



Image and Vision Computing Fitting

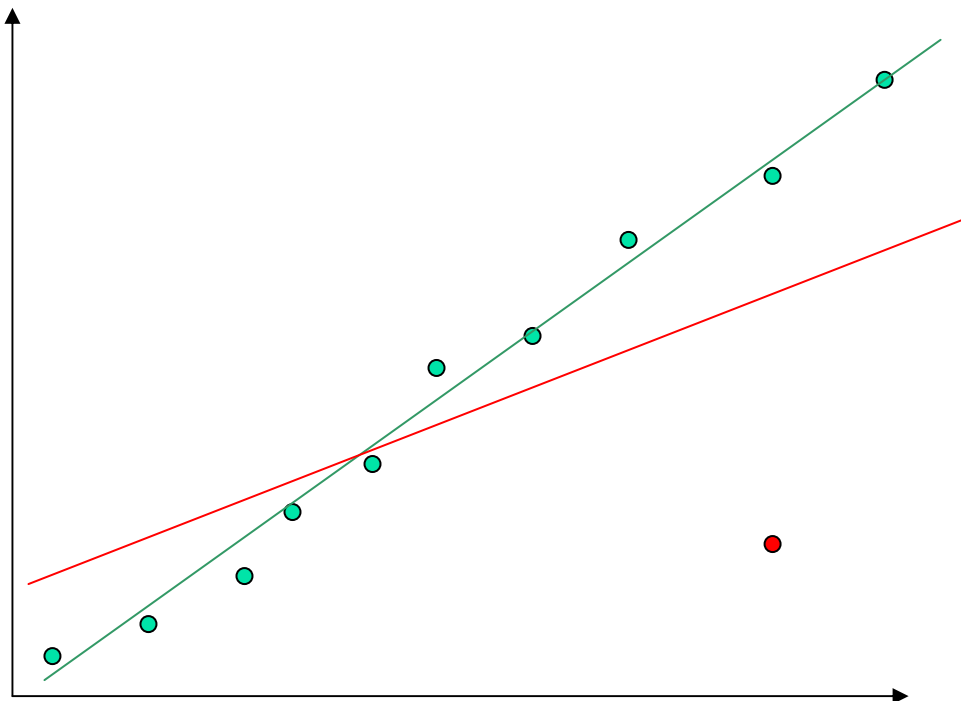
Instructor: Chen Yisong
HCI & Multimedia Lab, Peking University



Contents

- Curve Fitting(曲线拟合)
- Hough Transform(霍夫变换)
- Robust fitting

Line Fitting(直线拟合)



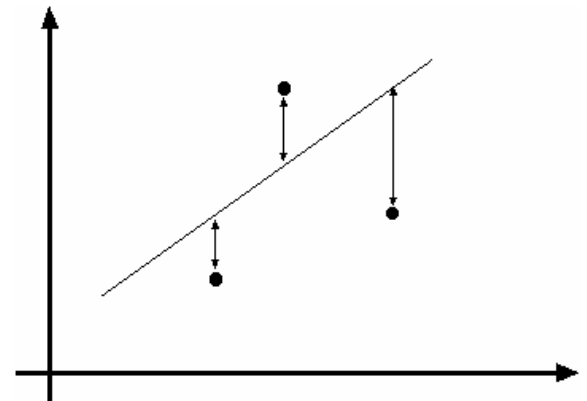
Least Squares Fit(最小二乘拟合)

- Standard linear solution to estimating unknowns.
 - If we know which points belong to which line
 - Or if there is only one line

$$y = ax + b = f(x, a, b)$$

$$\text{Minimize } E = \sum_i [y_i - f(x_i, a, b)]^2$$

Take derivative wrt a and b set to 0





Line Fitting

$$y = ax + b$$

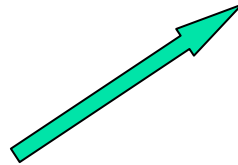


$$y_1 = ax_1 + b$$

$$y_2 = ax_2 + b$$

$$\vdots$$

$$y_n = ax_n + b$$



$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_B = \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & 1 \\ x_n & 1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_C \Rightarrow B = AC$$



$$A^T B = A^T A C$$

$$(A^T A)^{-1} A^T B = (A^T A)^{-1} (A^T A) C$$

$$C = (A^T A)^{-1} A^T B$$

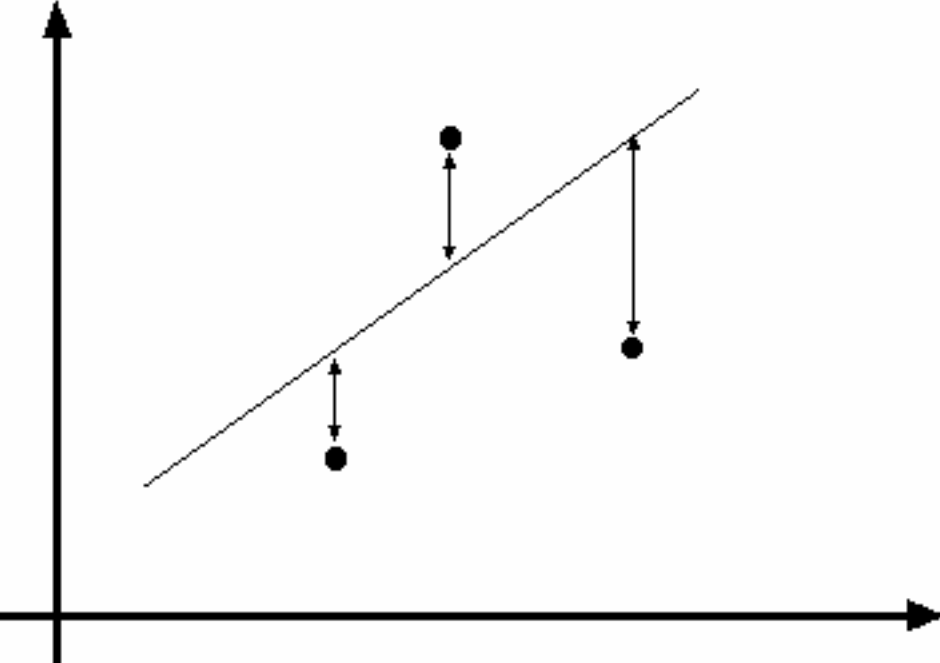


Least Squares Fit

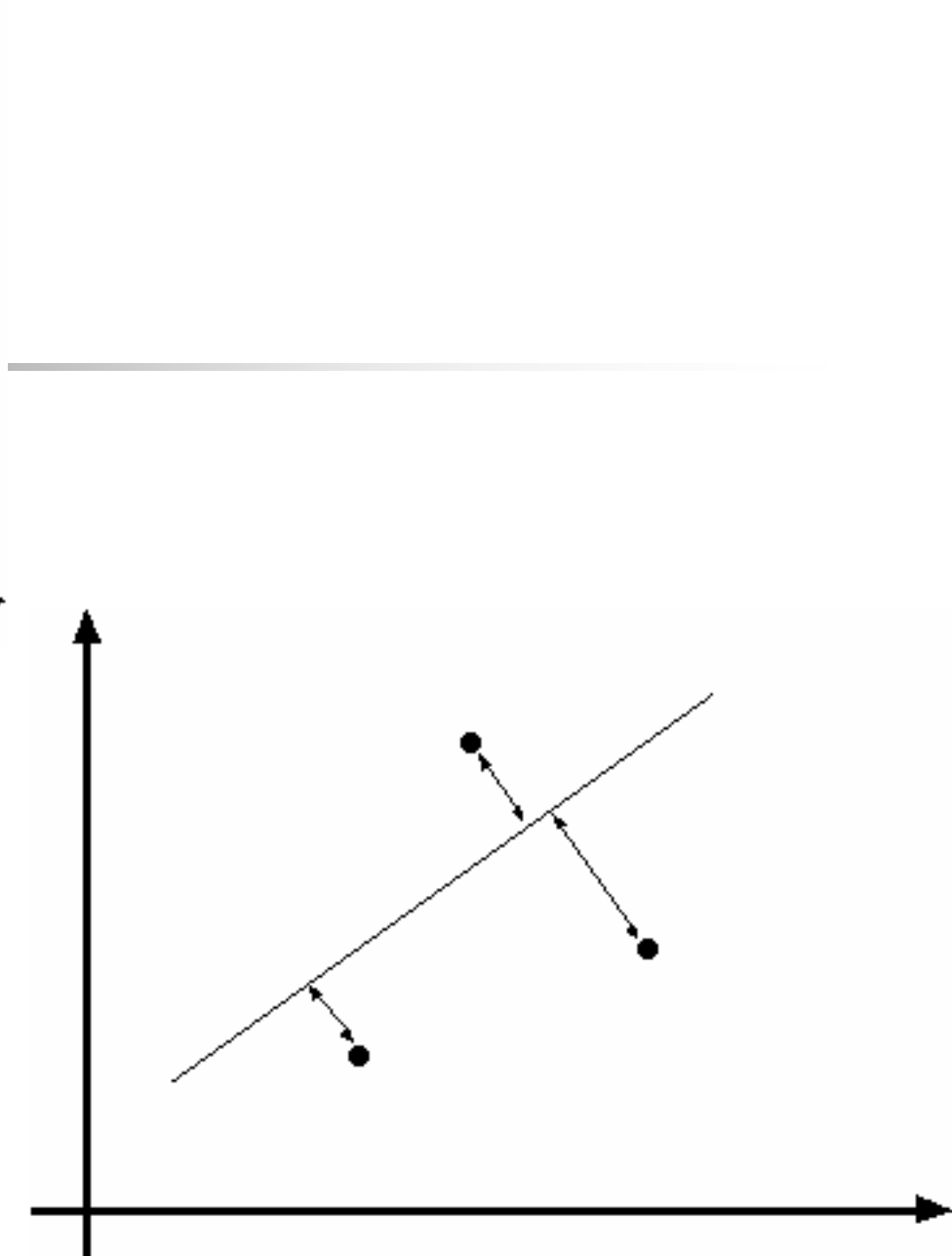
- Standard linear solution to a classical problem.
- Poor Model for vision applications.

$$y = ax + b = f(x, a, b)$$

$$\text{Minimize Err} = \sum_i [y_i - f(x_i, a, b)]^2$$



Line fitting can be max.
likelihood - but choice of
model is important





Least Squares Fit

$$\text{Minimize } E = \sum_i [y_i - (ax_i + b)]^2$$

$$\frac{\partial E}{\partial a} = \sum_i -2x_i[y_i - (ax_i + b)] = 0,$$

$$\frac{\partial E}{\partial b} = \sum_i 2[y_i - (ax_i + b)] = 0$$

Solution:

$$a = \frac{n \sum_i x_i y_i - \sum_i x_i \sum_i y_i}{n \sum_i x_i^2 - (\sum_i x_i)^2}, b = \frac{1}{n} \left(\sum_i y_i - a \sum_i x_i \right)$$



Let's make it more concise...

$$y = [y_1, y_2, y_3, \dots, y_n]', X = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \eta = [a, b]'$$

This is an over-determined linear equation set

$$X\eta = y$$

Solution: $\eta = (X^T X)^{-1} Xy$



Line Fitting

- Alternatives

- Least squares fit (take derivatives w.r.t. unknowns)
- Define observation matrices, vector of unknowns

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_B = \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & 1 \\ x_n & 1 \end{bmatrix}}_A \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_C$$

$$C = (A^T A)^{-1} A^T B$$

Matlab code

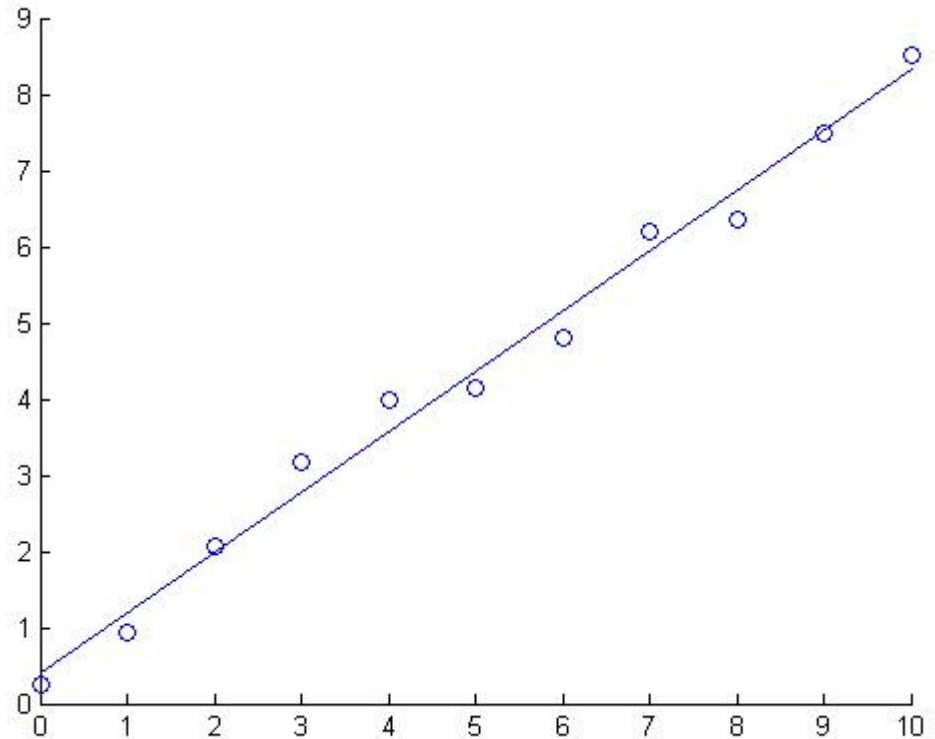
```
%a very easy demo of line fitting
a0=0.75;
b0=0.6;
x=(0:10)';
y=a0*x+b0+(rand(11,1)-0.5)*0.9;
A=[x ones(11,1)];
%coeff1=A\y;
coeff2=pinv(A)*y;
coeff=coeff2;
figure;
scatter(x,y);
line([0,10],[coeff(2),coeff(1)*10+coeff(2)]);
```

%choice 1: use backslash

%choice 2: use pseudo inverse

%draw all points

%draw fitted line

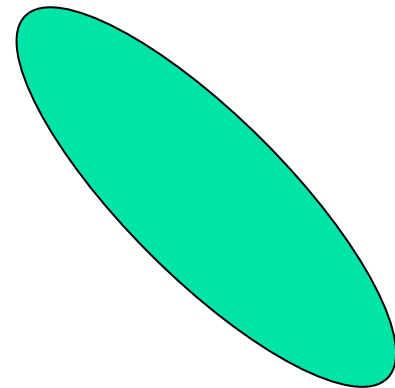
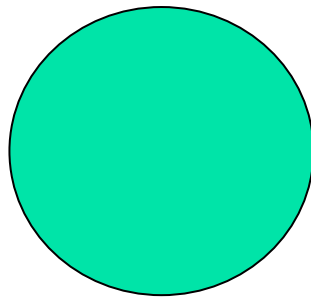




Conics Fitting

Curve described by 2nd-degree equation in the plane

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$



Five points define a conic

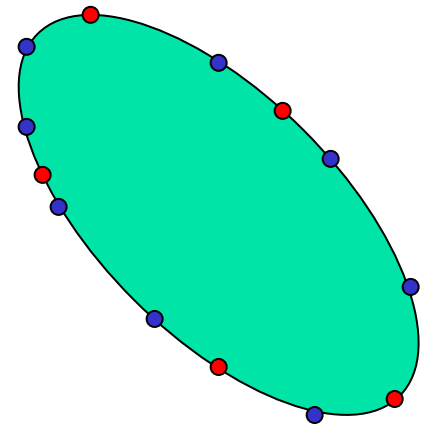
For each point the conic passes through

$$ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f = 0$$

$$\text{or } (x_i^2, x_iy_i, y_i^2, x_i, y_i, 1)\mathbf{c} = 0 \quad \mathbf{c} = (a, b, c, d, e, f)^\top$$

stacking constraints yields

$$\begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{bmatrix} \mathbf{c} = 0$$

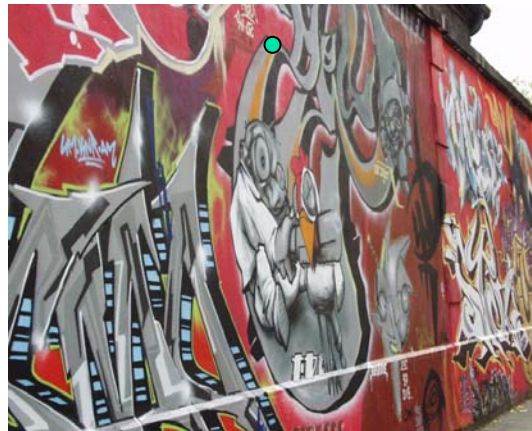




Who came from which line?

- Assume we know how many lines there are - but which lines are they?
 - easy, if we know who came from which line
- Strategies
 - Incremental line fitting
 - K-means

Invariance to ...



Scaling and rotation

Viewpoint

Illumination



Line Fitting: Challenge

- If we know nothing at all about how many lines and which point belongs to which line.....



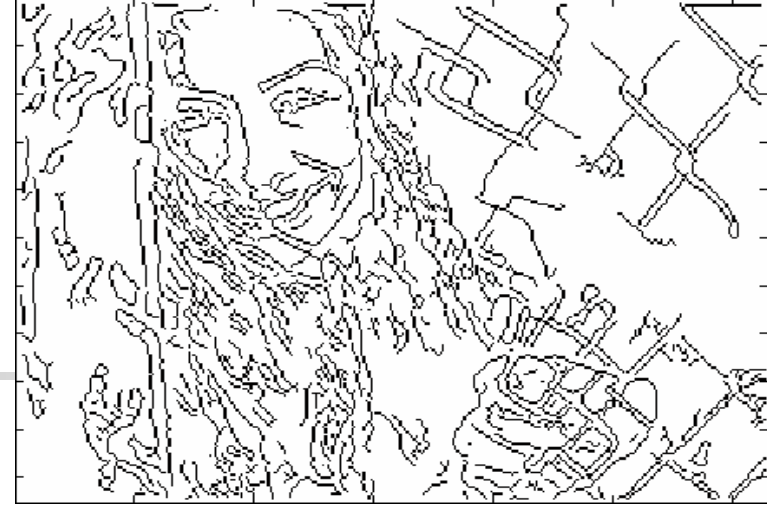


Contents

- Curve Fitting
- Hough Transform
- Robust fitting



Fitting



- Choose a parametric object/some objects to represent a set of tokens(标识点)
- Most interesting case is when criterion is not local
 - can't tell whether a set of points lies on a line by looking only at each point and the next.

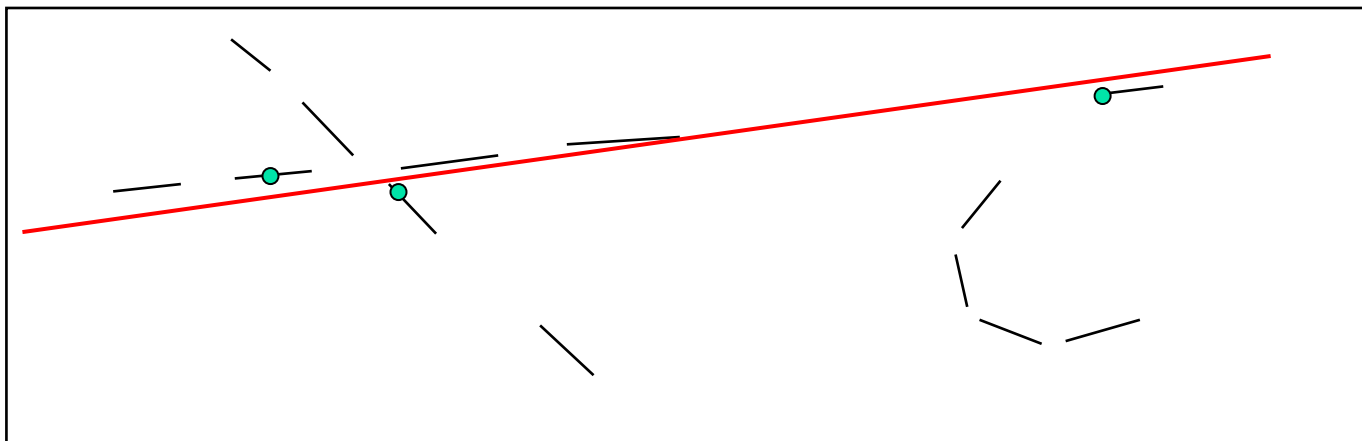


Fitting

- Three main questions:
 - what object represents this set of tokens best?
 - which of several objects gets which token?
 - how many objects are there?
- (you could read line for object here, or circle, or ellipse or...)

