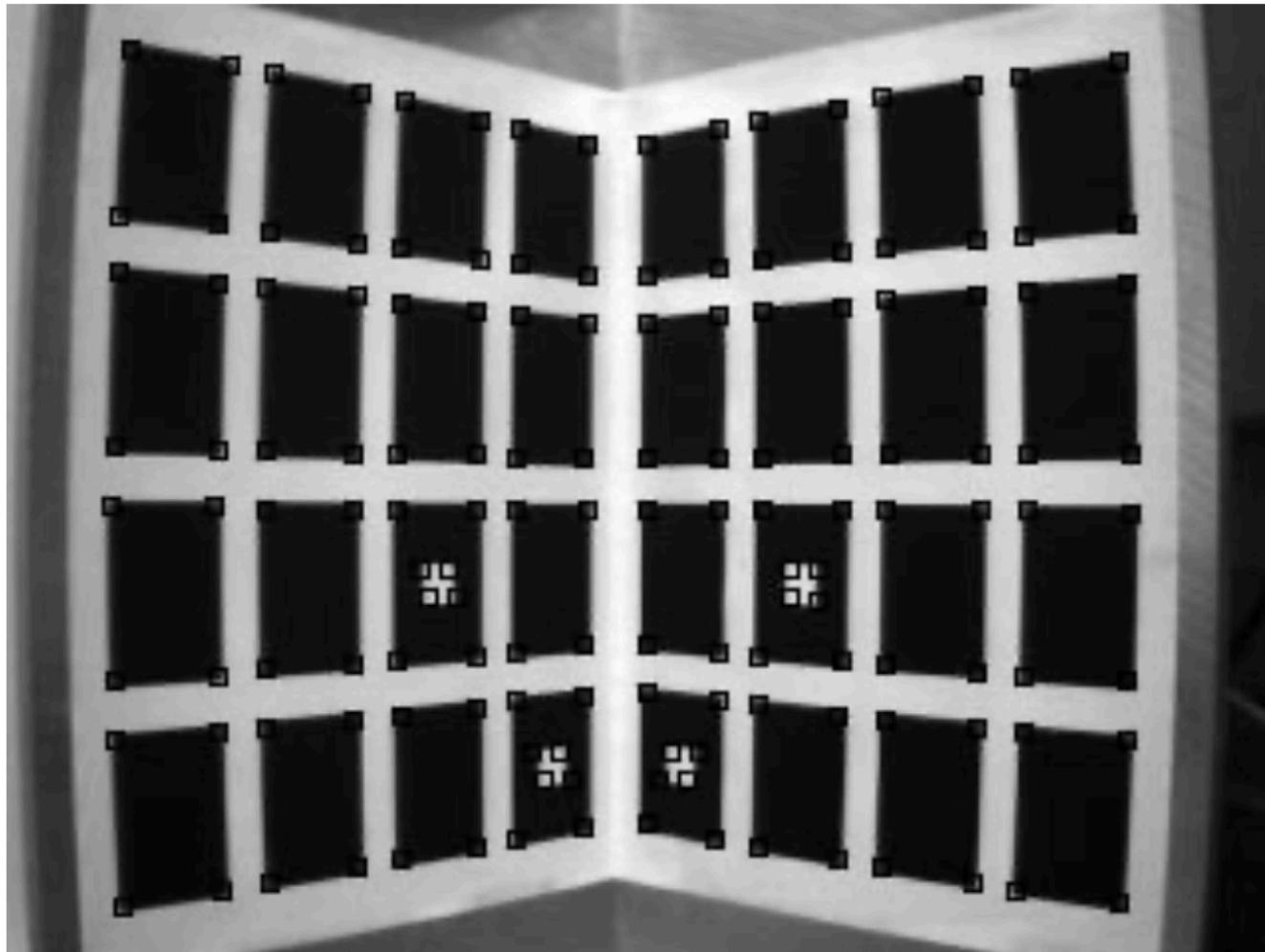


**bi-directional structure:** corners

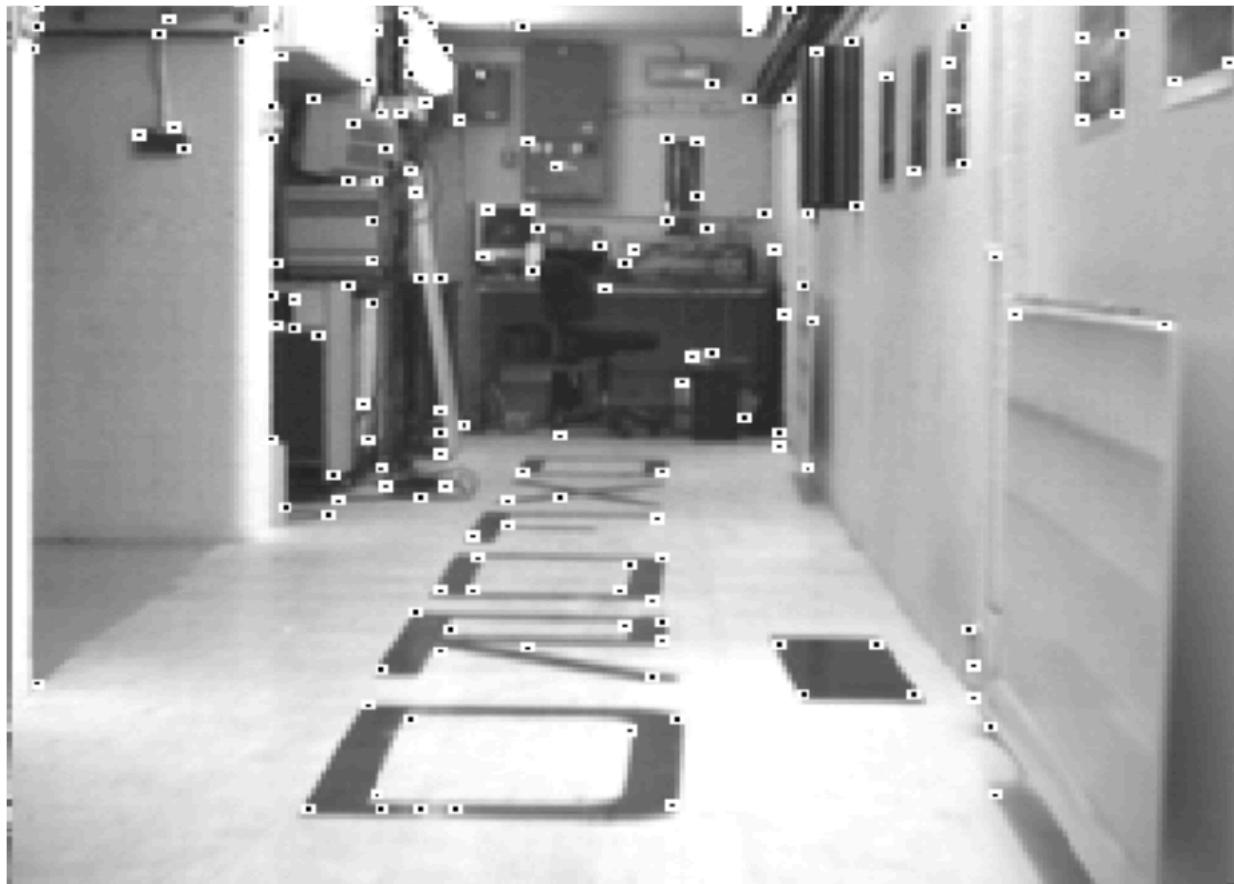
- corner, or **interest point**, can be localised in 2D, good for matching

Interest Points have 2-directional structure.