Finding lines in an image

- We do for lines only
 - General goal: From edges to straight lines
- Option 1:
 - Search for the line at every possible position/orientation
 - What is the cost of this operation?



- Option 2:
 - Use a **voting** scheme: **Hough transform**(霍夫变换)



Advantage of Voting

Example: Two candidates (A & B) run for the president

Assumption: the correct probability of each vote is 0.7

Scheme 1: assigned by the previous president.

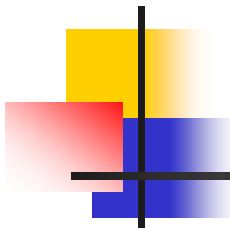
Correct rate: 70%=0.7

Scheme 2: general election, the candidate with more votes wins

Correct rate for 3 votes: $C_3^2 \cdot (0.7)^2 \cdot 0.3 + (0.7)^3 = 0.784$

Correct rate for 5 votes: $C_5^3 \cdot (0.7)^3 \cdot (0.3)^2 + C_5^4 \cdot (0.7)^4 \cdot (0.3) + (0.7)^5 = 0.837$

Correct rate for 10,000,000 votes:



Recall: Increase transmission reliability by voting

Example: Transmit one “0” or “1” bit in a channel of 80% reliability

Scheme 1: Directly transmit the bit(0 or 1).

Correct rate: 80%=0.8

Scheme 2: Transmit “000” for the bit 0 and “111” for the bit 1. Take the symbol appears more times in the received sequence as the correct one, eg. “101”->”1”
“001”->”0”, “100”->”0”, “111”->”1”

Correct rate: $C_3^2 \cdot (0.8)^2 \cdot 0.2 + (0.8)^3 = 0.896$

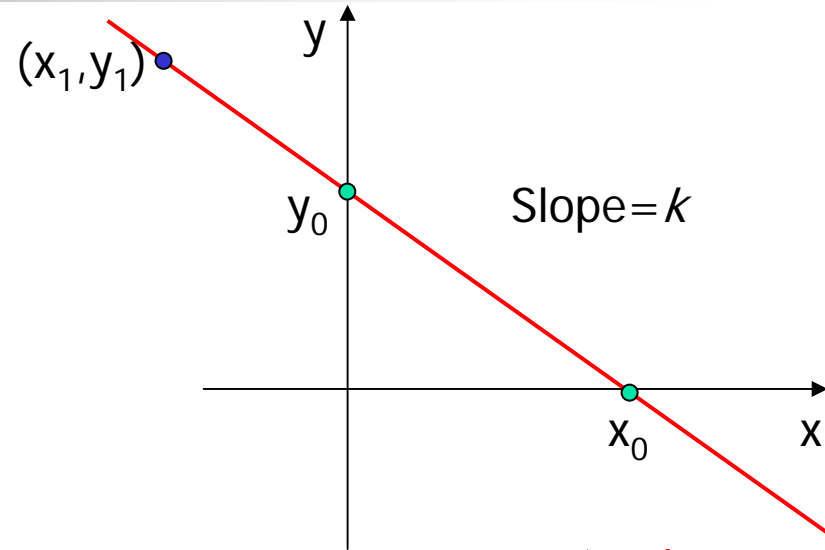
Lines in Cartesian Coordinate

A line is a set of points (x, y)

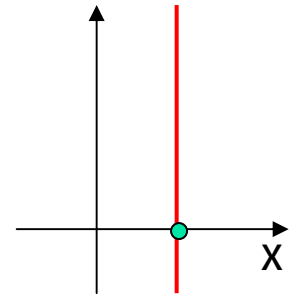
点斜式: $(y - y_1) = k(x - x_1)$

斜截式: $y = kx + y_0$

截距式: $x / x_0 + y / y_0 = 1$

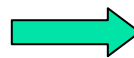


$k \rightarrow \infty$



一般式:

$$ax + by + c = 0$$



$$d = x \cos \theta + y \sin \theta$$

Hough Transform(霍夫变换)

$$b = -mx + y$$

$$b' = -m'x + y$$

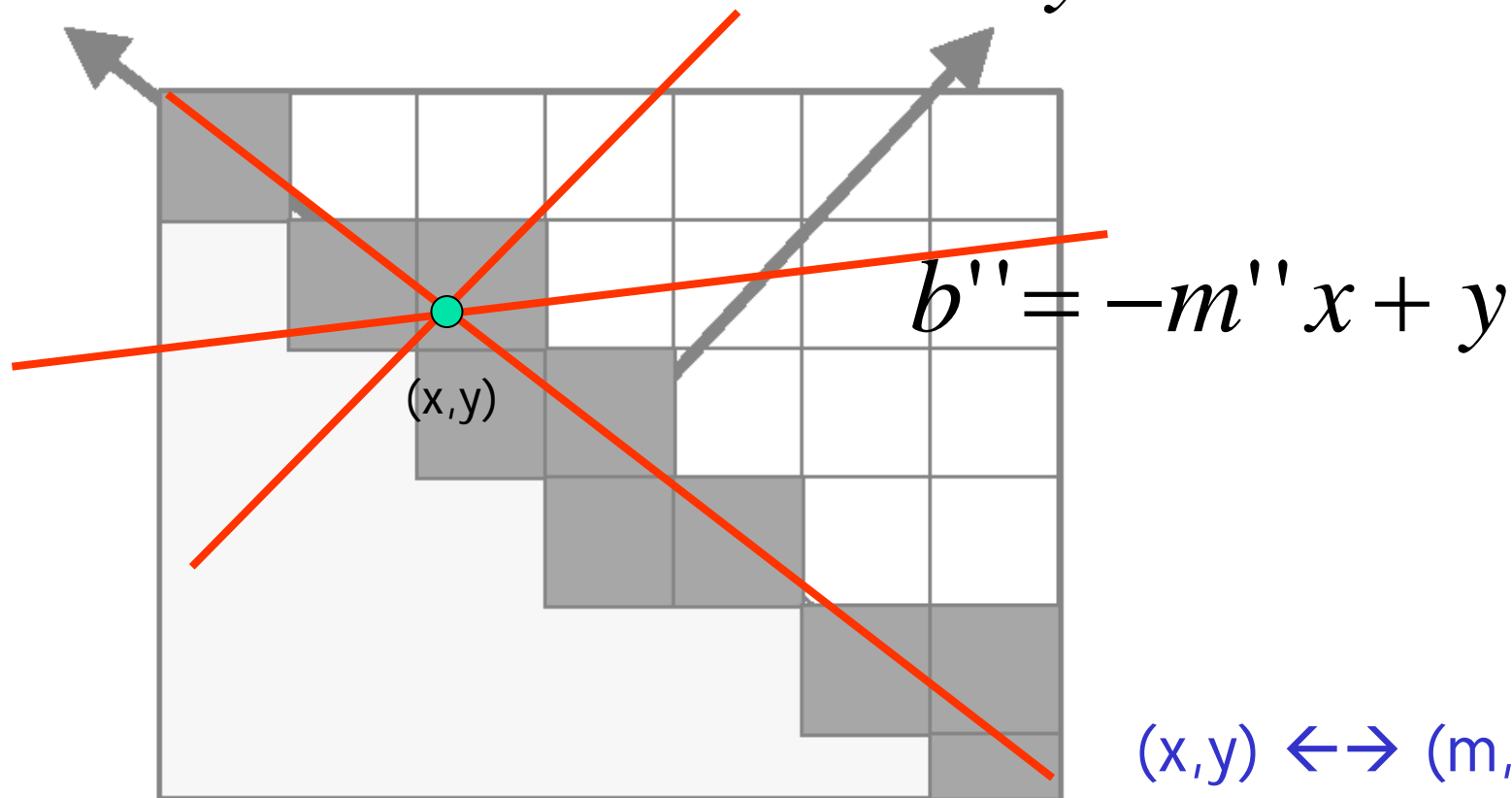
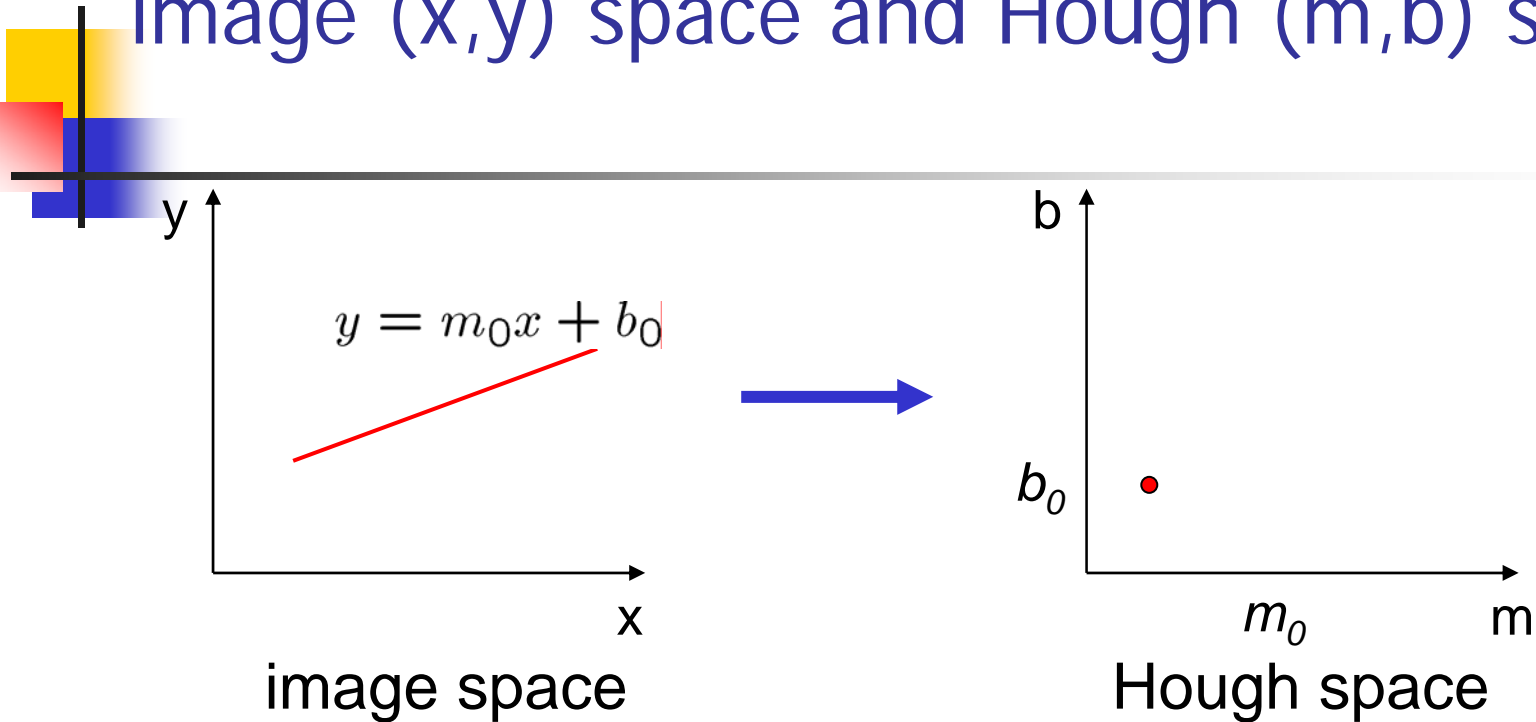
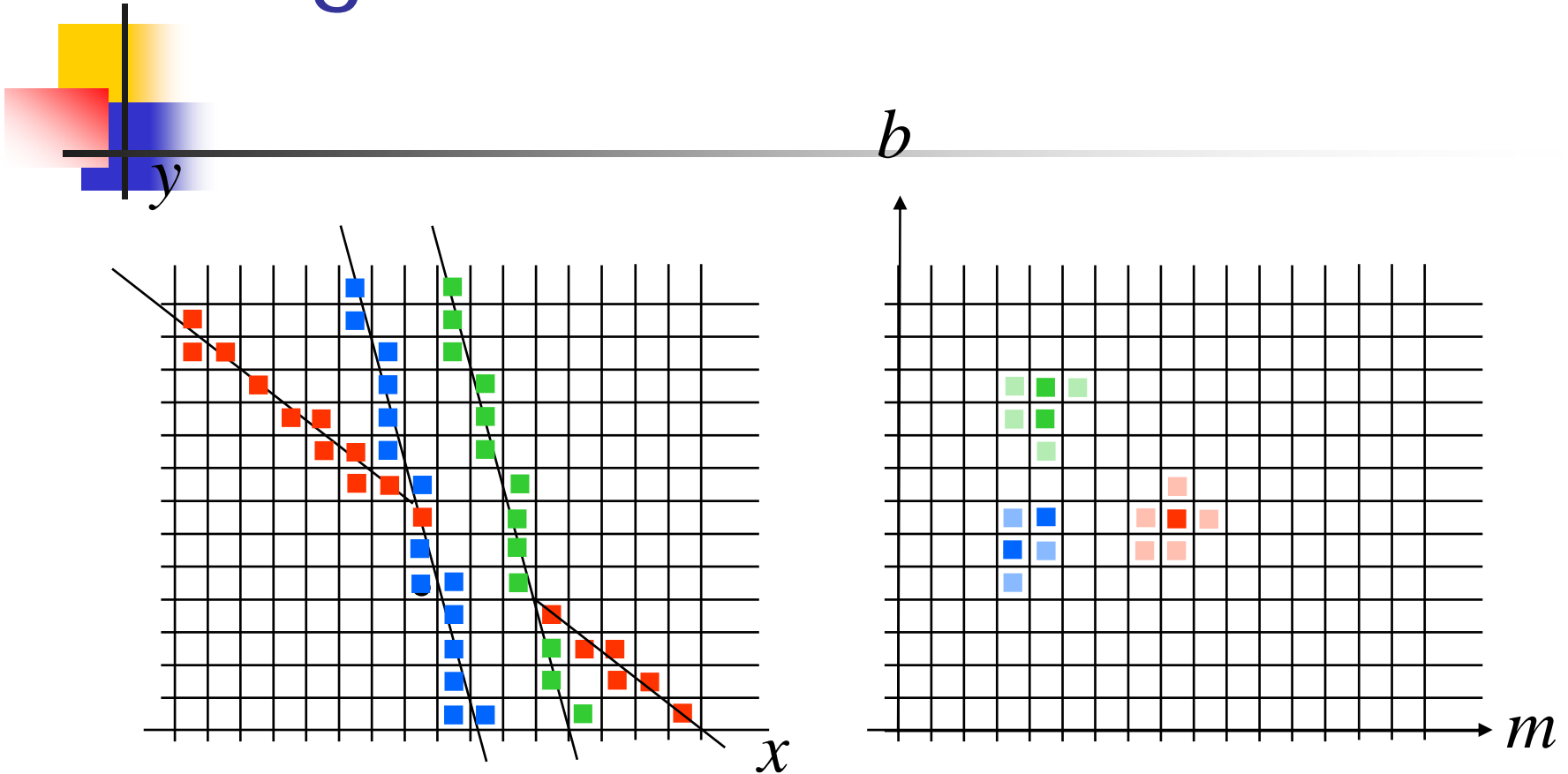


image (x,y) space and Hough (m,b) spaces



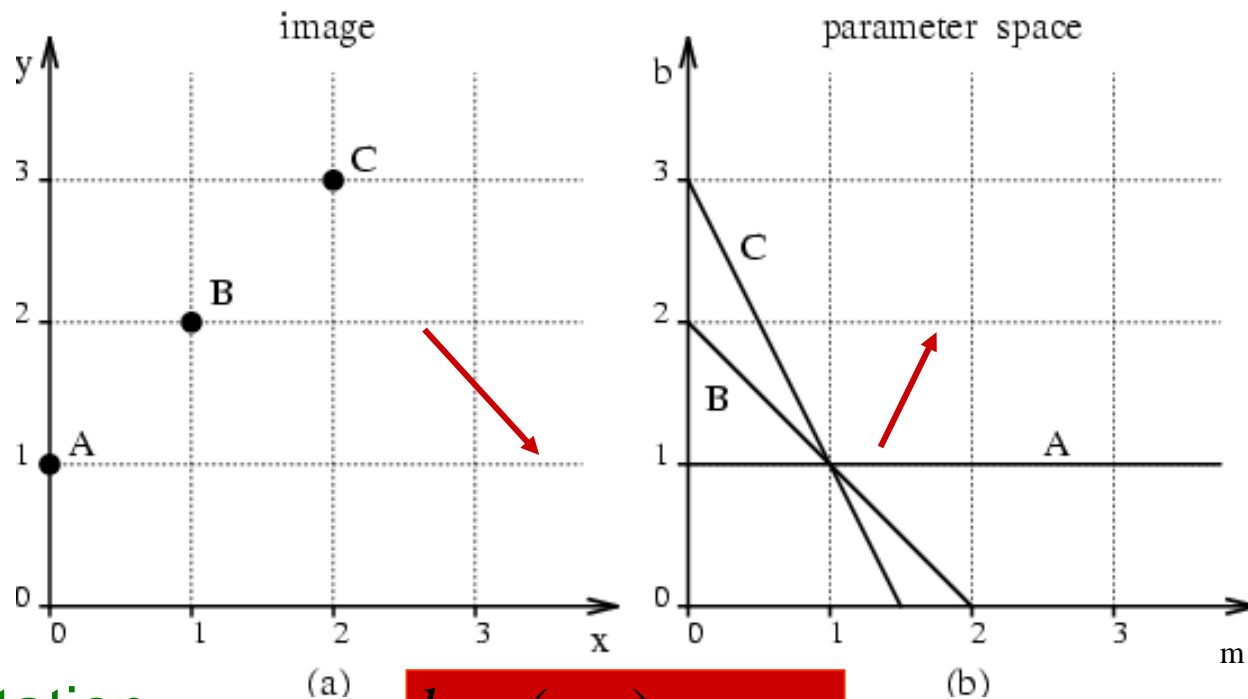
- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
 - given a set of points (x,y) , find all (m,b) such that $y = mx + b$
- What does a point (x_0, y_0) in the image space map to?
 - A: the solutions of $b = -x_0m + y_0$
 - this is a line in Hough space

Hough Transform: Quantization



Detecting Lines by finding maxima / clustering in parameter space

Hough transform: Principle



implementation :

$$b = (-x)m + y$$

1. the parameter space is discretised
2. a counter is incremented at each cell where the lines pass
- 3. peaks are detected



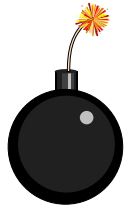
Hough Transform: Principle

- For each image point, determine
 - most likely line parameters b, m (direction of gradient)
 - strength (magnitude of gradient)
- Increment parameter counter by strength value
- Cluster in parameter space, pick local maxima