# Application: Corner Detection (for camera calibration)

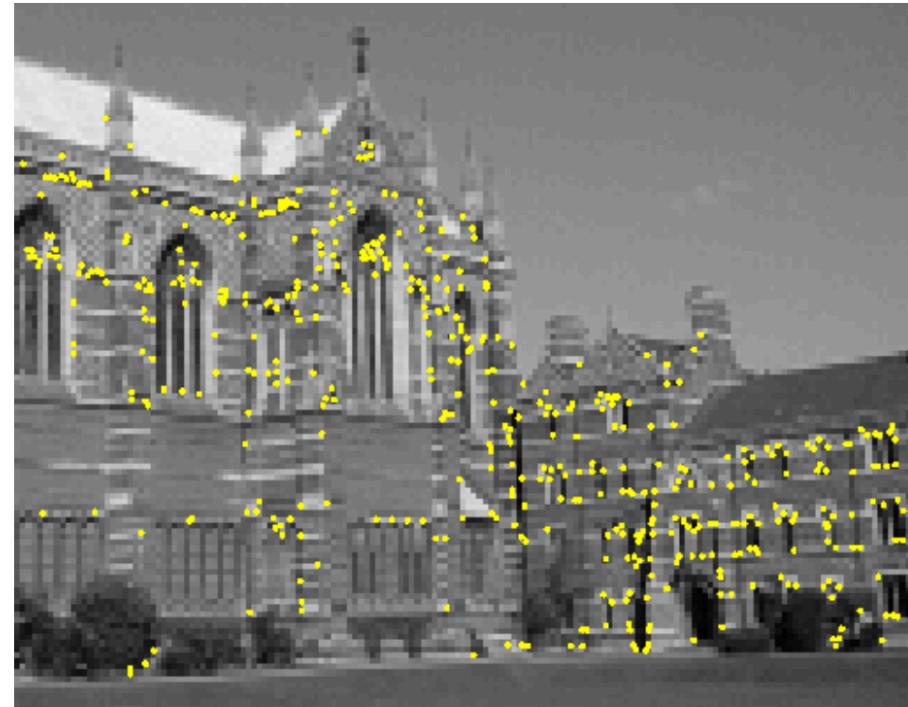
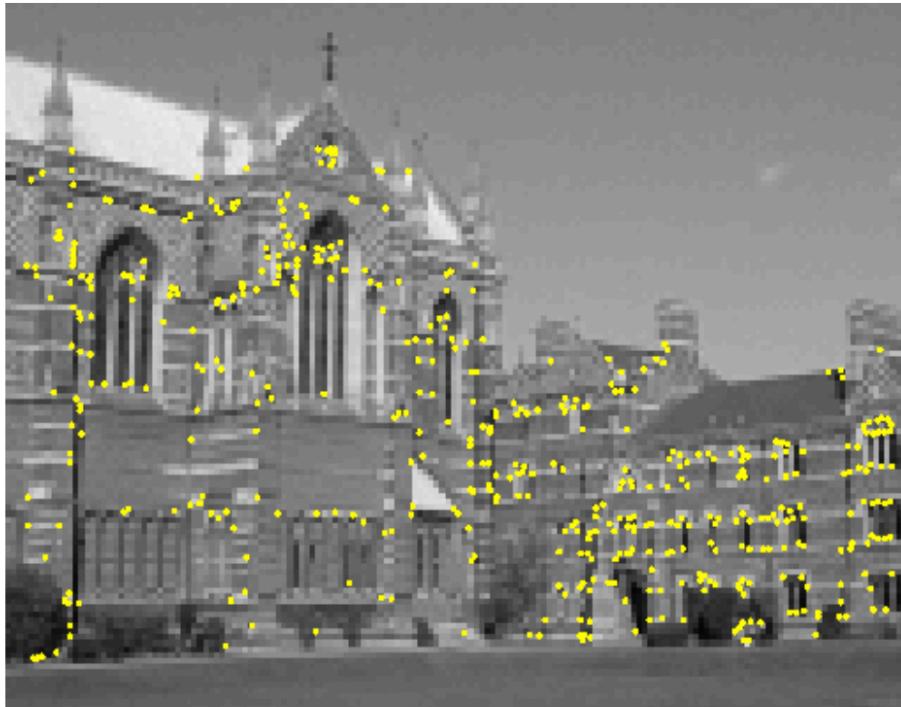