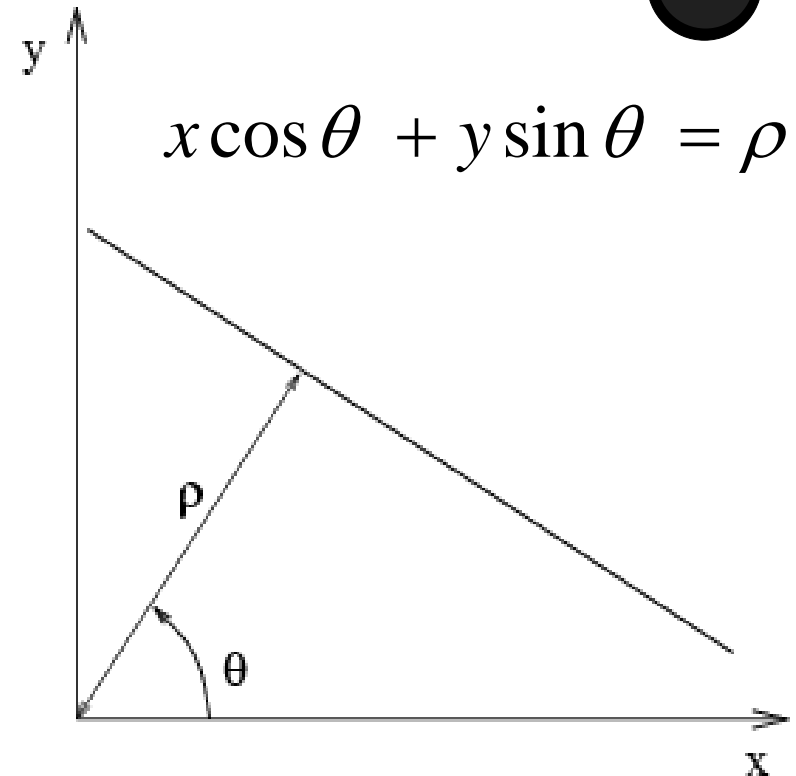
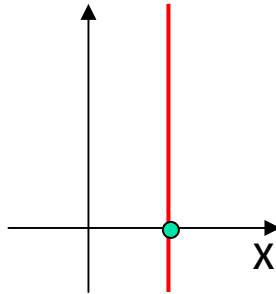
$$b = (-x)m + y$$

Hough transform: Principle

Problem : unbounded parameter domain, vertical lines require infinite m



$$m \rightarrow \infty$$



Solution: alternative representation

θ : determine the slope of the line: m

ρ : the distance from the line to the origin

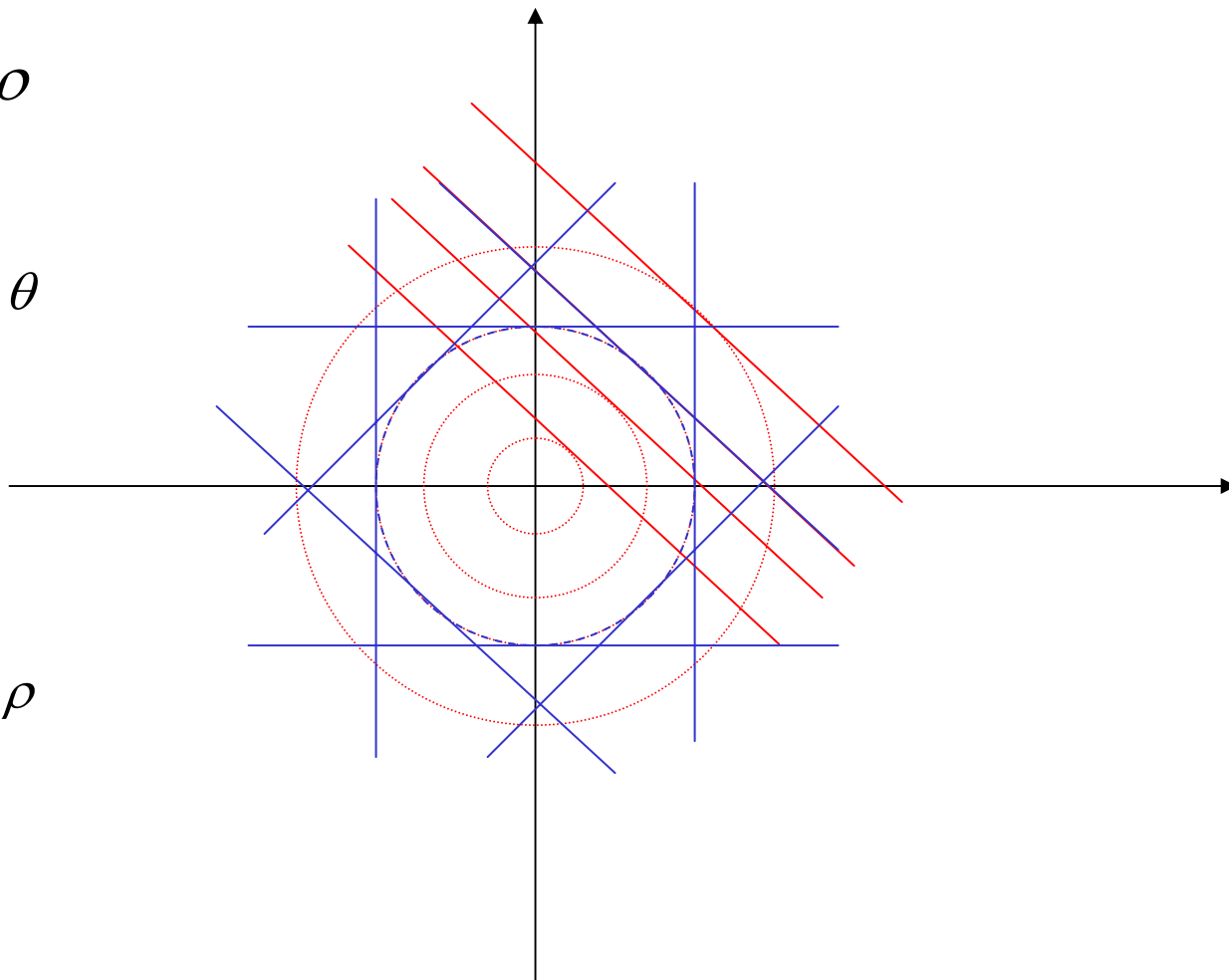
Each point will add a cosine function in the (θ, ρ) parameter space

Hough transform: Principle

$$x \cos \theta + y \sin \theta = \rho$$

Different ρ for constant θ

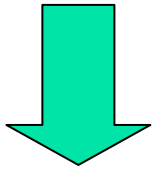
Different θ for constant ρ



Hough transform: Principle

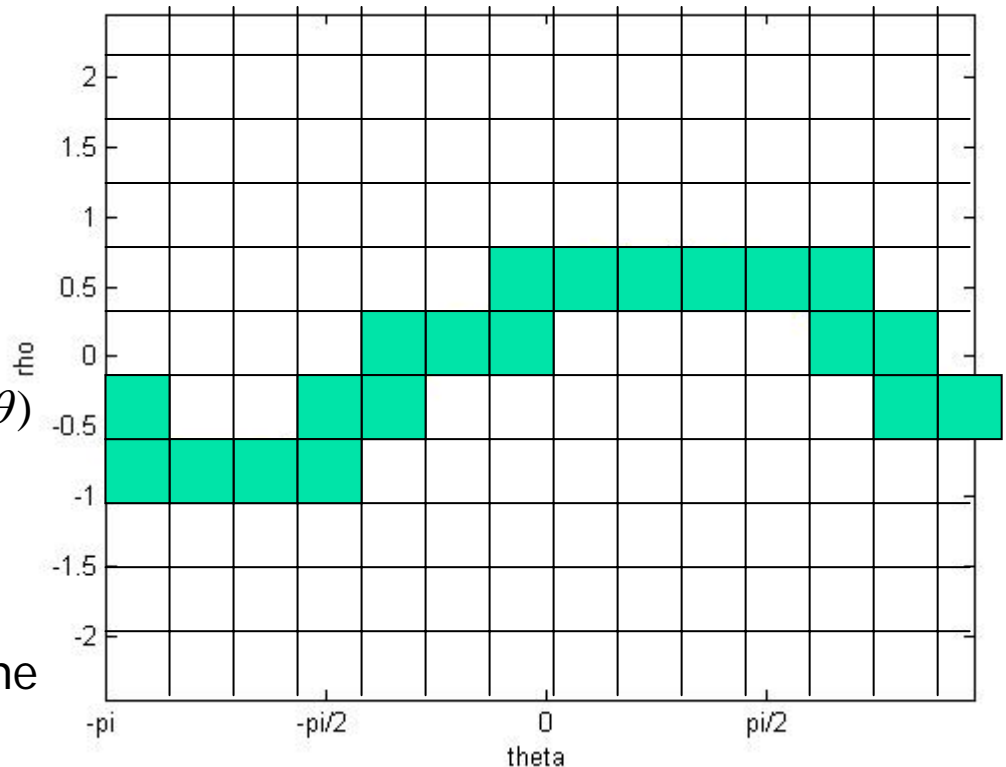
$$x \cos \theta + y \sin \theta = \rho$$

Each green bin get a vote



$$\begin{aligned} \rho &= x \cos \theta + y \sin \theta \\ &= \sqrt{x^2 + y^2} \left(\frac{x}{\sqrt{x^2 + y^2}} \cos \theta + \frac{y}{\sqrt{x^2 + y^2}} \sin \theta \right) \\ &= \sqrt{x^2 + y^2} \sin(\theta + \varphi) \end{aligned}$$

Each feature point corresponds to a sine curve in Hough space





Hough transform: Hough space

- Typically use a different parameterization

$$d = x \cos \theta + y \sin \theta$$

- d is the perpendicular distance from the line to the origin
- θ is the angle this perpendicular makes with the x axis
- Why?



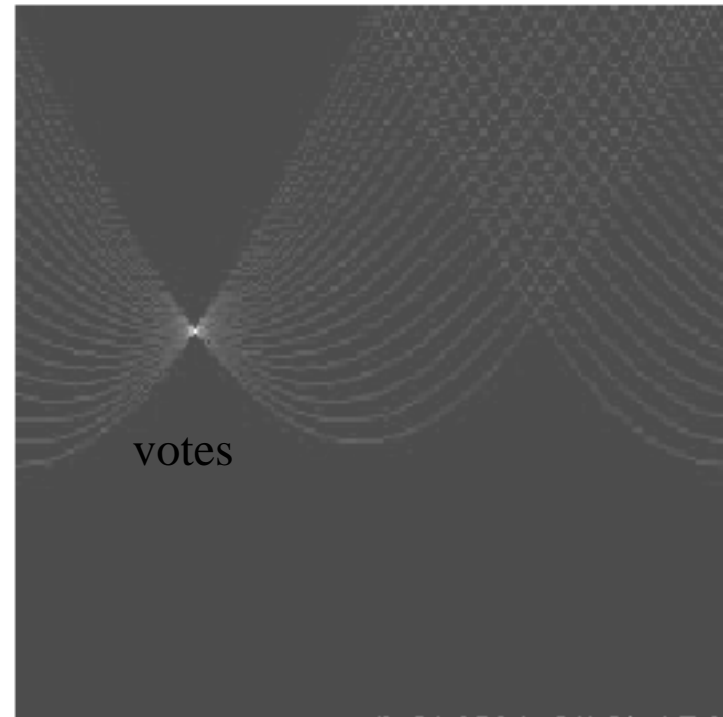
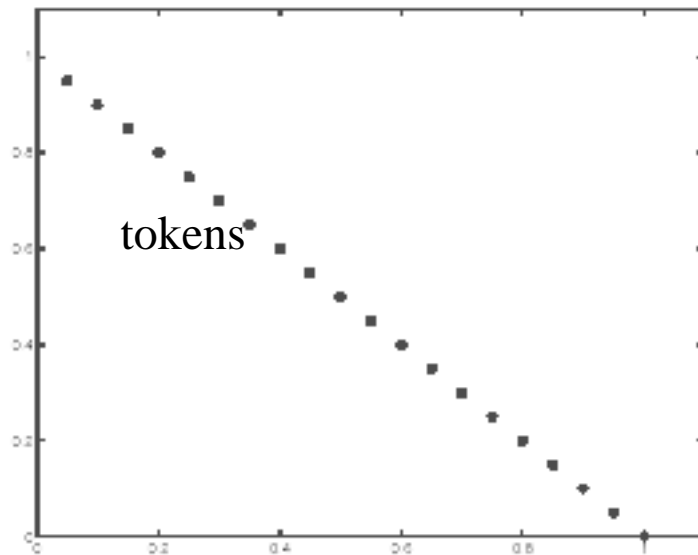
Hough Transform

- Different choices of θ , $d > 0$ give different lines
- For any (x, y) there is a one parameter family of lines through this point, given by

$$d = x \cos \theta + y \sin \theta$$

- Each point gets to vote for each line in the family; if there is a line that has lots of votes, that should be the line passing through the points

Hough Transform: Demo



$$d = x\cos\theta + y\sin\theta$$

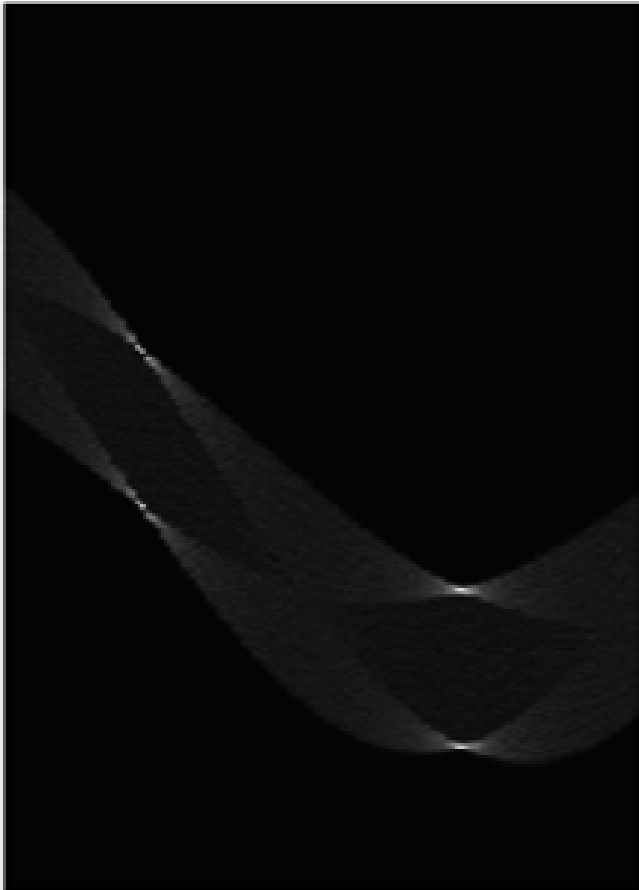


Hough transform algorithm

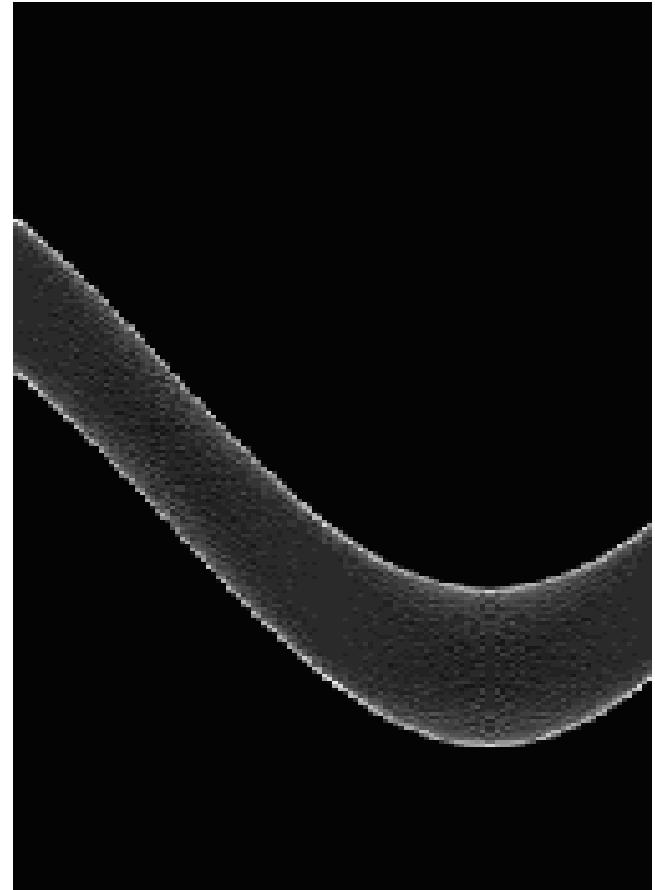
- Basic Hough transform algorithm
 1. Initialize $H[d, \theta] = 0$
 2. for each edge point $I[x, y]$ in the image
for $\theta = 0$ to 180
 $H[d, \theta] += 1$
 3. Find the value(s) of (d, θ) where $H[d, \theta]$ is maximum
 4. The detected line in the image is given by
 $d = x\cos\theta + y\sin\theta$

Hough Transform: Straight lines

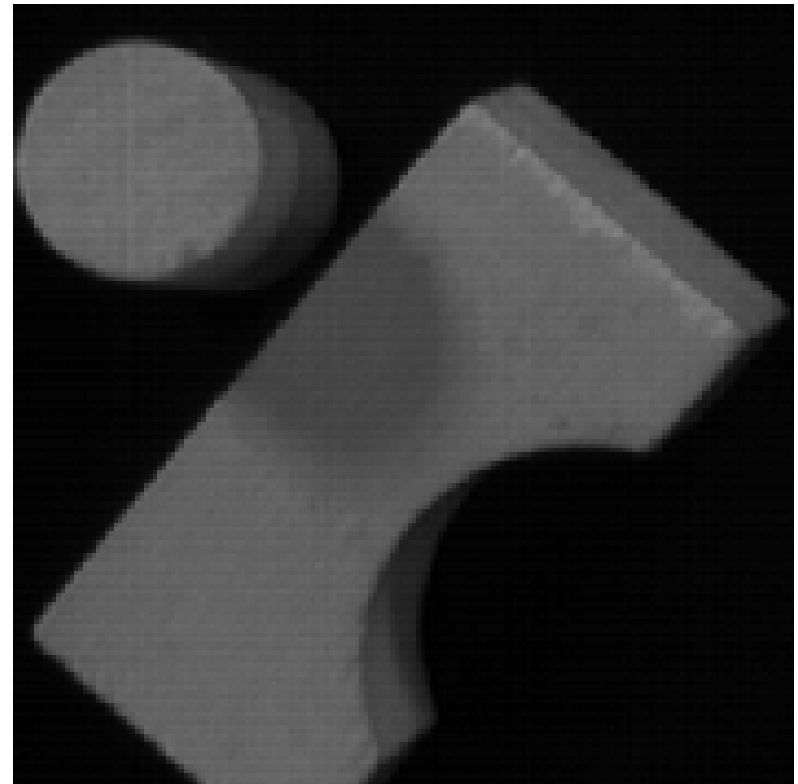
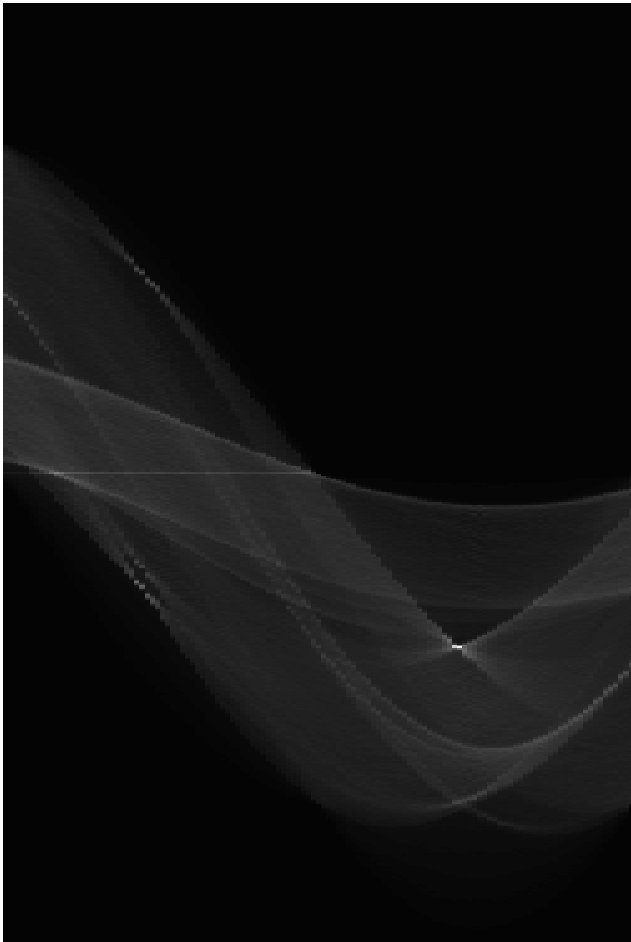
Square :