# Application: Robot navigation



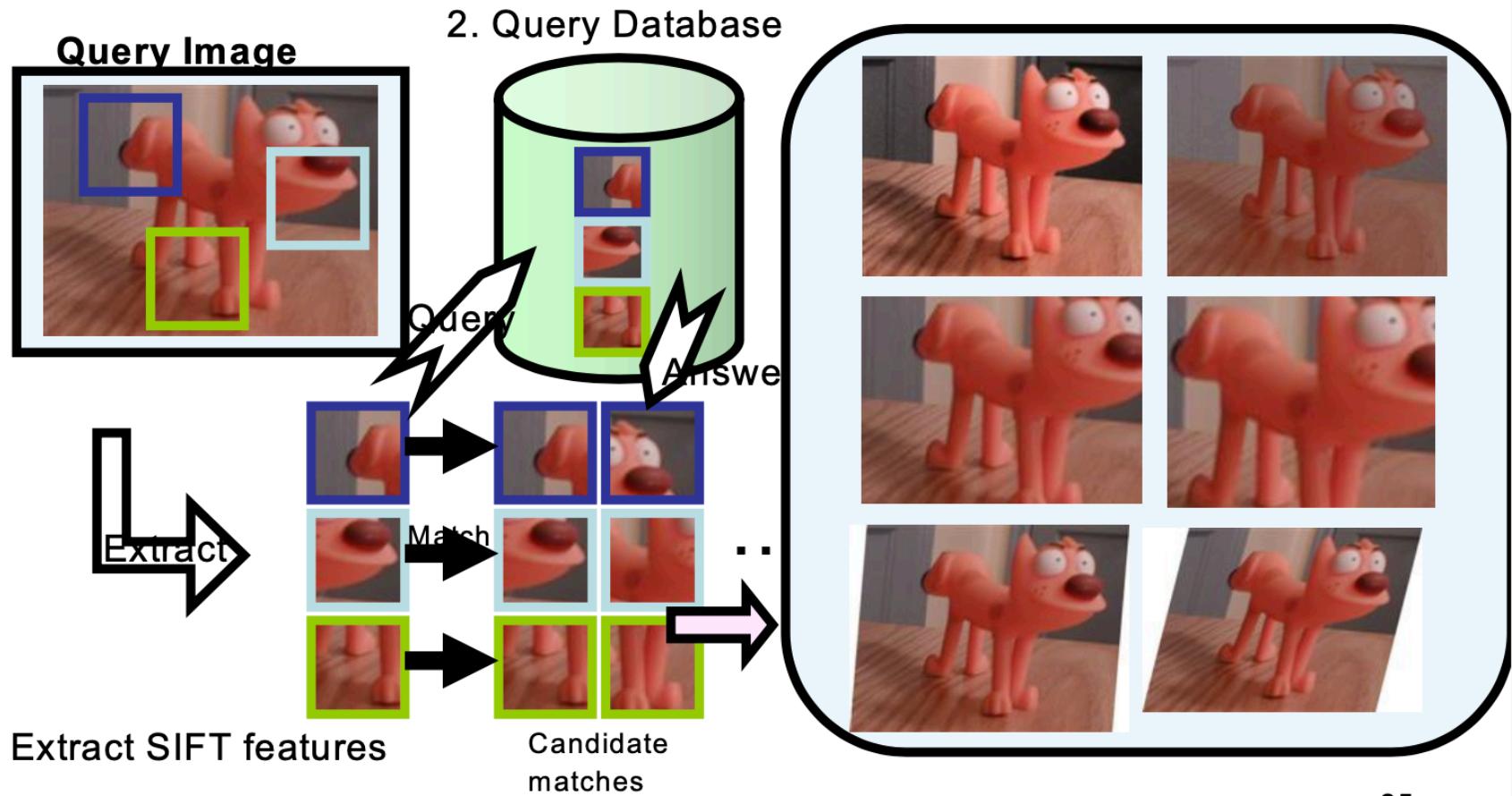
courtesy of S. Smith

# Application: Matching between two images

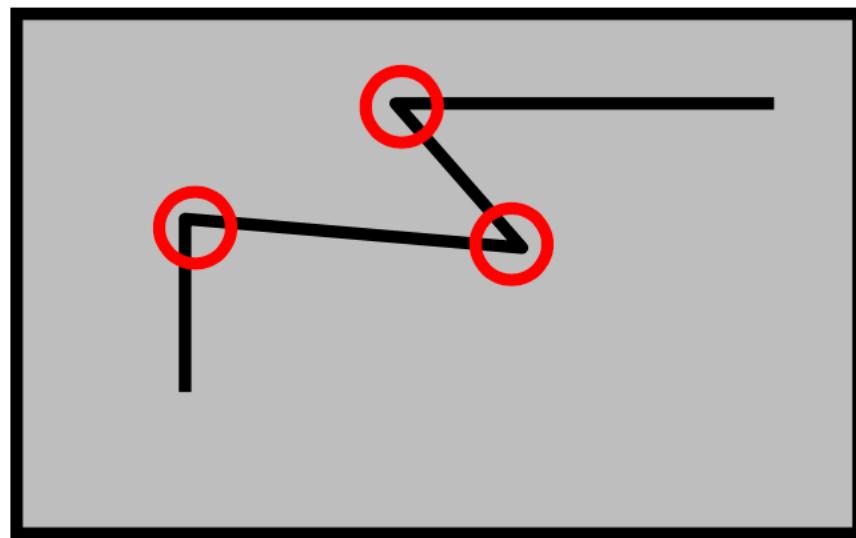


Interest points extracted with Harris (~ 500 points)

# Content Based Image Retrieval (CBIR)



# Harris Corner Detector

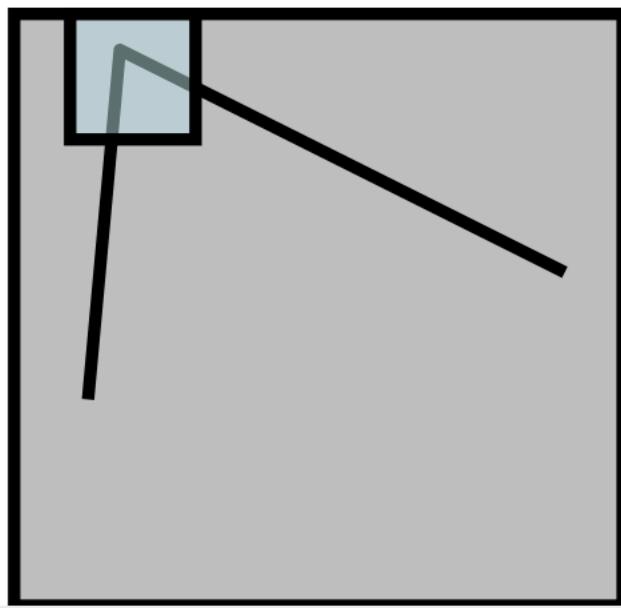


# Reference

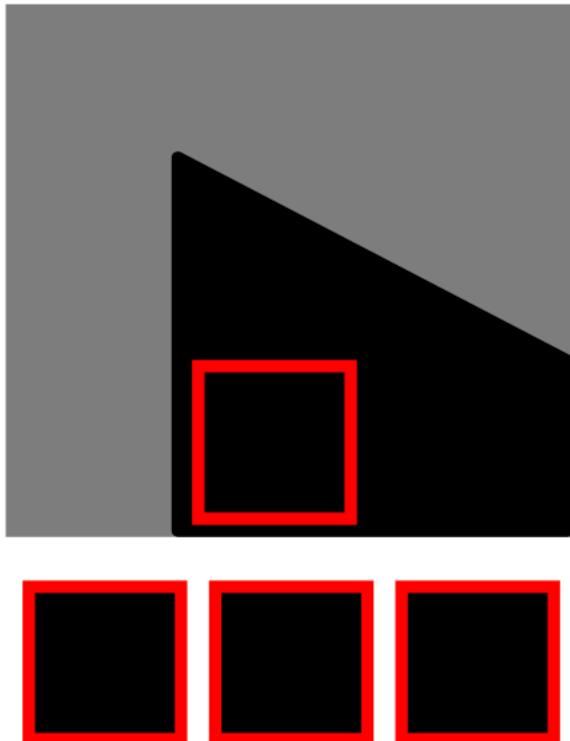
*Chris Harris & Mike Stephens, CVIU, 1988*  
**“A Combined Corner and Edge Detector”**

# Harris Corner: Intuition

- We should easily recognize a corner point by looking through a small window
- Shifting a window in *any direction* should give a *large change* in intensity

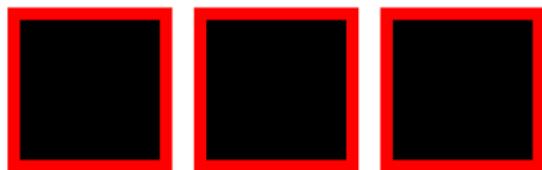
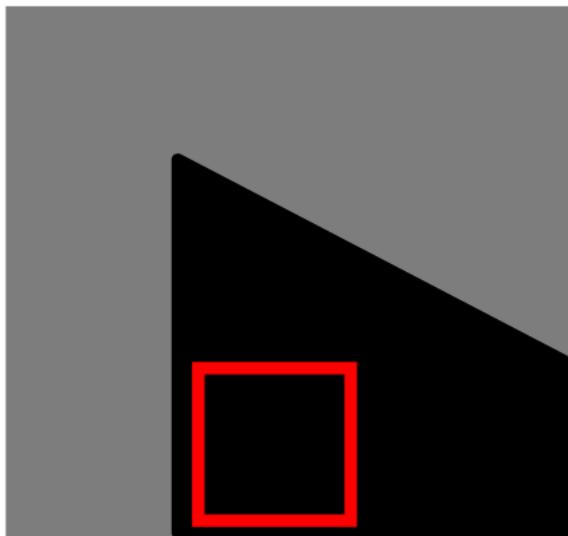


# Corner detector

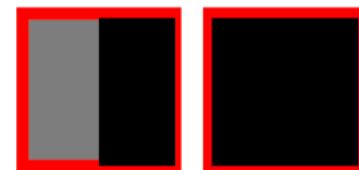
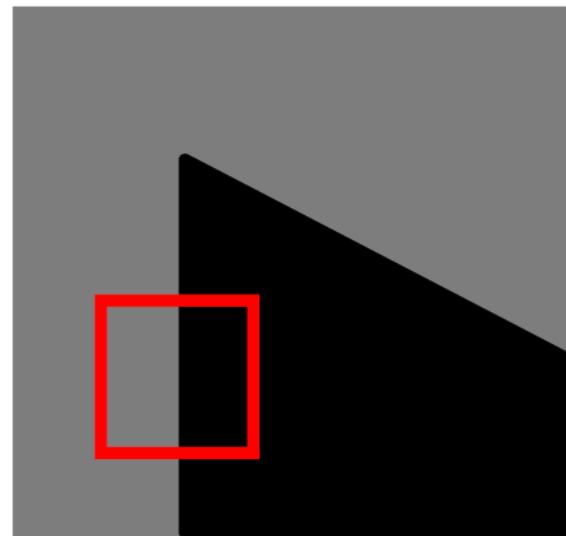


Flat

# Corner detector

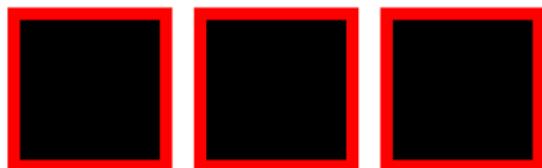
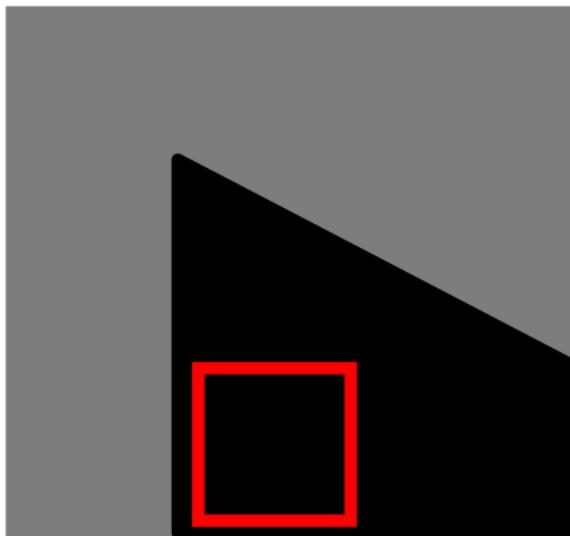


Flat

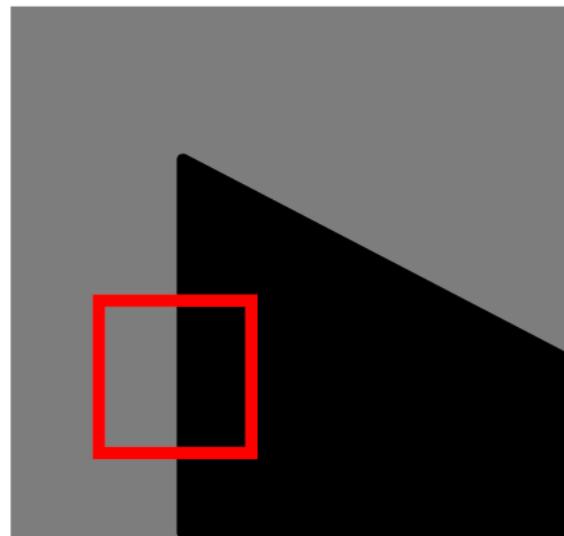


Edge

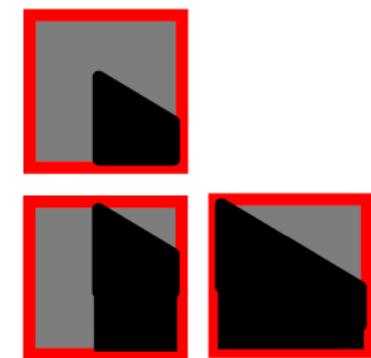
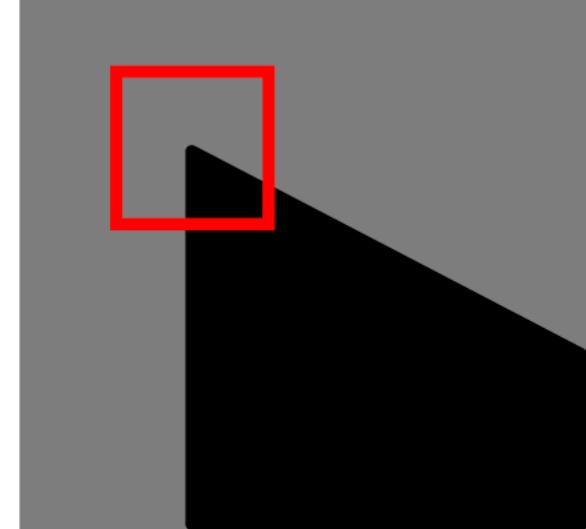
# Corner detector