Circle :

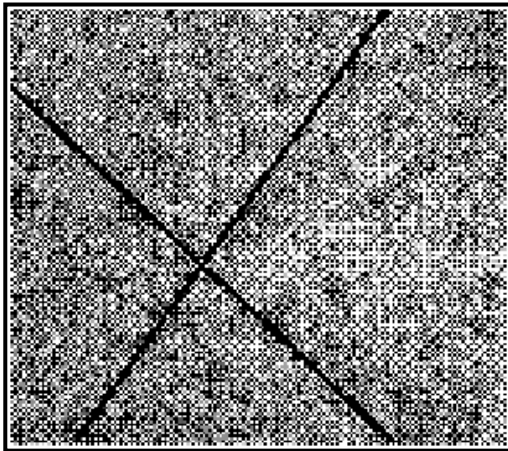


Hough Transform: Straight lines





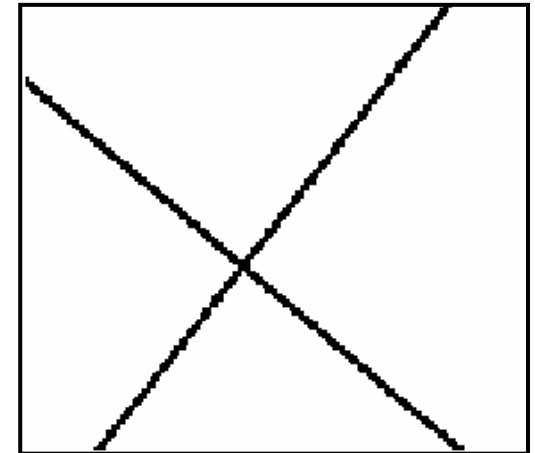
Hough Transform: Results



Image

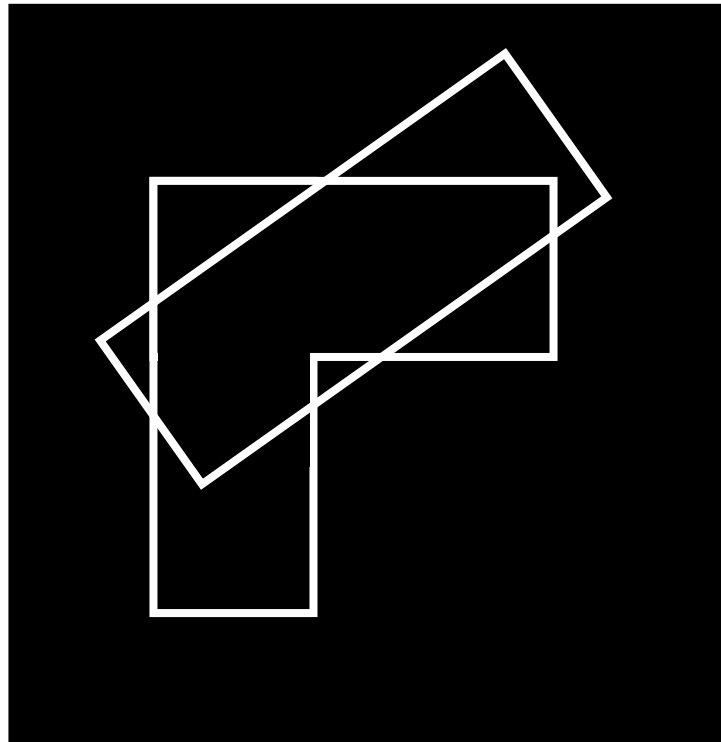


Edge detection



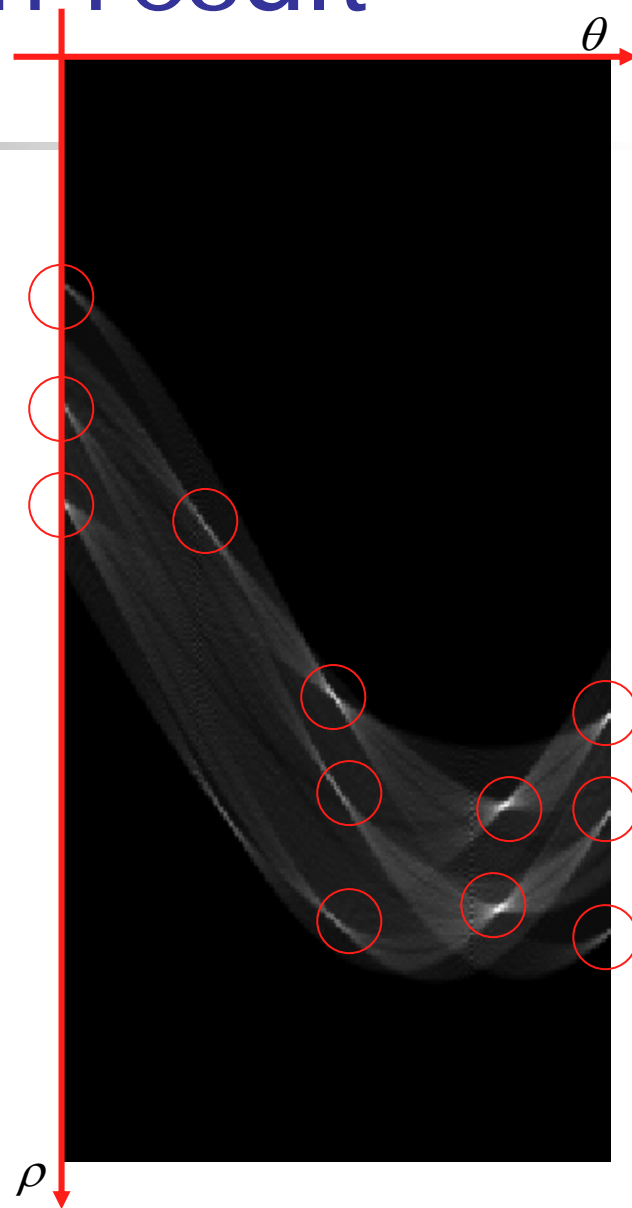
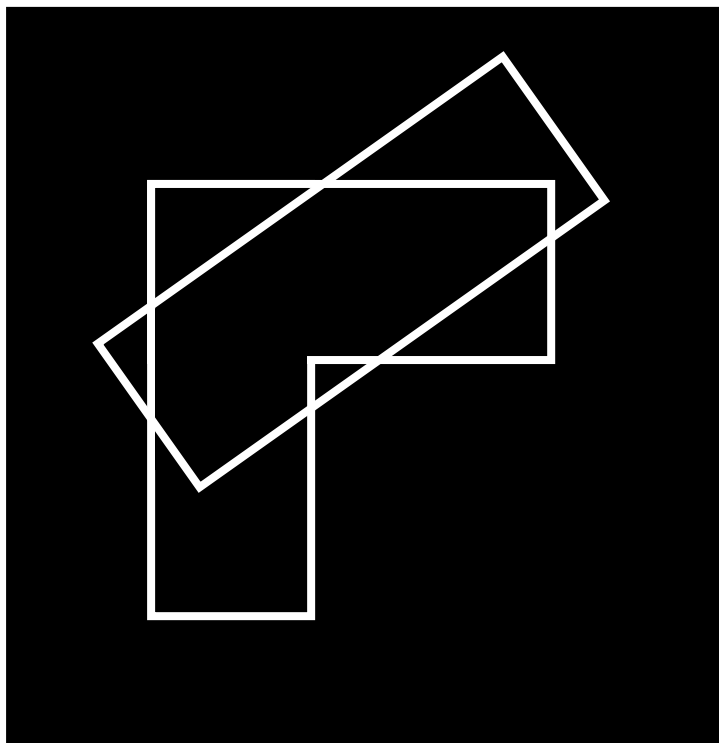
Hough Transform

Hough Transform: Examples



Problem: Looking for straight lines in the above image

Hough transform result



Hough Transform Summary:

Problem formulation

- Line equation

$y = mx + b$ m is slope, b is y - intercept

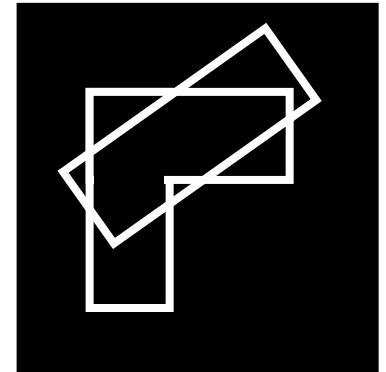
- Using edge pixels

- Compute b for every m

$$b_i = y - m_j x$$

- Problematic for vertical lines

- m and b grow to infinity



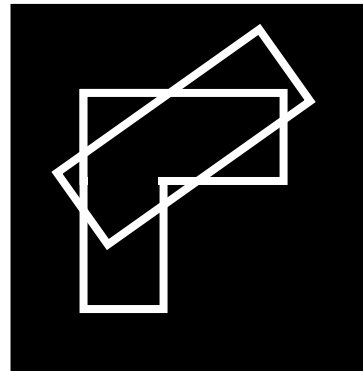
Hough space selection

- For each edge point
 - Fix m compute

$$b_i = y - m_j x$$

- Fix θ compute

$$x \cos \theta + y \sin \theta = \rho$$





Discussions

- Polar coordinate representation
 - For each point on line θ and ρ are constant
 - Numerically stable for lines in any orientation

$$x \cos \theta + y \sin \theta = \rho \quad (\mathbf{A})$$

- Different choices of θ for constant ρ gives different choices of lines, and vice versa

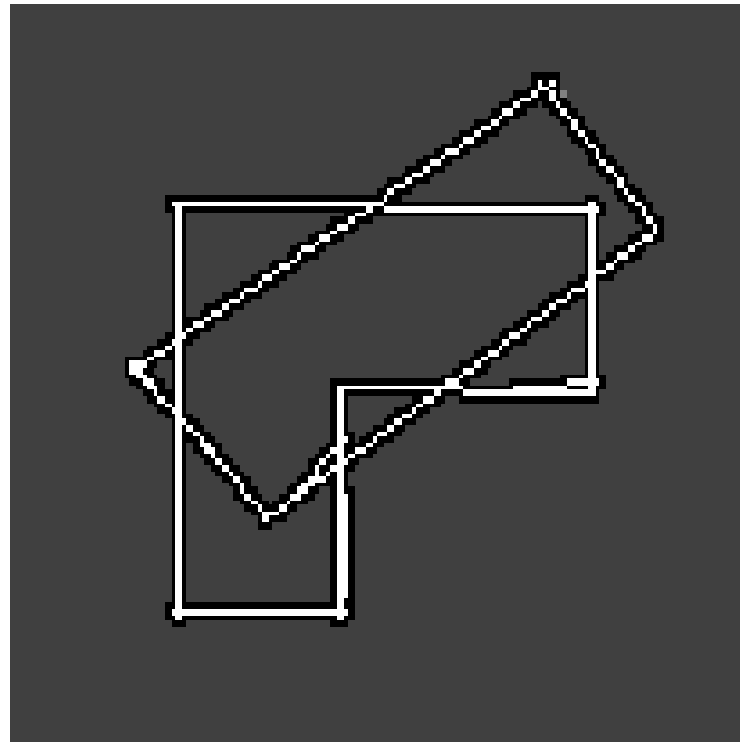
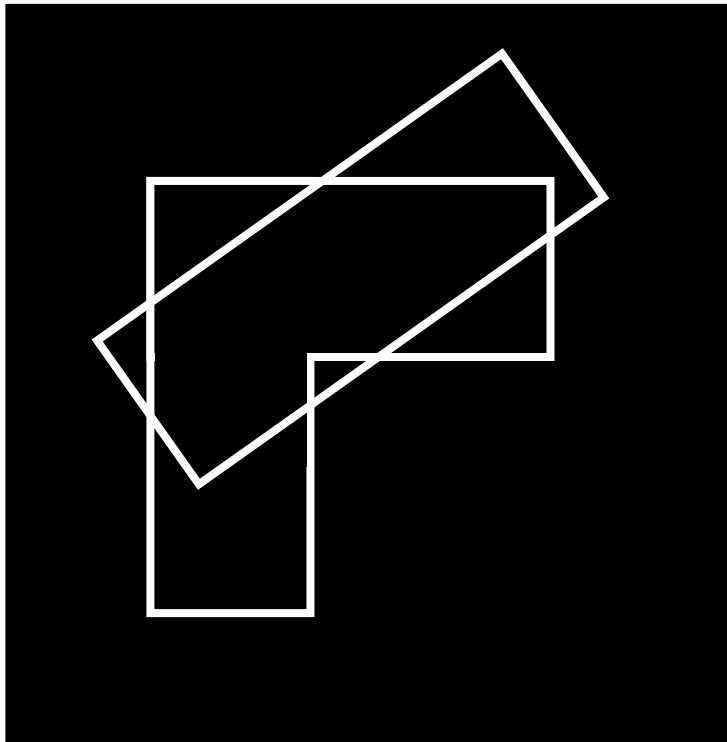


Algorithm

- Construct accumulator array in 2D (θ, ρ)
 - Initial values 0
- Select granularity of angle θ
 - For instance 10° increments
- For every edge point
 - Compute ρ using (A)
 - Increment accumulator array by one for each computed (θ, ρ) pair.

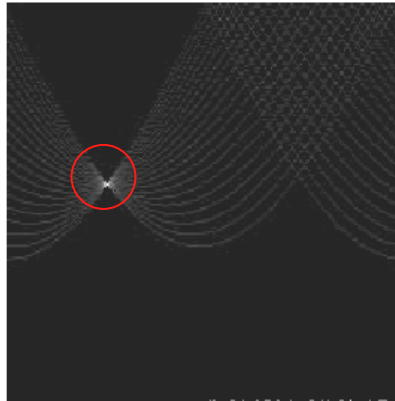
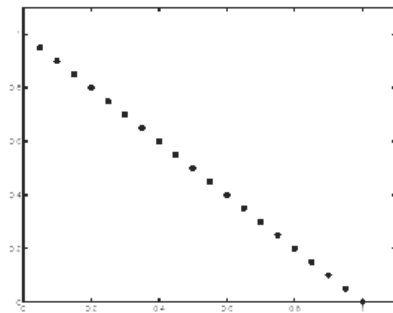


Line detection result

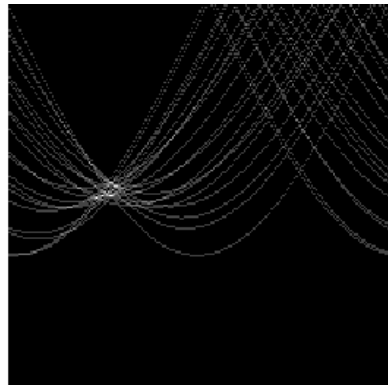
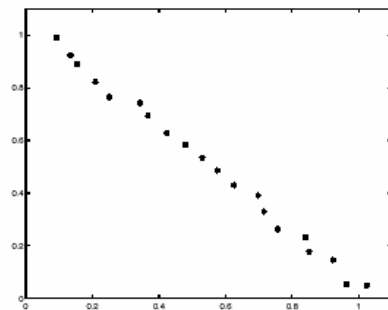


Error analysis

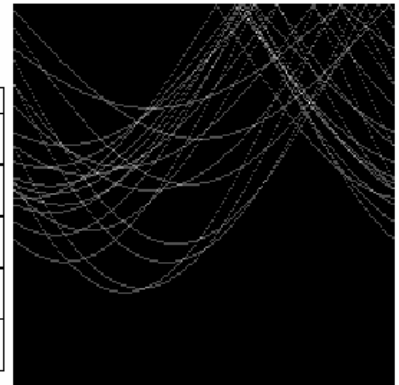
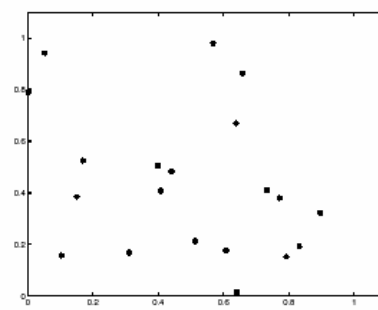
ideal

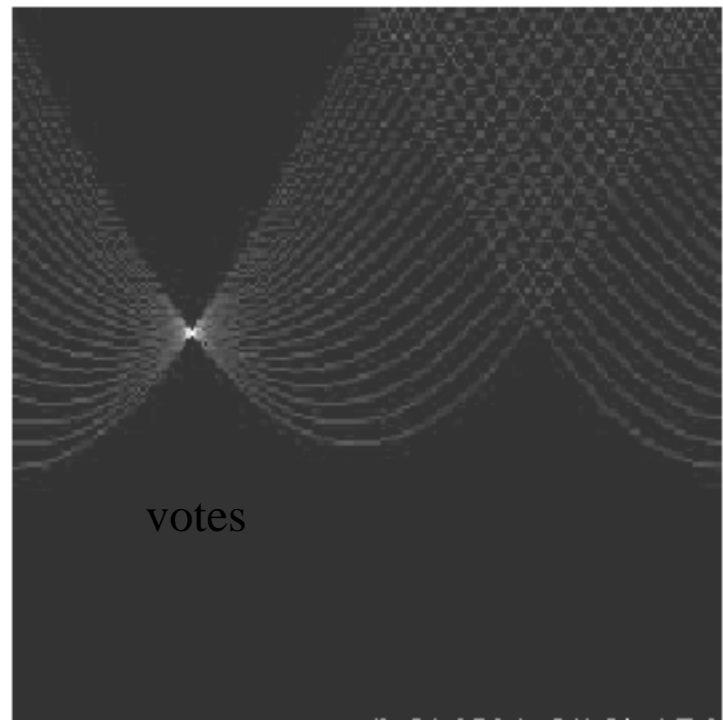
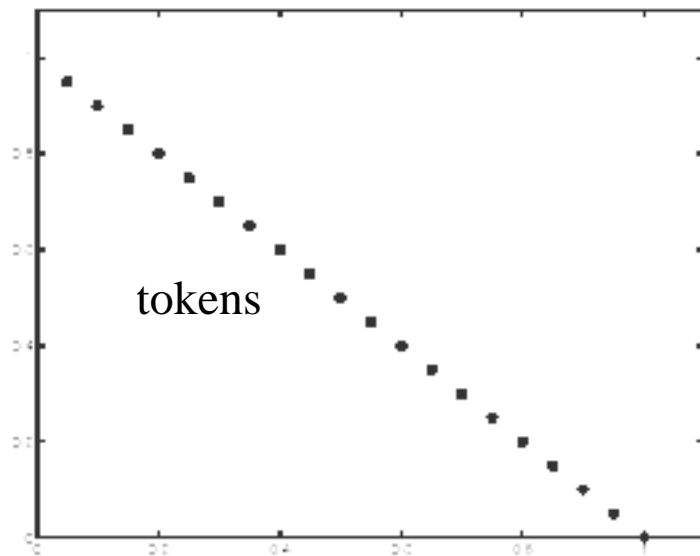
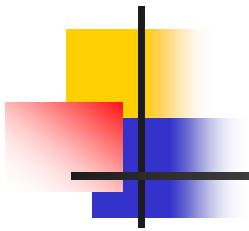