Flat



Edge



Corner <sup>121</sup>

# Harris Corner: Mathematics

Change of intensity for the shift  $[u, v]$   
(local auto-correlation analysis):

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

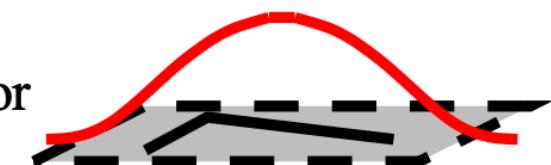
Window function      Shifted intensity      Intensity

Window function  $w(x, y) =$



1 in window, 0 outside

or



Gaussian

# Harris Corner: Mathematics

- For small shifts  $[u, v]$ , we have first-order Taylor approximation for  $I(x+u, y+v)$

$$I(x + u, y + v) \approx I(x, y) + uI_x + vI_y$$

# Harris Corner: Mathematics

$$\begin{aligned} E(u, v) &= \sum_{x,y} w(x, y)(I(x + u, y + v) - I(x, y))^2 \\ &= \sum_{x,y} w(x, y)(I(x, y) + uI_x + vI_y - I(x, y))^2 \\ &= \sum_{x,y} w(x, y)(uI_x + vI_y)^2 \\ &= \sum_{x,y} w(x, y)(u^2I_x^2 + v^2I_y^2 + 2uvI_xI_y) \\ &= [u, v] M \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

# Harris Corner: Mathematics

- For small shift  $[u, v]$ , we have 1<sup>st</sup> order Tylor approximation

$$E(u, v) \approx [u, v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

- Where  $M$  is a  $2 \times 2$  matrix computed from image derivatives:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

125

$M$  is called the second order moments (or auto correlation ) matrix of gradients.

# Harris Corner: Mathematics

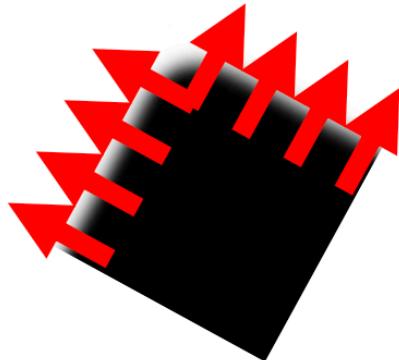
Intensity change in shifting window: eigenvalue analysis

$$E(u, v) \cong [u, v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad \lambda_1, \lambda_2 - \text{eigenvalues of } M$$

If we try every possible orientation  $\mathbf{n}$ ,  
the biggest change and smallest changes in intensity happen in  $\lambda_s$ .

# Recall: eigenvalue decomposition in linear algebra

Since  $M$  is symmetric, we have



$$M = X \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} X^T$$

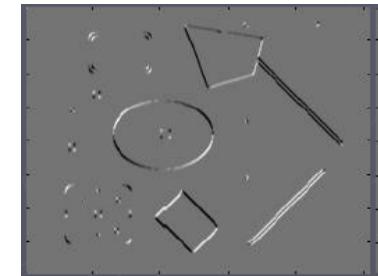
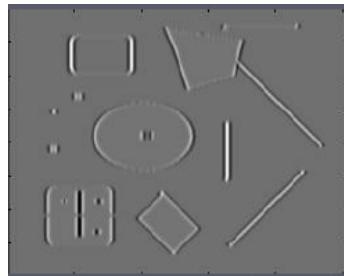
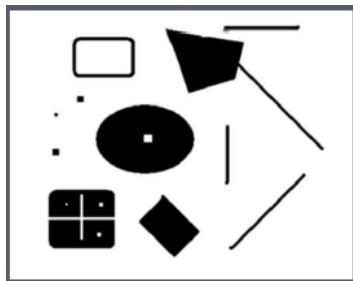
$$Mx_i = \lambda_i x_i$$

The *eigenvalues* of  $M$  reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.

# Corners as distinctive interest points

$$M = \sum w(x, y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

2 x 2 matrix of image derivatives (averaged in neighborhood of a point).



Notation:

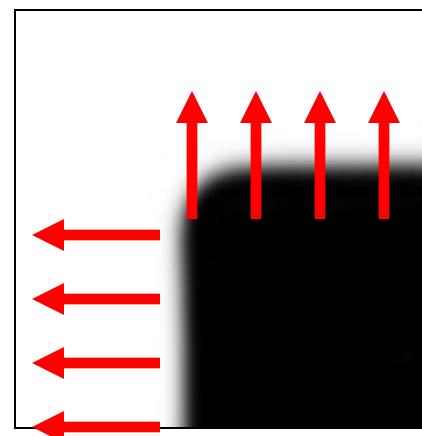
$$I_x \Leftrightarrow \frac{\partial I}{\partial x}$$

$$I_y \Leftrightarrow \frac{\partial I}{\partial y}$$

$$I_x I_y \Leftrightarrow \frac{\partial I}{\partial x} \frac{\partial I}{\partial y}$$

# What does this matrix reveal?

First, consider an axis-aligned corner:



# What does this matrix reveal?

First, consider an axis-aligned corner:

$$M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

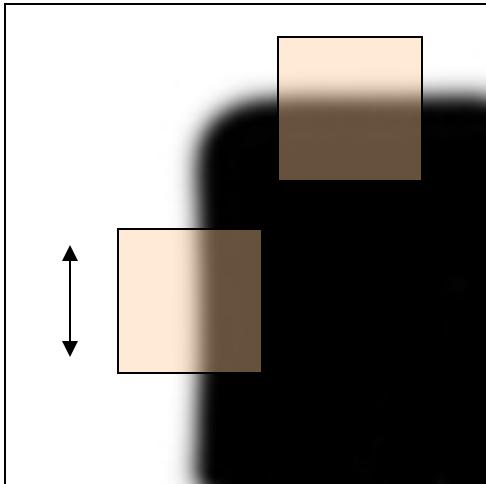
This means dominant gradient directions align with x or y axis

Look for locations where **both**  $\lambda$ 's are large.

If either  $\lambda$  is close to 0, then this is **not** corner-like.

What if we have a corner that is not aligned with the image axes?

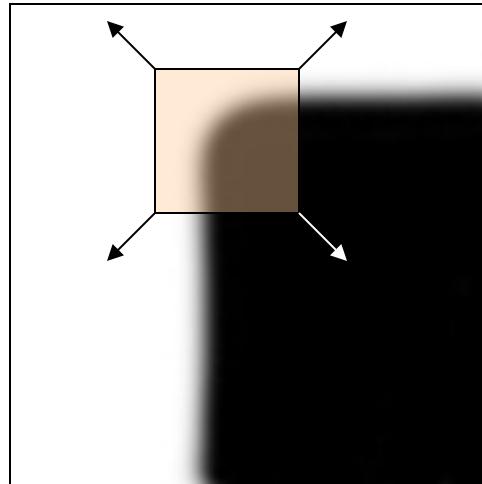
# Corner response function



“edge”:

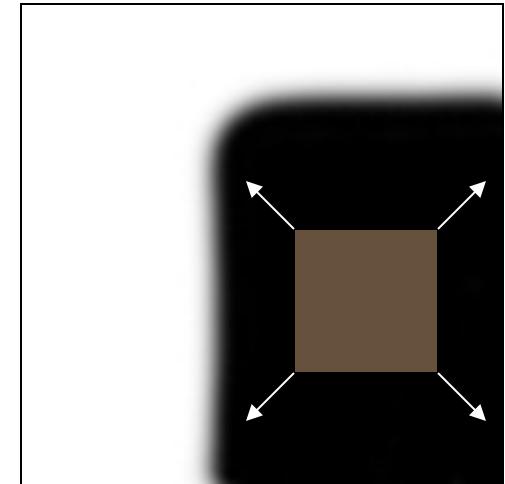
$$\lambda_1 \gg \lambda_2$$

$$\lambda_2 \gg \lambda_1$$



“corner”:

$\lambda_1$  and  $\lambda_2$  are large,  
 $\lambda_1 \sim \lambda_2$ ;



“flat” region

$\lambda_1$  and  $\lambda_2$  are  
small;

# Harris corner detector

- 1) Compute  $M$  matrix for each image window to get their *cornerness* scores.
- 2) Find points whose surrounding window gave large corner response ( $R >$  threshold)
- 3) Take the points of local maxima, i.e., perform non-maximum suppression

# Harris Corner Detector

Classification of image points using eigenvalues of  $M$ :

$\lambda_1$  and  $\lambda_2$  are small;  
 $E$  is almost constant  
in all directions

$\lambda_2$

“Edge”

$\lambda_2 \gg \lambda_1$

“Corner”

$\lambda_1$  and  $\lambda_2$  are both large,  
 $\lambda_1 \sim \lambda_2$ ;

$E$  increases in all  
directions



“Flat”  
region

“Edge”

$\lambda_1 \gg \lambda_2$

# Harris Corner Detector: Mathematics

Measure of corner response: (Cornerness)

$$R = \det M - k (\operatorname{trace} M)^2$$

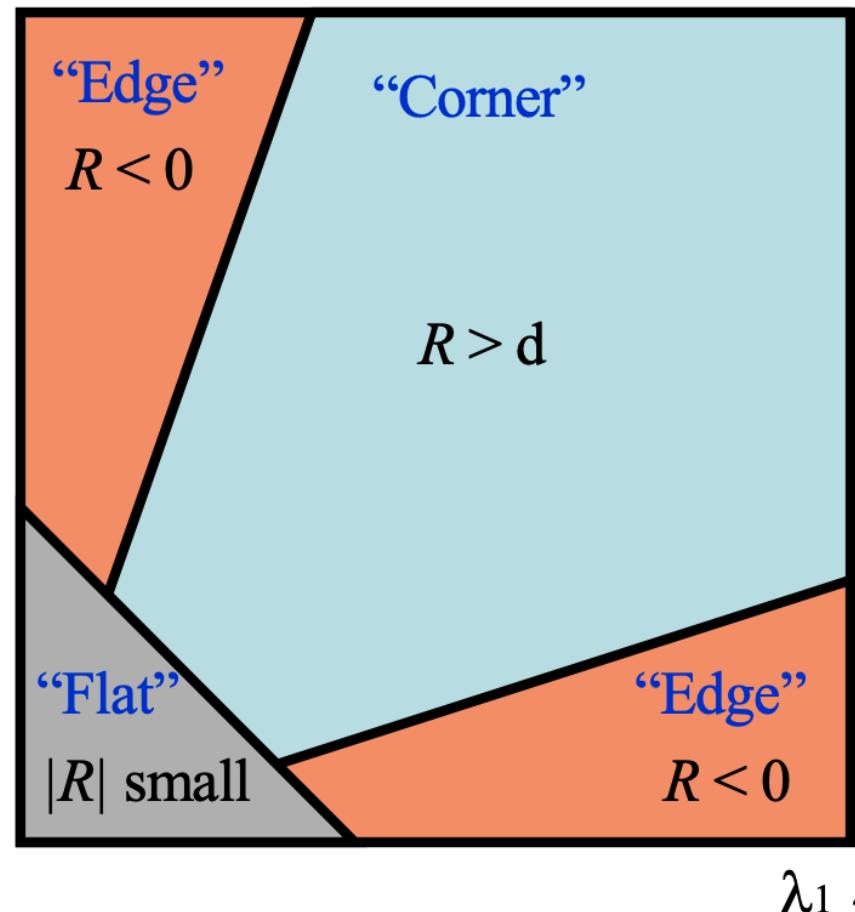
$$\det M = \lambda_1 \lambda_2$$

$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

( $k$  – empirical constant,  $k = 0.01\text{-}0.1$ )

# Harris Corner Detector: Mathematics

- $R$  depends only on eigenvalues of  $M$
- $R$  is large for a corner
- $R$  is negative with large magnitude for an edge
- $|R|$  is small for a flat region