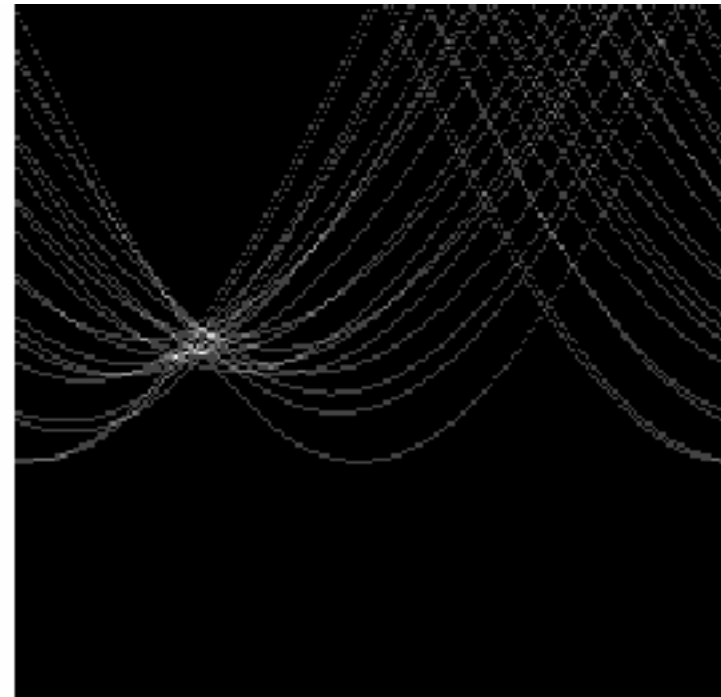
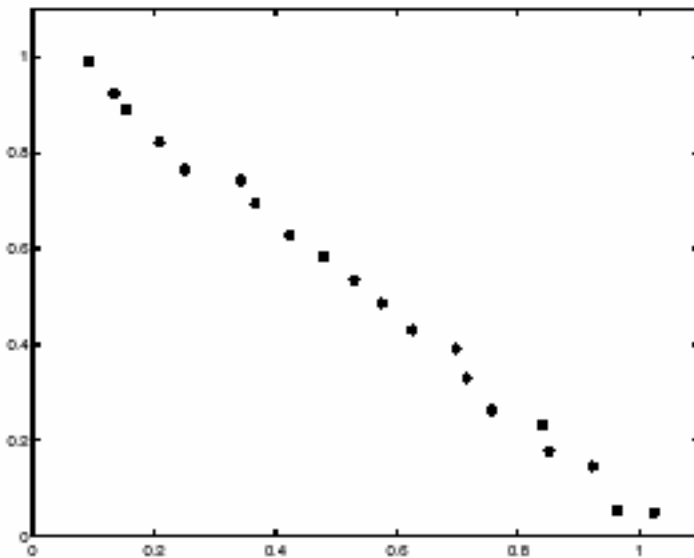
noisy



very noisy



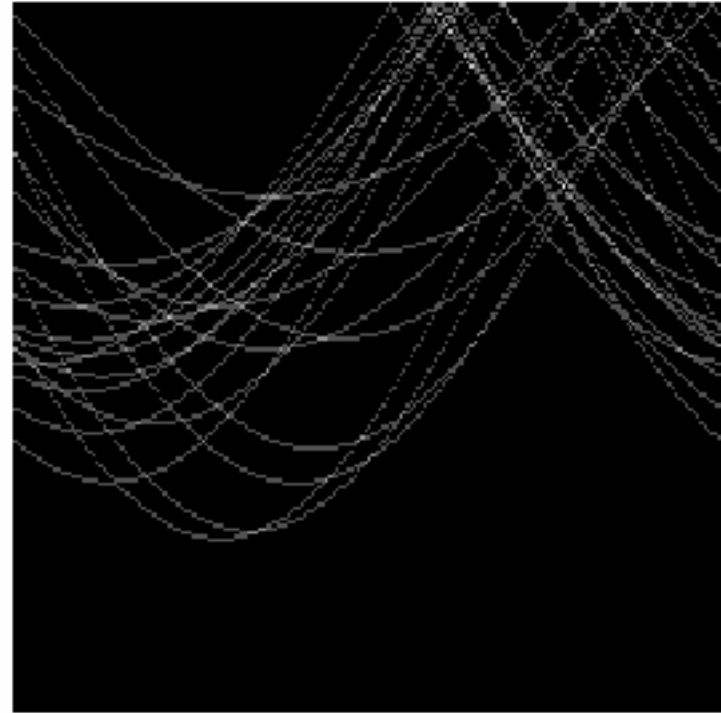
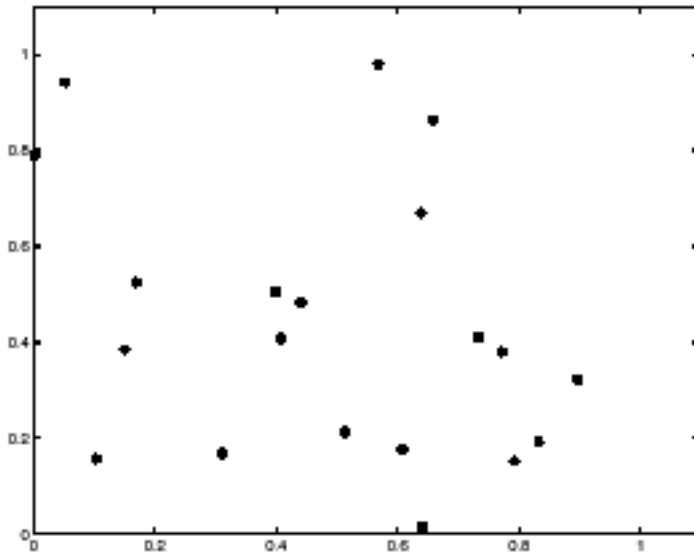
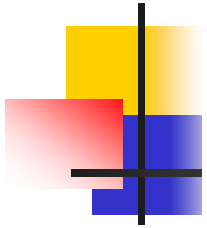




tokens

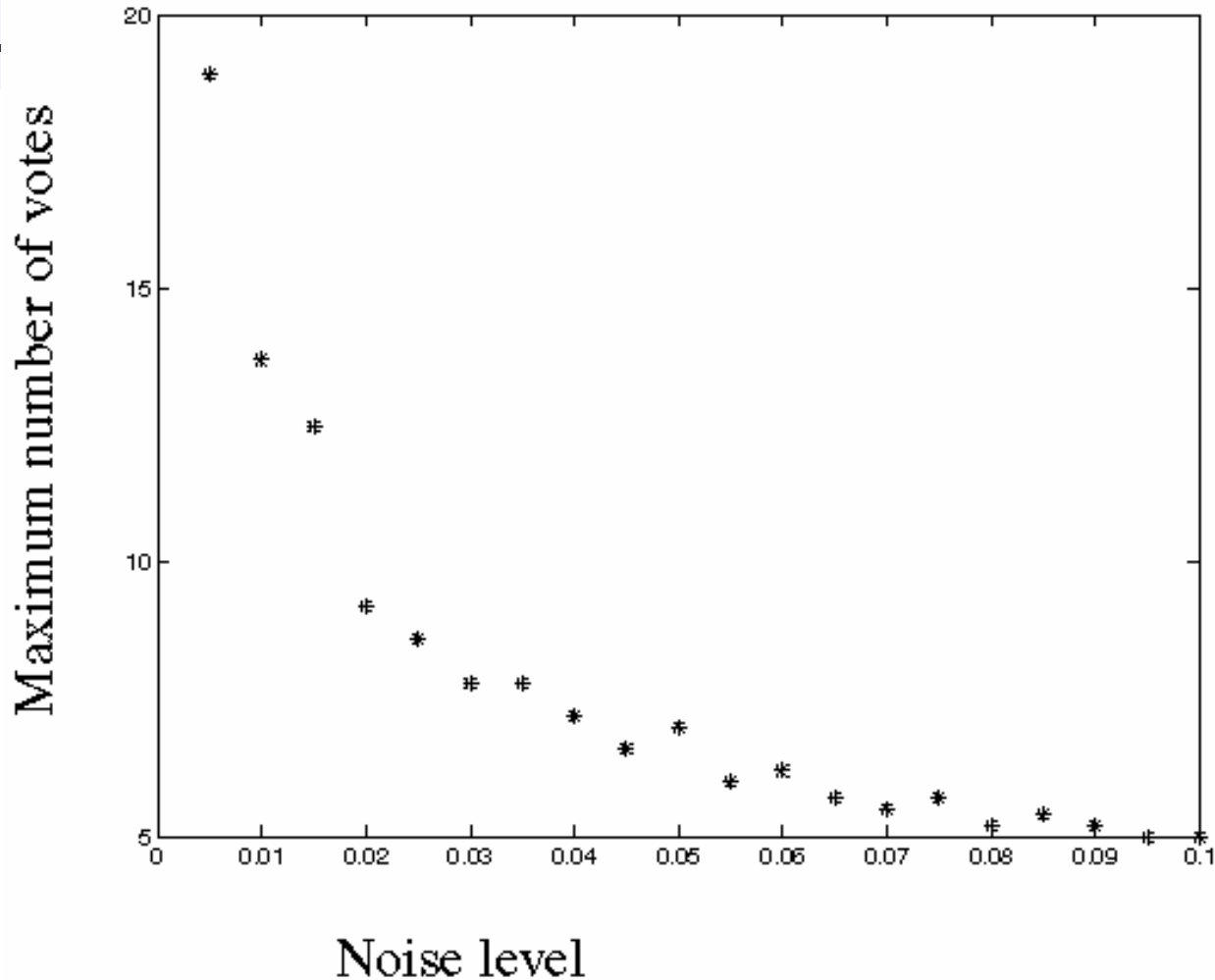
votes

Noise degrades the performance



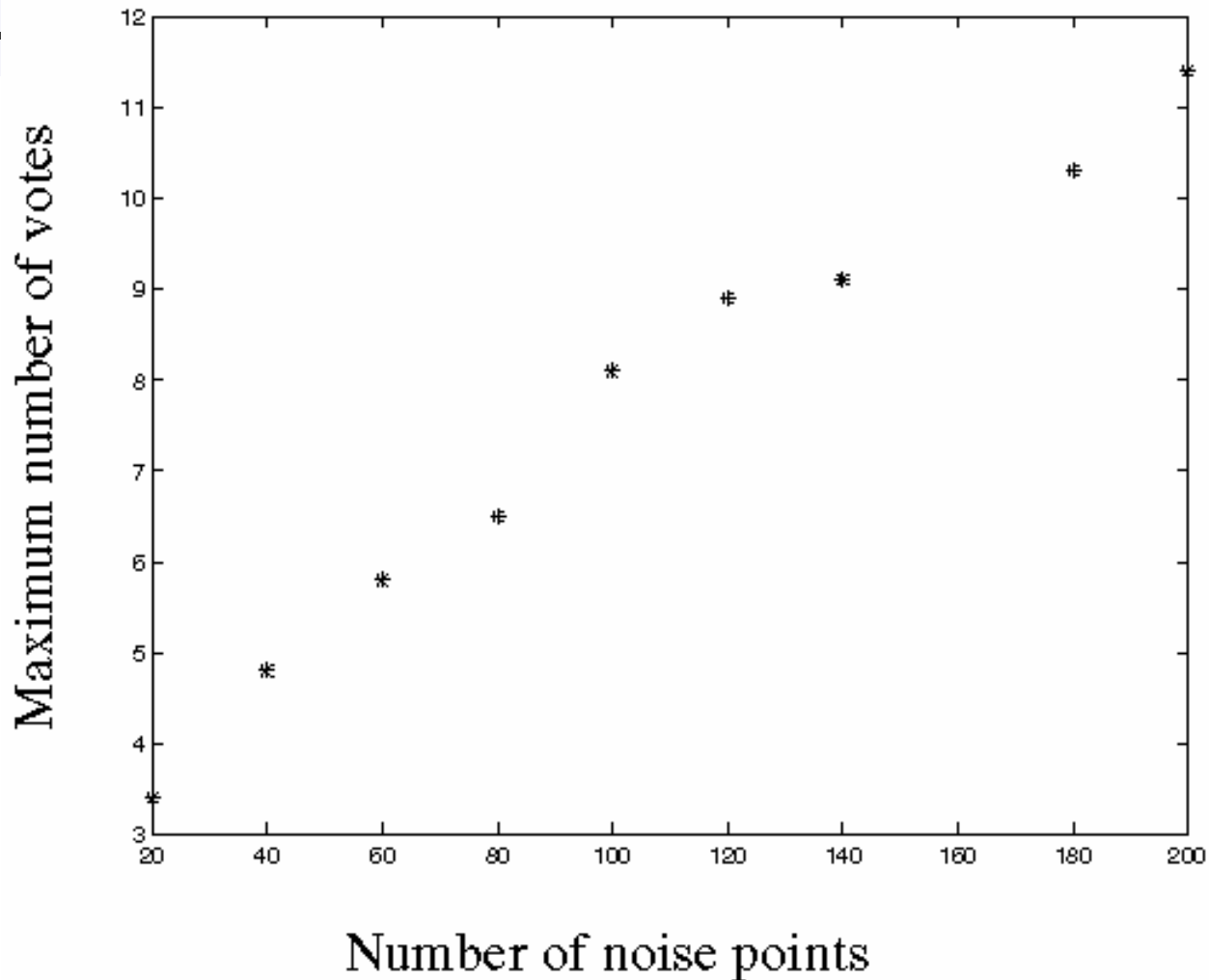
lots of noise can lead to large peaks in the array

Noise Factor



This is the number of votes that the **real line** of 20 points gets with increasing noise

Noise Factor



As the noise increases in a picture **without a line**, maximal number of votes goes up too.



Difficulties

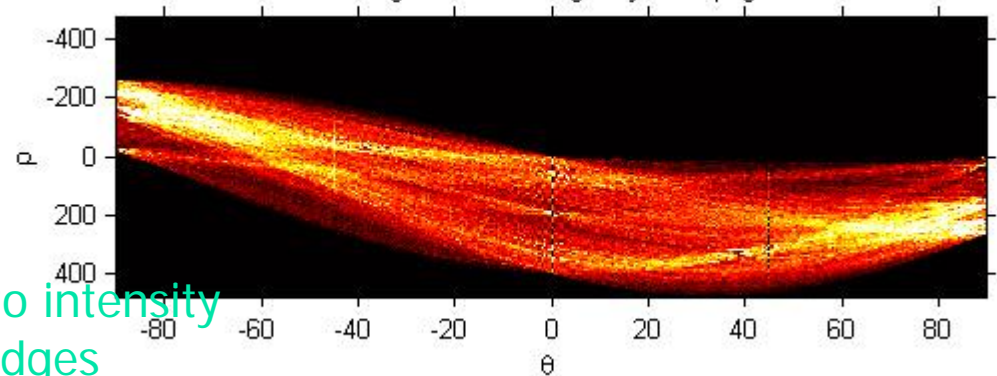
- What is the increments for θ and ρ .
 - too big? We cannot distinguish between different lines
 - too small? noise causes lines to be missed
- How many of detected lines are correct
 - count the peaks in the Hough array
- Which edge point belongs to which line
 - tag the votes
- Hardly ever satisfactory due to noise.
 - problems with noise and cell size defeat it

Matlab Demos

gantrycrane.png

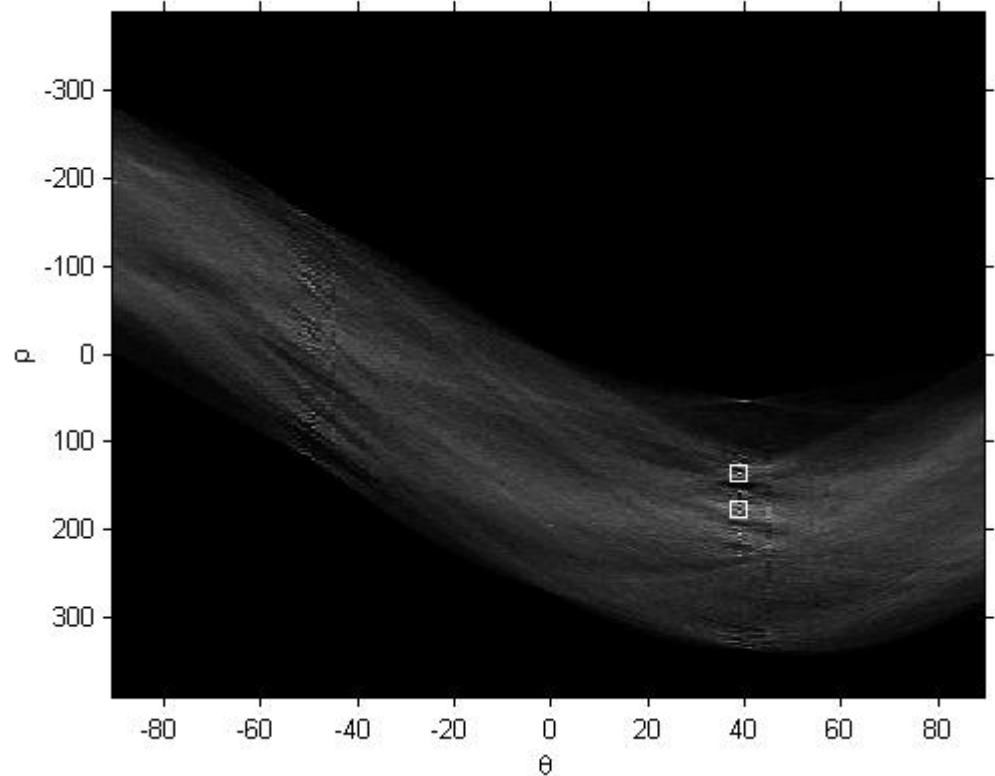
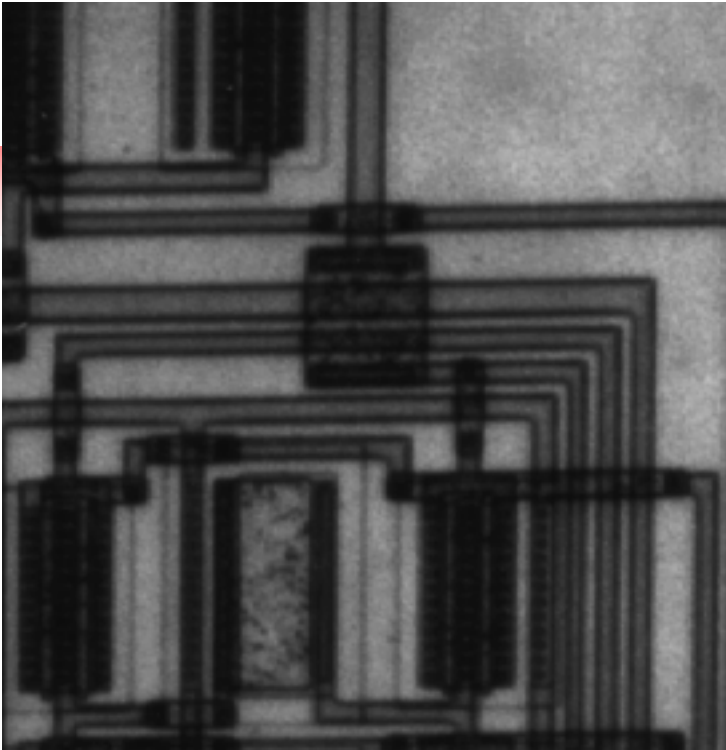


Hough transform of gantrycrane.png



```
RGB = imread('gantrycrane.png');  
I = rgb2gray(RGB); %convert to intensity  
BW = edge(I,'canny'); %extract edges  
[H,T,R] = hough(BW,'RhoResolution',0.5,'ThetaResolution',0.5); %hough transform  
subplot(2,1,1);  
imshow(RGB); %display original image  
title('gantrycrane.png');  
subplot(2,1,2);  
imshow(imadjust(mat2gray(H)),'XData',T,'YData',R,'InitialMagnification','fit');  
title('Hough transform of gantrycrane.png');  
xlabel('\theta'), ylabel('\rho');  
axis on, axis normal, hold on;  
colormap(hot);
```

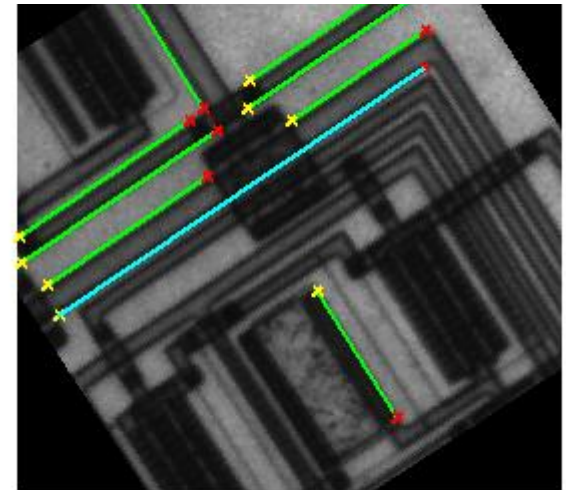
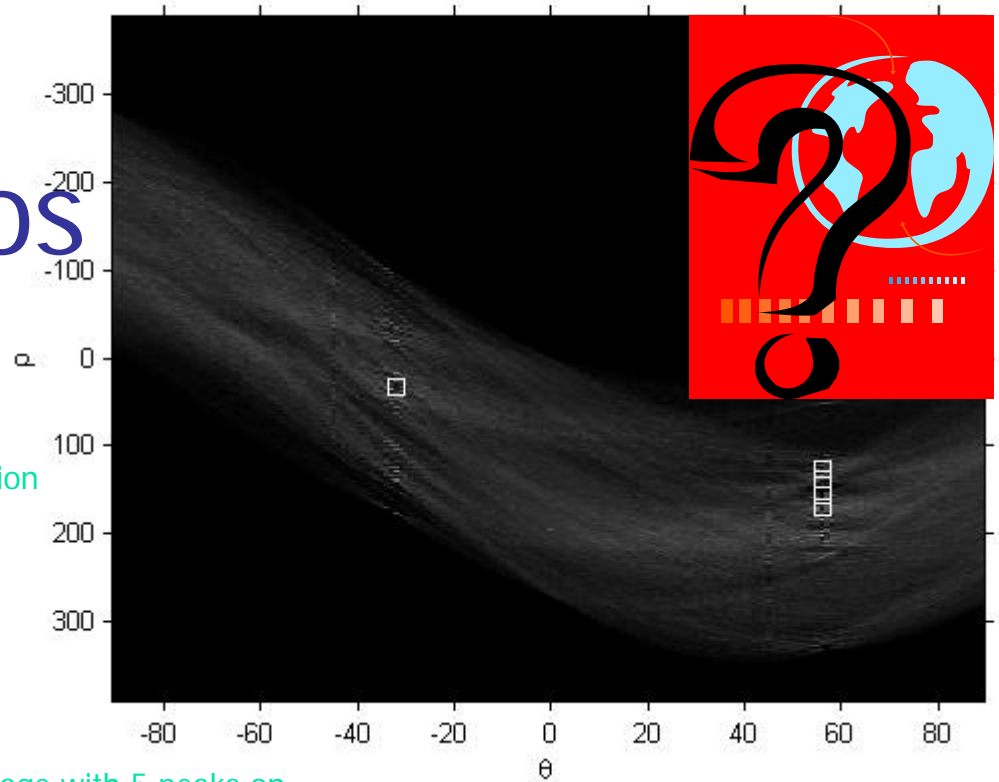
Matlab Demos



```
I = imread('circuit.tif');  
BW = edge(imrotate(I,50,'crop'),'canny');    %canny edge detector  
[H,T,R] = hough(BW);  
P = houghpeaks(H,2); % P records indices of 2 highest peak T-R pairs  
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');  
xlabel('\theta'), ylabel('\rho');  
axis on, axis normal, hold on;  
plot(T(P(:,2)),R(P(:,1)),'s','color','white'); %draw hough peaks as white squares
```

Matlab Demos

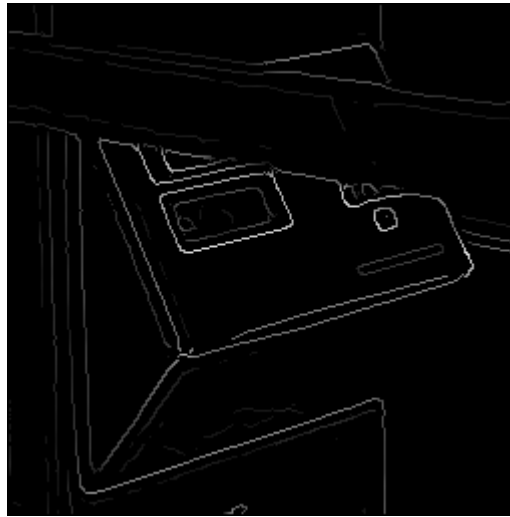
```
I = imread('circuit.tif');
rotI = imrotate(I,33,'crop'); %rotate to prevent degradation
BW = edge(rotI,'canny');
[H,T,R] = hough(BW);
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
P = houghpeaks(H,5,'threshold',ceil(0.3*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white'); %draw hough transform image with 5 peaks on
lines = houghlines(BW,T,R,P,'FillGap',5,'MinLength',7);
figure, imshow(rotI), hold on;
max_len = 0;
for k = 1:length(lines)
    xy = [lines(k).point1; lines(k).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green'); %green line
    plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow'); %yellow start point
    plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red'); %red end point
    len = norm(lines(k).point1 - lines(k).point2);
    if ( len > max_len)
        max_len = len;
        xy_long = xy;
    end
end
plot(xy_long(:,1),xy_long(:,2),'LineWidth',2,'Color','cyan'); % highlight the longest line segment
```



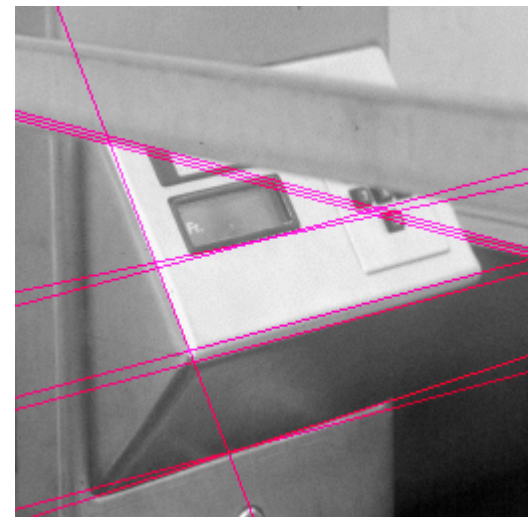
Real World Example



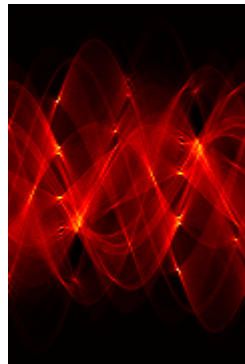
Original



Edge Detection



Found Lines



Parameter Space



Curve Fitting by Hough Transform

- Let $y=f(x,\mathbf{a})$ be the chosen parameterization of a target curve.
- Discretize the intervals of variation of a_1, \dots, a_k and let s_1, \dots, s_k be the number of the discretized intervals.
- Let $A(s_1, \dots, s_k)$ be an array of integer counters and initialize all its elements to zero.
- For each pixel $E(i,j)$ such that $E(i,j)=1$, increment all counters on the curve defined by $y=f(x,\mathbf{a})$ in A .
- Find all local maxima above certain threshold.



Curve Fitting by Hough Transform

- Suffer with the same problems as line fitting by Hough Transform.
- Computational complexity and storage complexity increase rapidly with number of parameters.
- Not very robust to noise



Circle Fitting

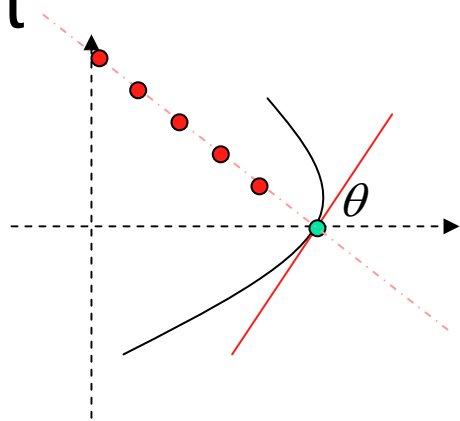
- Similar to line fitting
 - Three unknowns

$$(x - x_o)^2 + (y - y_o)^2 - r^2 = 0$$

- Construct a 3D accumulator array **A**
 - Dimensions: x_0, y_0, r
- Fix one of the parameters change the others
- Increment corresponding entry in **A**.
- Find the local maxima in **A**

More Practical Circle Fitting

- Use the tangent direction θ at the edge point



- Compute x_0, y_0 given x, y, r

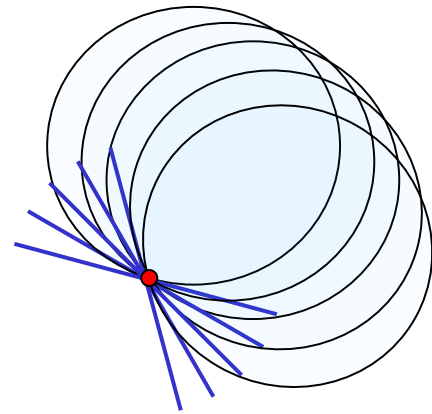
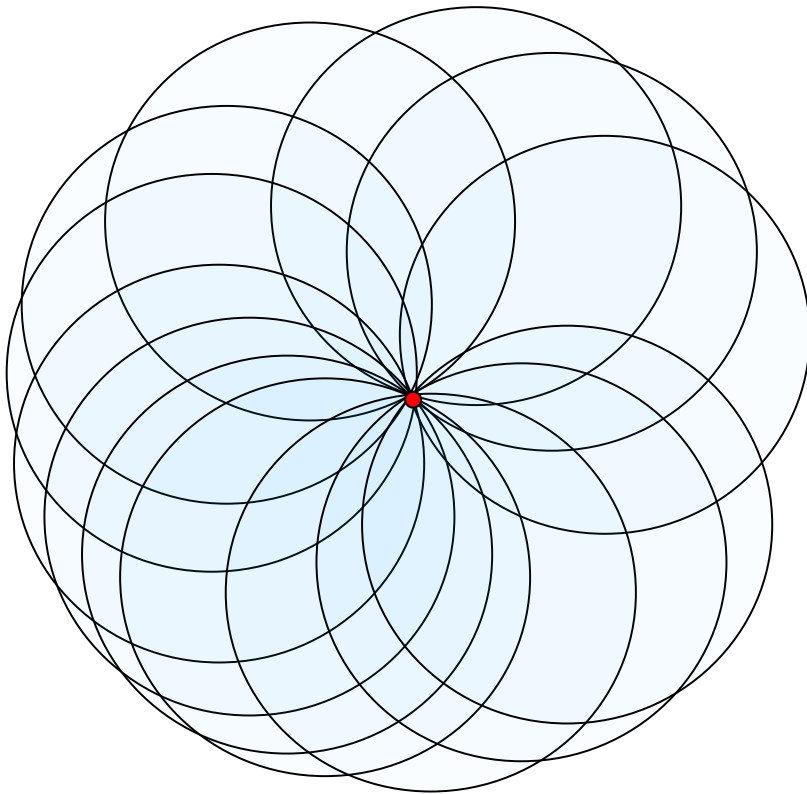
$$x_0 = x - r \cos \theta$$

$$y_0 = y - r \sin \theta$$



In the image space...

With no orientation, each token (point) votes for all possible circles.
With orientation, each token can vote for a smaller number of circles.

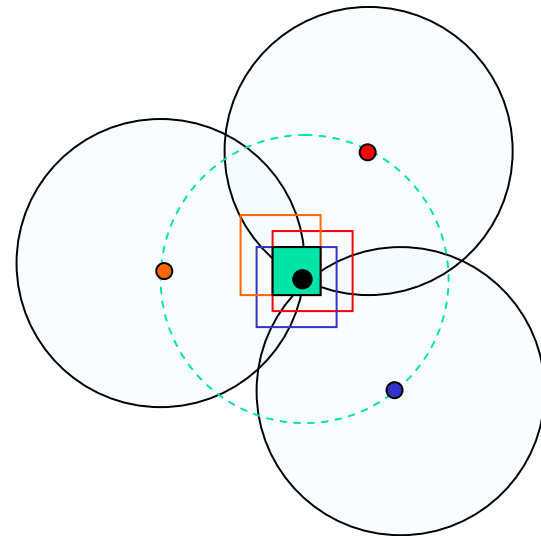
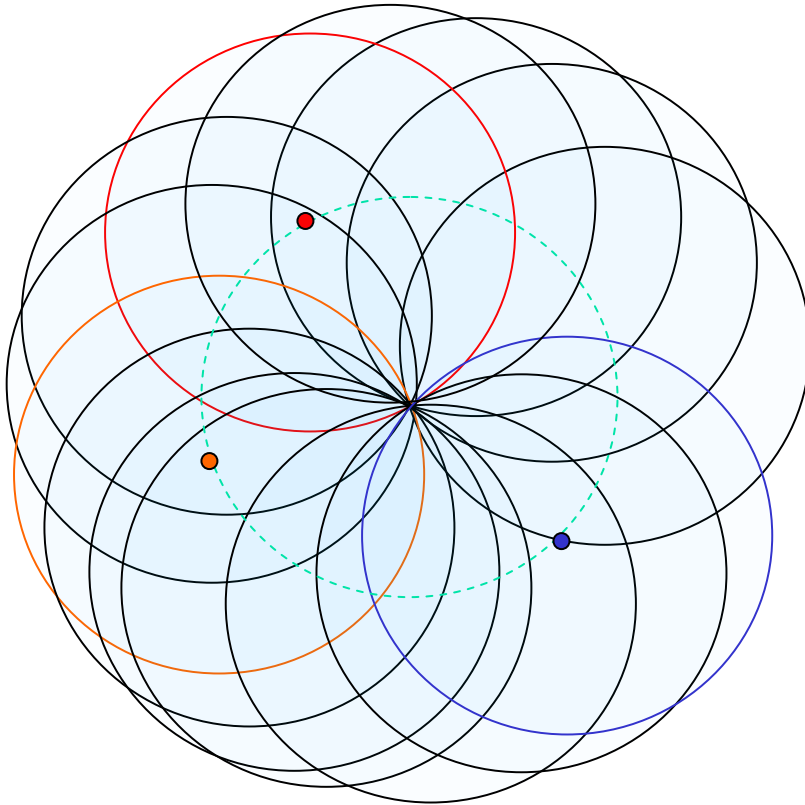




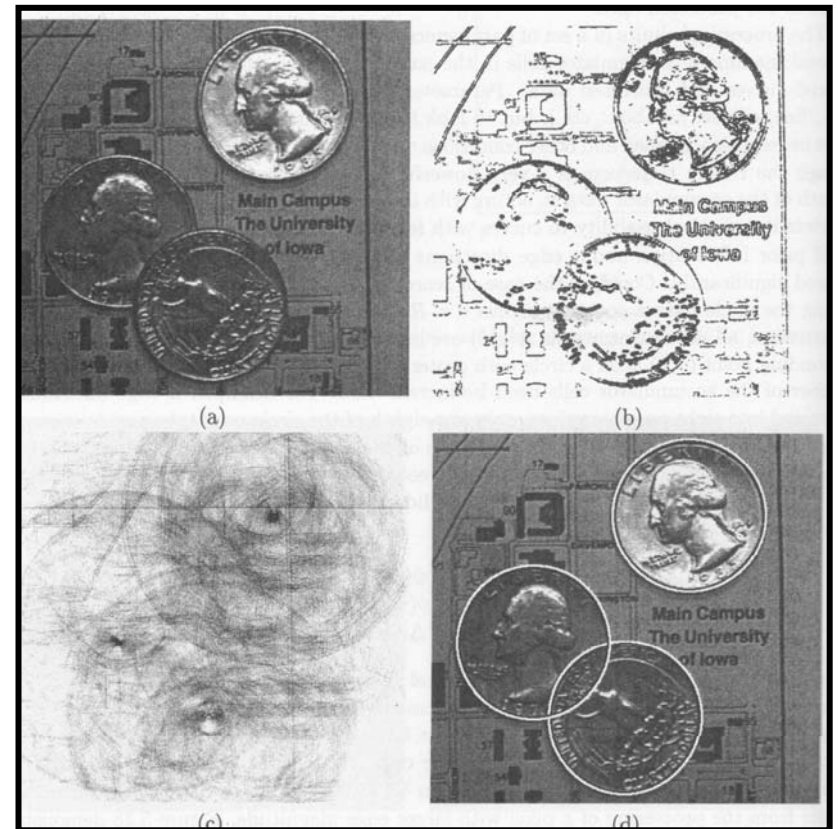
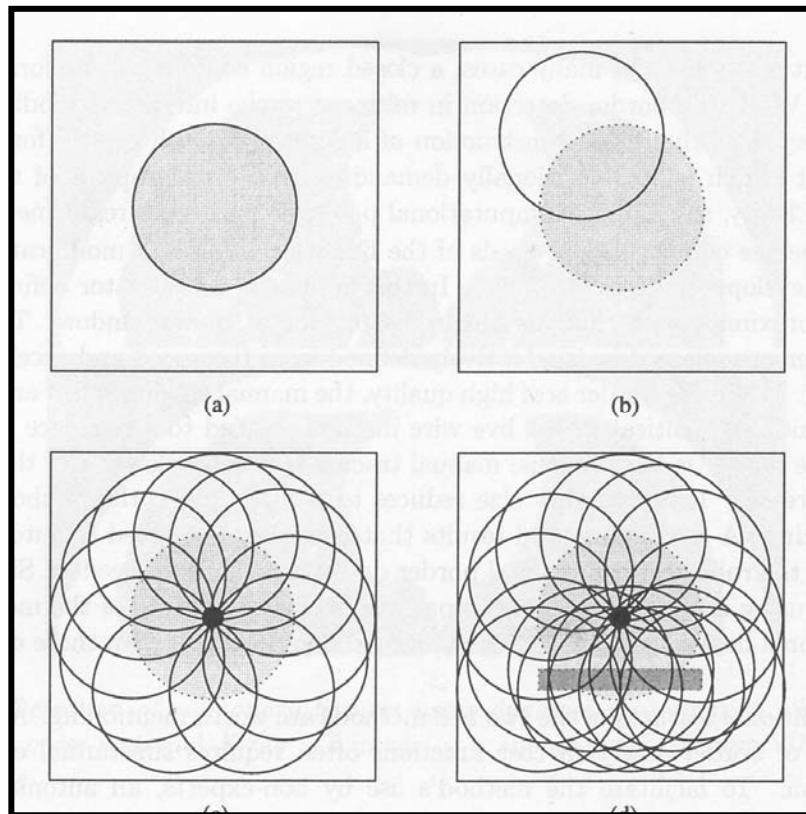
In Hough space...

With no orientation, each token (point) votes for one full circle.

With orientation, each token can vote for a smaller part of a circle.