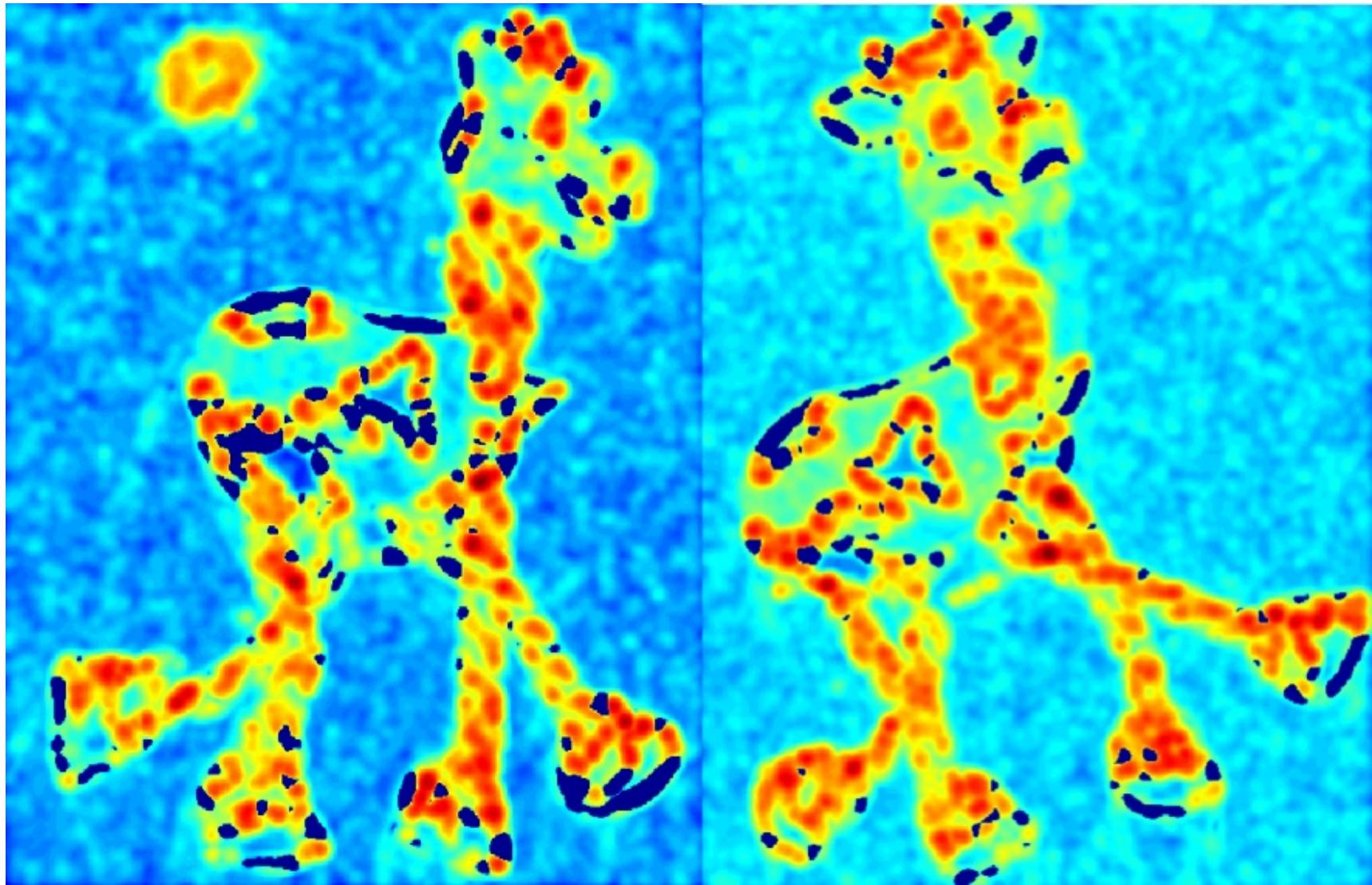


# Harris Detector: Steps



# Harris Detector: Steps

Compute corner response  $R$  (*i.e.* the “*cornerness*”)