Examples



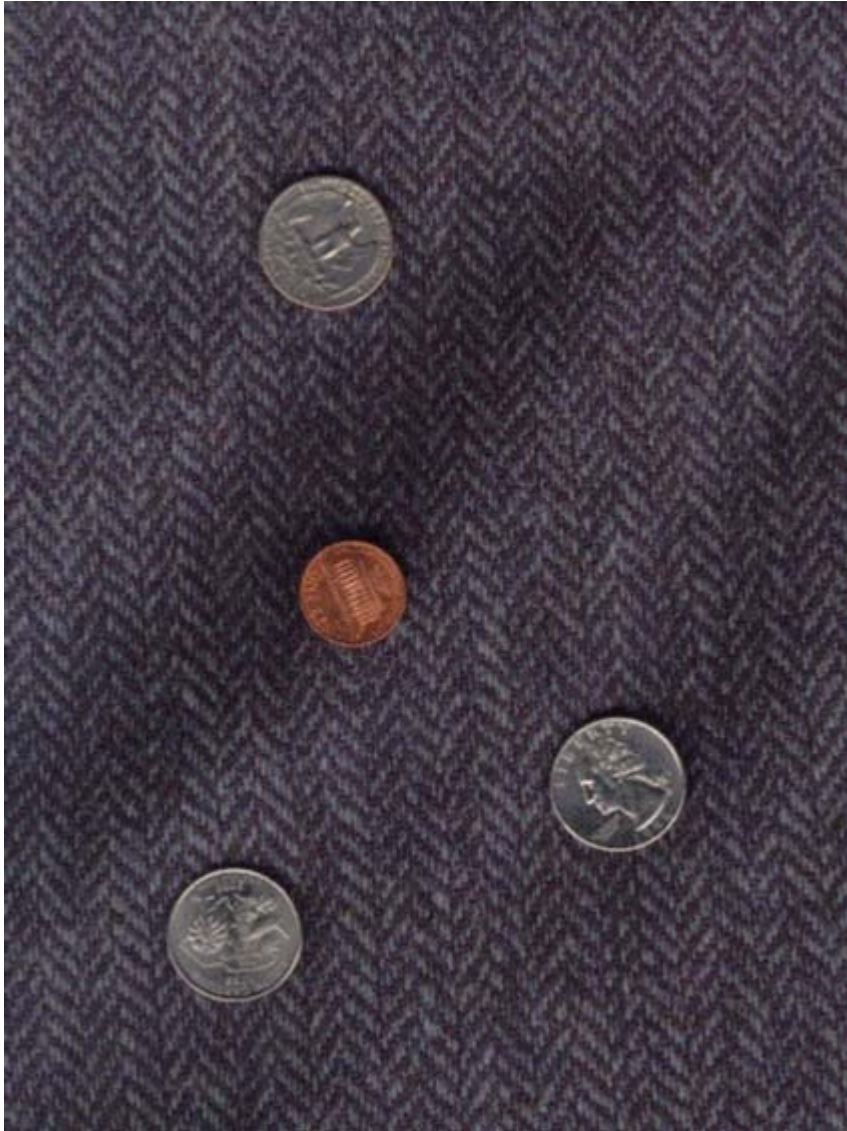
Real World Circle Examples



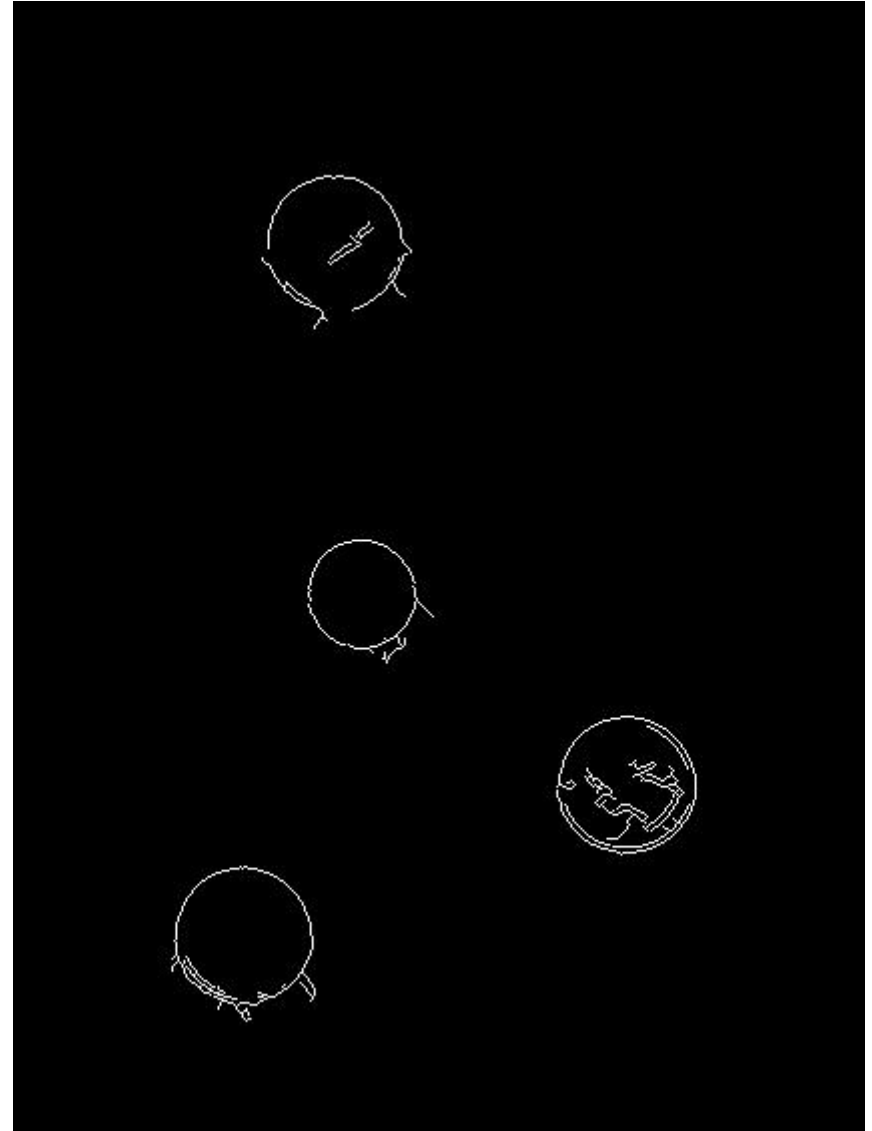
Crosshair indicates results of Hough transform, bounding box found via motion differencing.

Finding Coins

Original



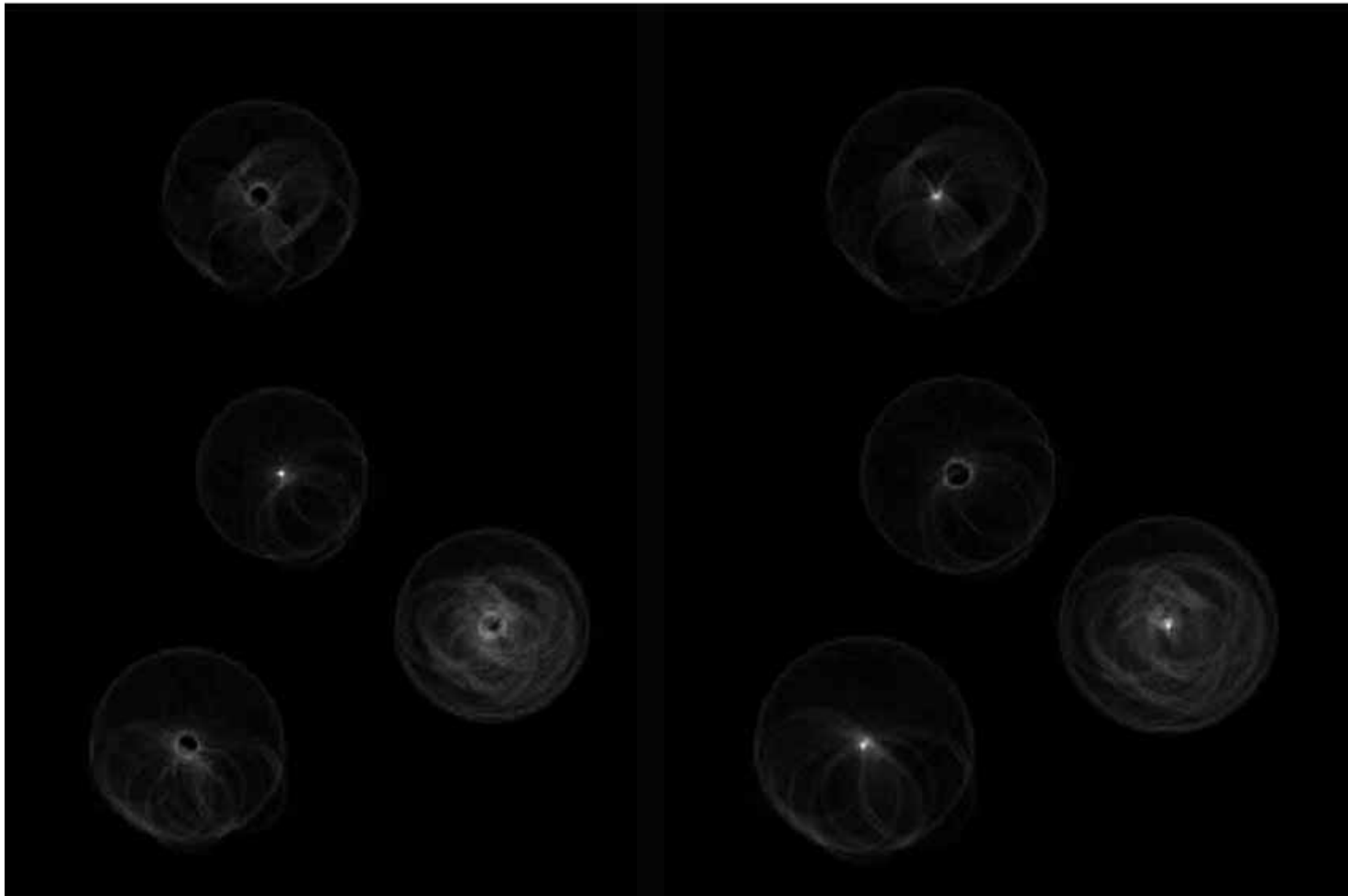
Edges (note noise)



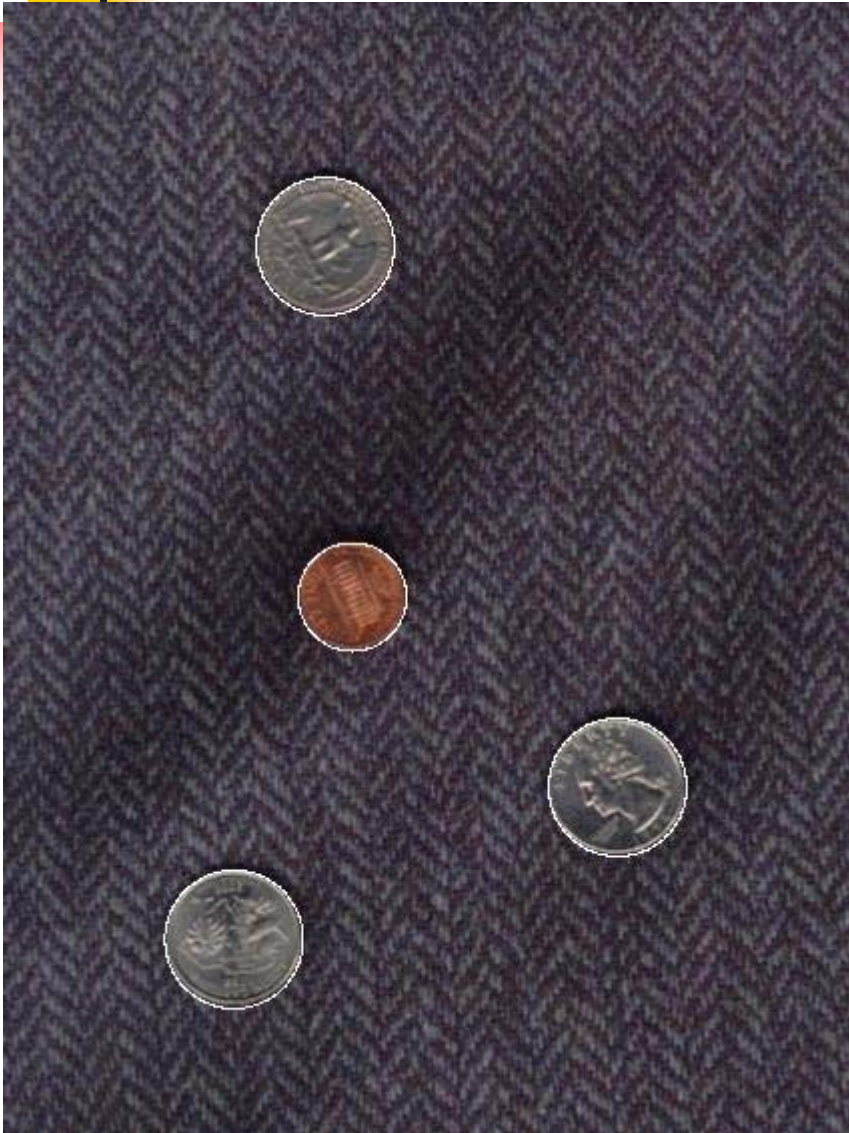
Finding Coins (Continued)

Penny

Quarters



Finding Coins (Continued)



Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

Coin finding sample images from: Vivik Kwatra



Conclusion

- Finding lines and other parameterized objects is an important task for computer vision.
- The (generalized) Hough transform can detect arbitrary shapes from (edge detected) tokens.
- Success rate depends directly upon the noise in the edge image.
- Downsides: Can be slow, especially for objects in arbitrary scales and orientations (extra parameters increase accumulator space exponentially).



Extensions

- Extension 1: Use the image gradient
 1. same
 2. for each edge point $I[x,y]$ in the image
compute unique (d, θ) based on image gradient at (x,y)
$$H[d, \theta] += 1$$
 3. same
 4. same
- 1. What's the running time measured in votes?



Extensions

- Extension 2
 - give more votes for stronger edges
- Extension 3
 - change the sampling of (d, θ) to give more/less resolution
- Extension 4
 - the same procedure can be used with circles, squares, or any other shape



Summary Hough Transform

- Smart counting
 - Local evidence for global features
 - Organized in a table
 - Careful with parameterization!
- Subject to Curse of Dimensionality
 - Works great for simple features with 3 unknowns
 - Will fail for complex objects (e.g., all faces)



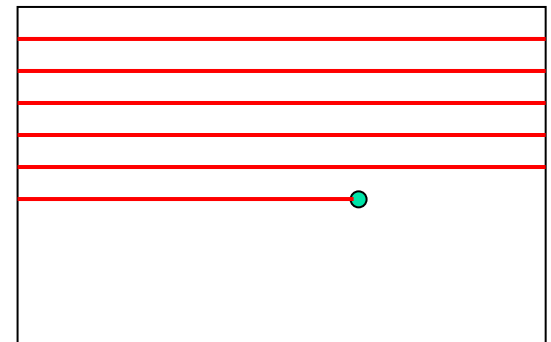
Curse of Dimensionality

- Goal: looking for some target in a given space

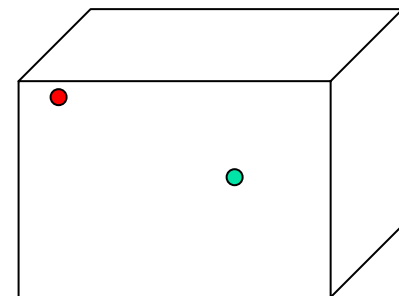
- Case 1: 1D space



- Case 2: 2D space



- Case 3: 3D space





Optional Assignments: Curve fitting

- Goal:
 - Line fitting
 - Conic fitting
 - Other curve fitting
- Techniques:
 - Least squares fit
 - Hough transform
 - Robust implementation
 -



Contents

- Curve Fitting
- Hough Transform
- Robust fitting



Outliers

- Least squares assumes Gaussian errors
- Outliers: Points with extremely low probability of occurrence (according to Gaussian statistic)
 - Can result from data association errors
- Strongly influence least squares



Robust estimation

- Goal: develop parameter estimation methods insensitive to small numbers of large errors
- General approach: try to give large deviations less weight
- M-estimators: minimize some function other than $(y - f(x, a, b, \dots))^2$



Least absolute value fitting

- Minimize $E = \sum_i |y_i - f(x_i, a, b, \dots)|$

instead of $E = \sum_i [y_i - f(x_i, a, b, \dots)]^2$

- Points far away from trend get comparatively less influence



Example: constant

- For constant function $y=a$
 - Minimizing squares gives $a=\textit{mean}$
 - Minimizing absolute value give $a=\textit{median}$



Example: constant – Proof

- For constant function $y=a$
 - Minimizing squares gives $a=mean$

$$E = \sum_i (y_i - a)^2$$

$$\frac{\partial E}{\partial a} = \sum_i -2(y_i - a) = -2(\sum_i y_i - n \cdot a) = 0$$

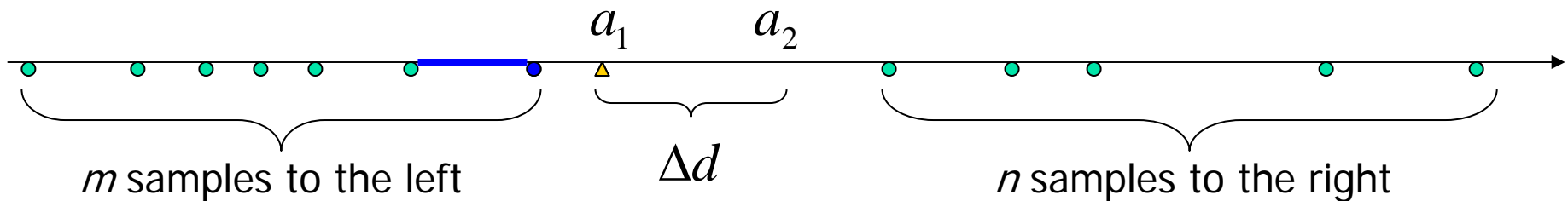
$$a = \sum_i y_i / n = mean(Y)$$



Example: constant – Proof

- For constant function $y=a$
 - Minimizing absolute value gives $a=\text{median}$

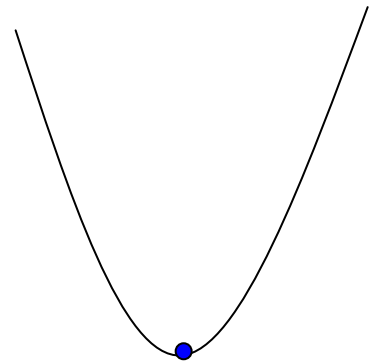
$$E = \sum_i |y_i - a|$$



$$\Delta E = E_2 - E_1 = \Delta d \cdot m - \Delta d \cdot n = \Delta d(m - n)$$

Minimum is reached when $m=n$

- Sample number is *odd*
- Sample number is *even*





Who came from which line?

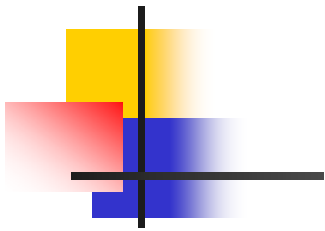
- Assume we know how many lines there are - but which lines are they?
 - easy, if we know who came from which line
- Three strategies
 - Incremental line fitting
 - K-means
 - Probabilistic (later!)



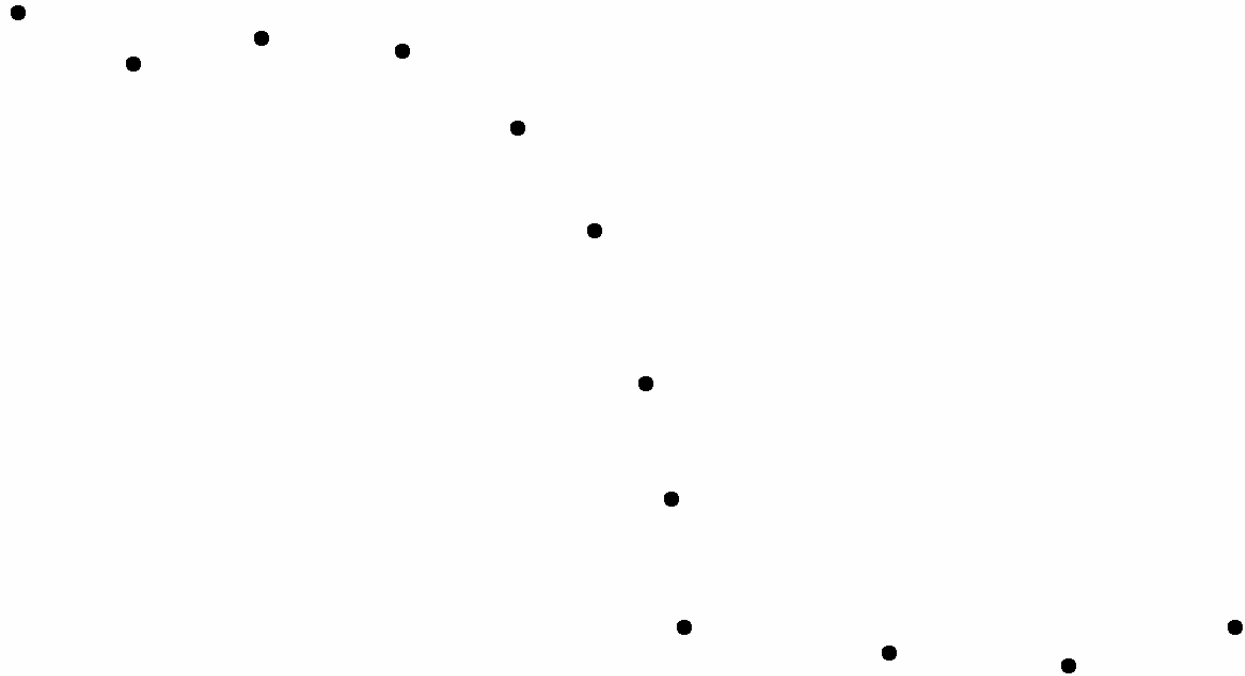
Incremental line fitting

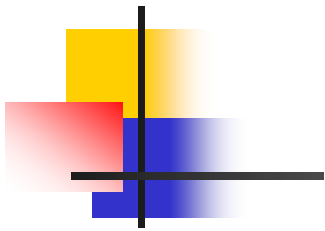
Algorithm 15.1: Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large

```
Put all points on curve list, in order along the curve
Empty the line point list
Empty the line list
Until there are too few points on the curve
    Transfer first few points on the curve to the line point list
    Fit line to line point list
    While fitted line is good enough
        Transfer the next point on the curve
            to the line point list and refit the line
    end
    Transfer last point(s) back to curve
    Refit line
    Attach line to line list
end
```

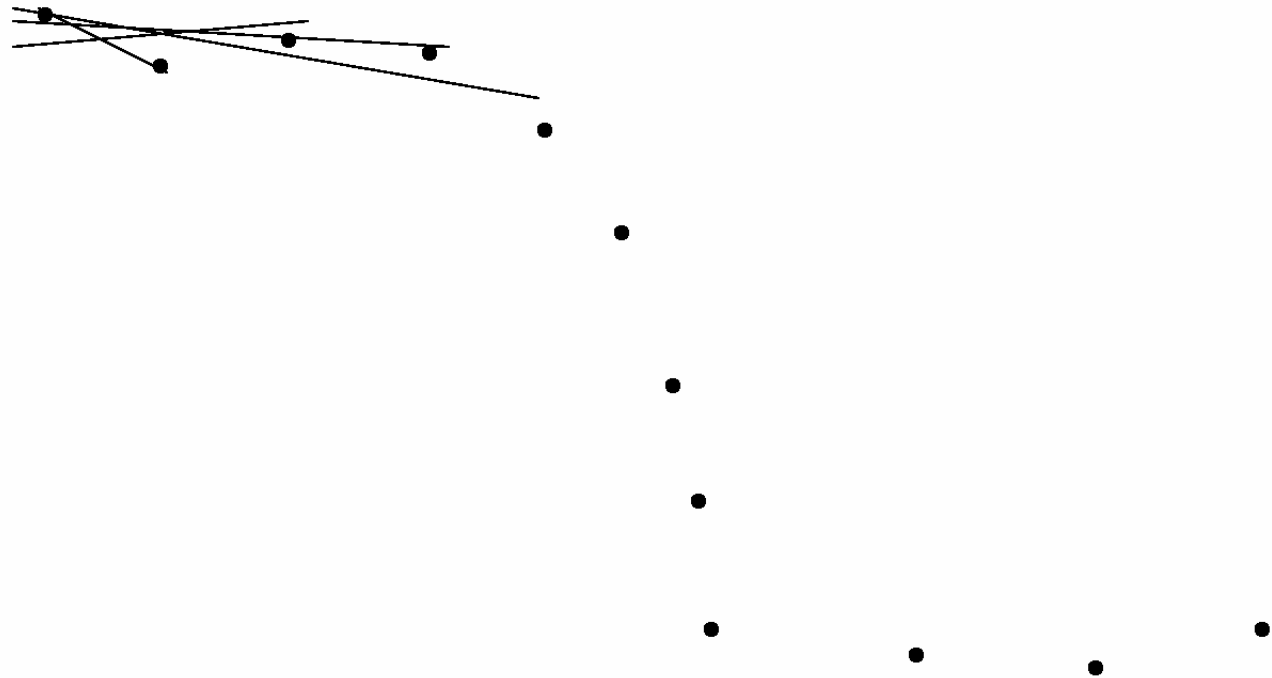


Incremental line fitting

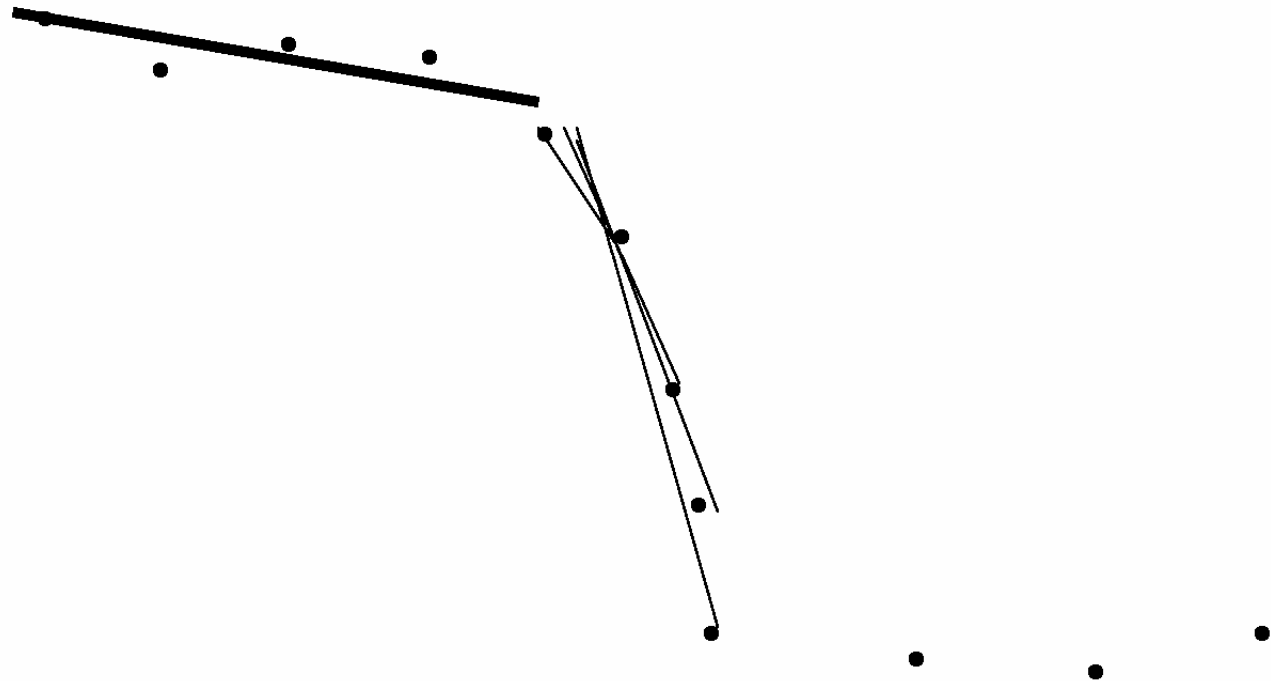




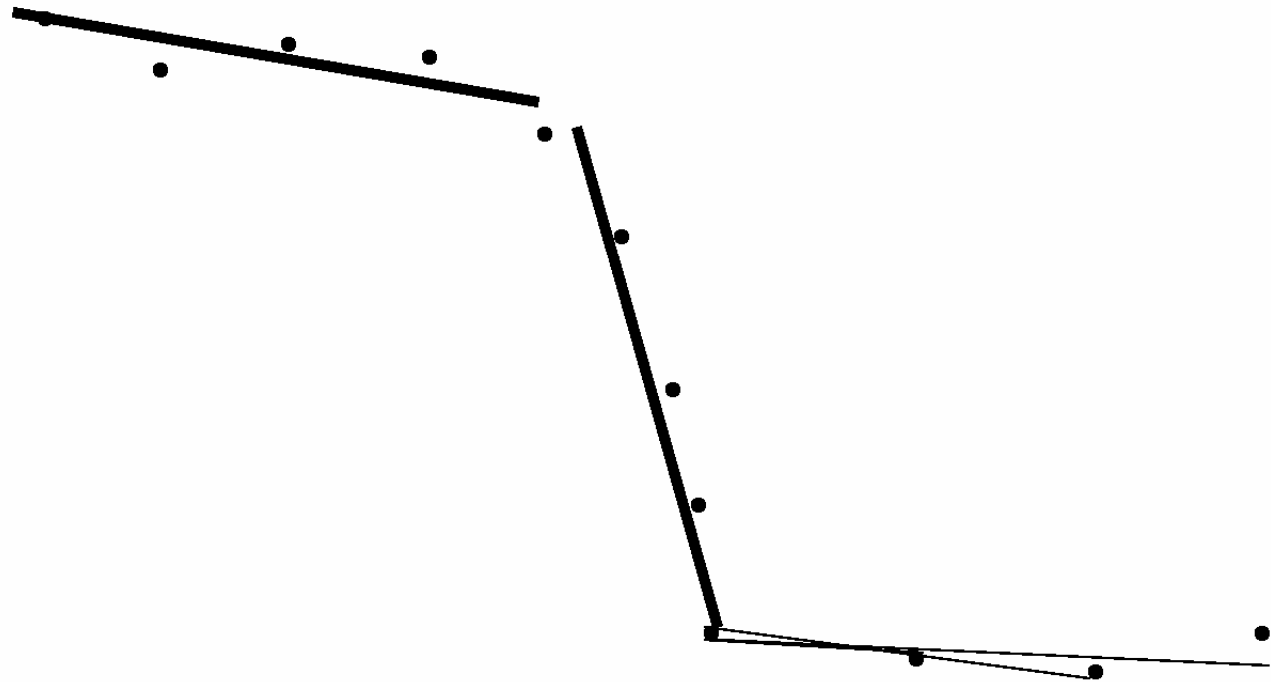
Incremental line fitting



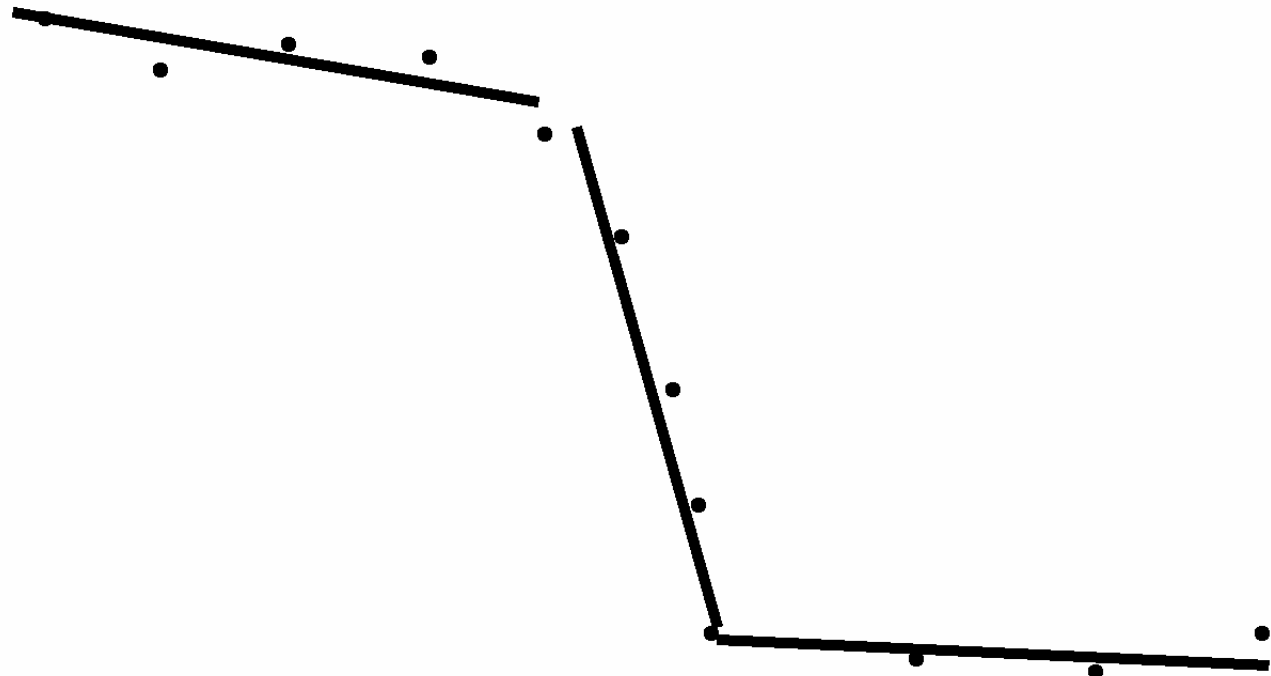
Incremental line fitting



Incremental line fitting



Incremental line fitting





K-means line fitting

Algorithm 15.2: K-means line fitting by allocating points to the closest line and then refitting.

Hypothesize k lines (perhaps uniformly at random)

or

Hypothesize an assignment of lines to points
and then fit lines using this assignment

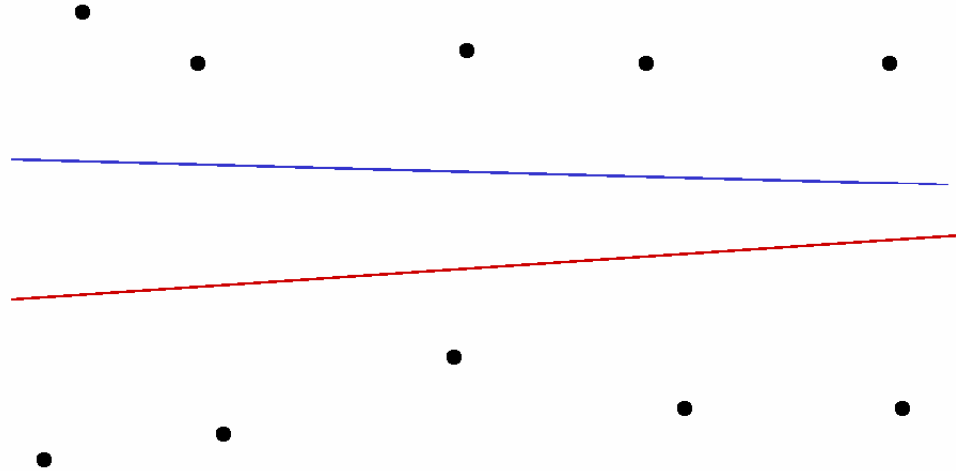
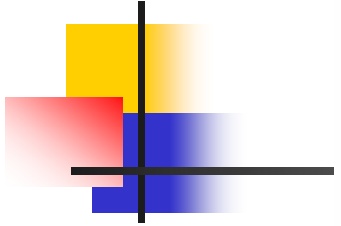
Until convergence

 Allocate each point to the closest line

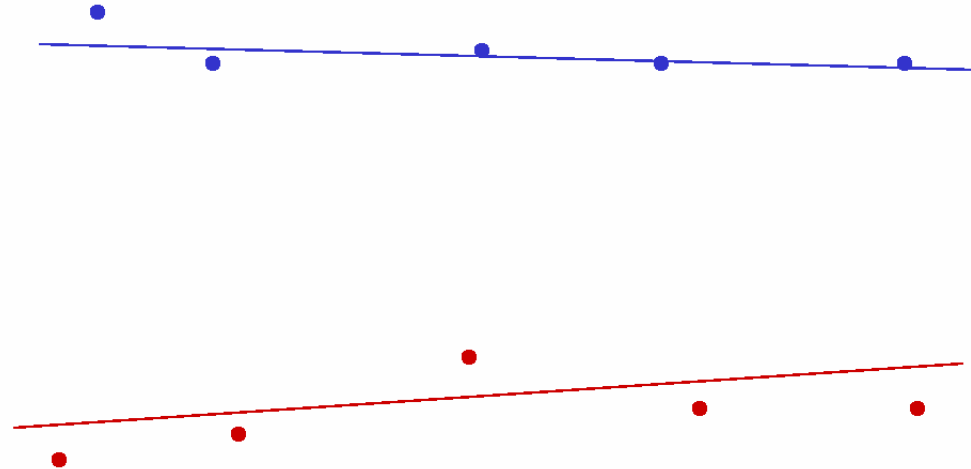
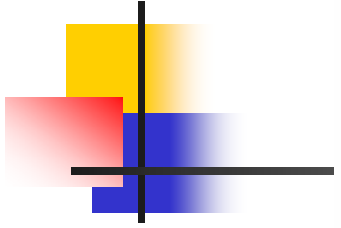
 Refit lines

end

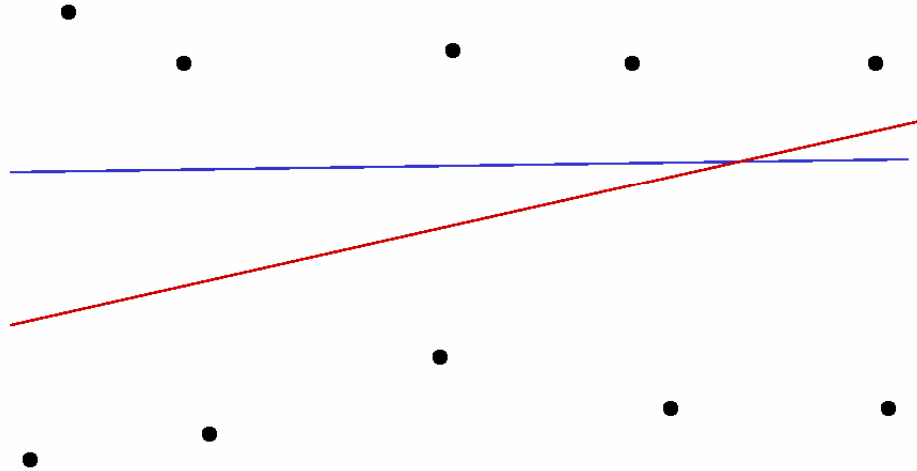
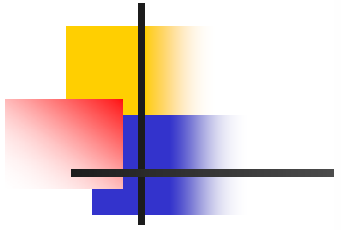
K-means line fitting