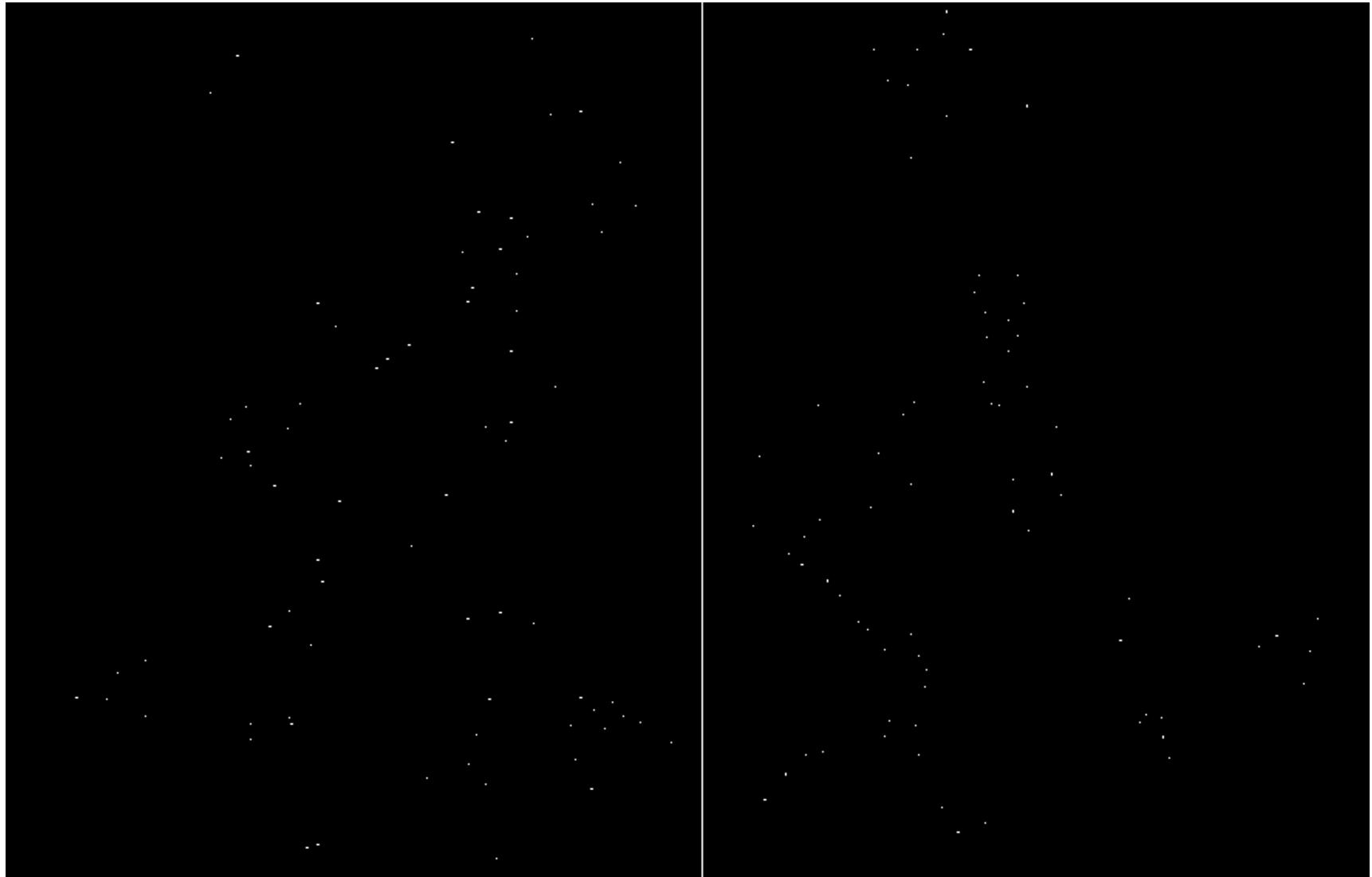
# Harris Detector: Steps

Find points with large corner response:  $R >$  threshold

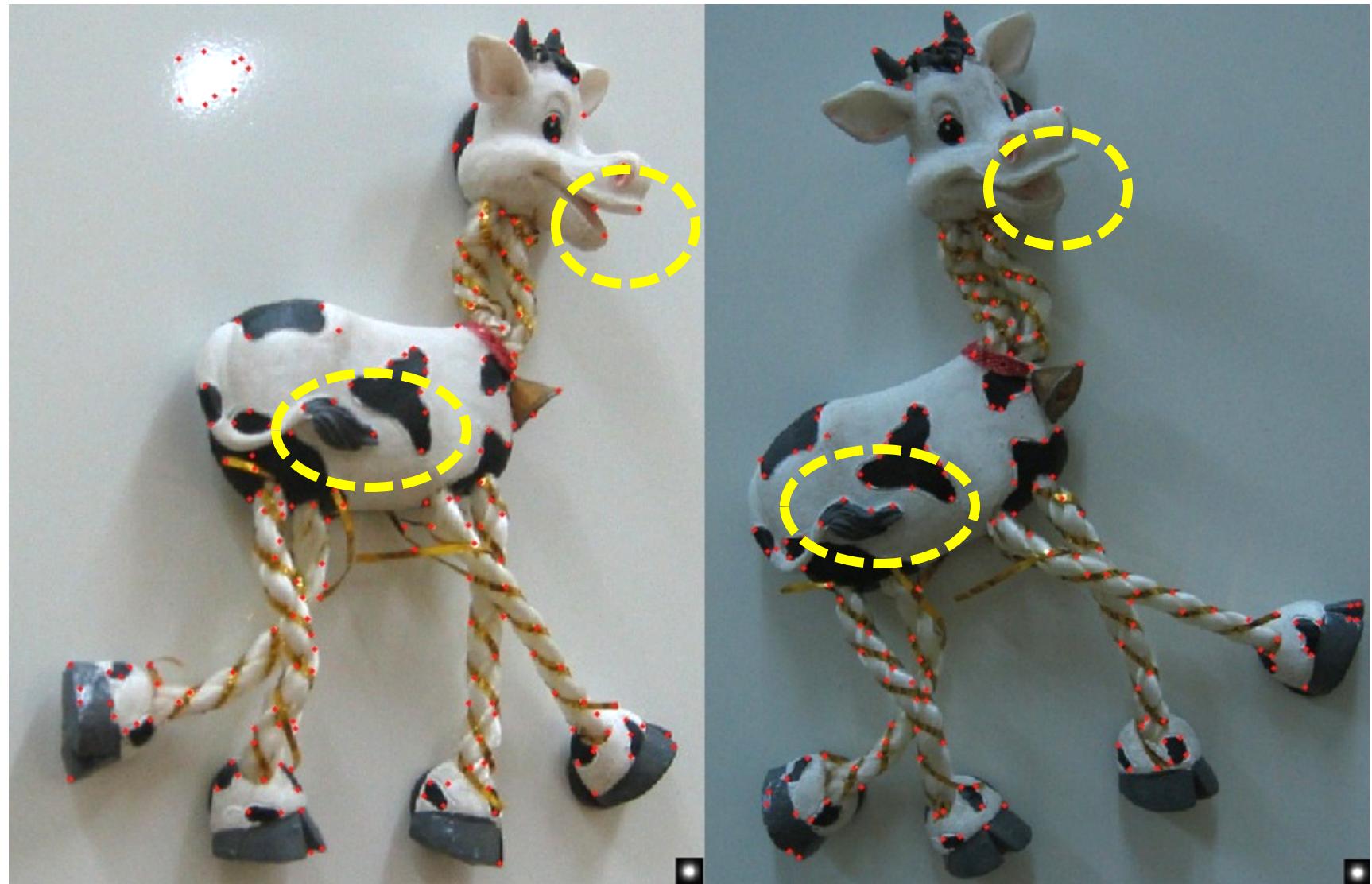


# Harris Detector: Steps

Take only the points of local maxima of  $R$



# Harris Detector: Steps



# Recap: Harris Corner

- Compute the moment (auto-correlation) matrix M
  - captures the structure of the local neighborhood
  - measure based on **eigenvalues** of M
    - 2 strong eigenvalues => interest point
    - 1 strong eigenvalue => edge or contour
    - 0 or very weak eigenvalues => flat region
- Interest point detection
  - threshold on the eigenvalues
  - local maximum for localization

# Recap: Harris Corner

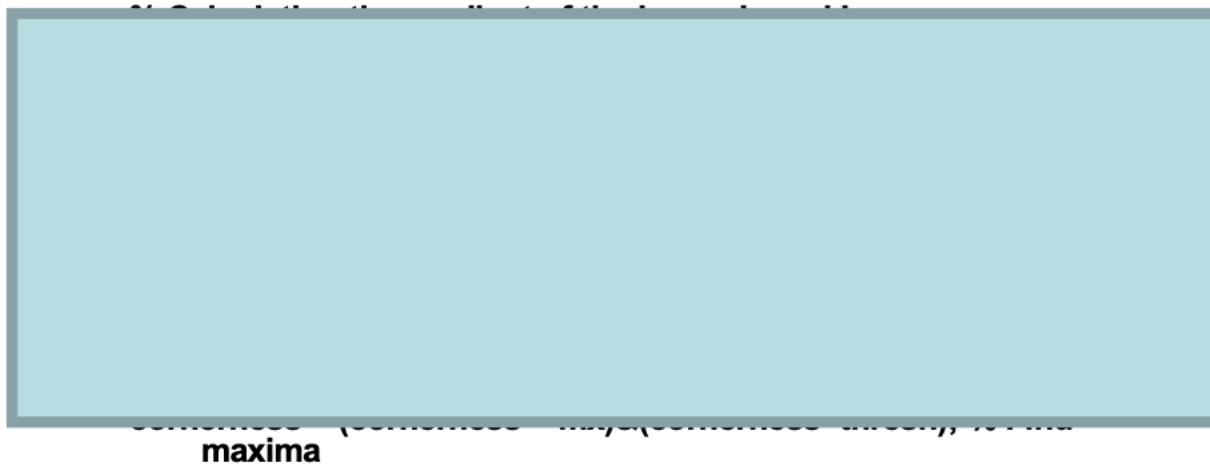
- Corner strength  $R = \det(M) - k \operatorname{Tr}(M)^2$
- Let  $\alpha$  and  $\beta$  be the two eigenvalues. **We don't have to calculate them!** Instead, use trace and determinant:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21} \\ \operatorname{tr}(\mathbf{A}) = a_{11} + a_{22}$$

- $R$  is positive for corners, negative for edges, and small for flat regions
- **Non-maximal suppression:** select corners that are 8-way local maxima

# Harris Corner in Matlab Code (incomplete)

```
% Harris Corner detector  
  
sigma=2; thresh=0.1; sze=11; disp=0;  
  
% Derivative masks  
dy = [-1 0 1; -1 0 1; -1 0 1];  
dx = dy'; %dx is he transpose matrix of dy
```



```
[rws,cols] = find(cornerness); % Find row,col coords. clf ;  
imshow(bw);  
hold on;  
p=[cols rws];  
plot(p(:,1),p(:,2),'or');  
title('bf Harris Corners')
```

In your C-Lab-2: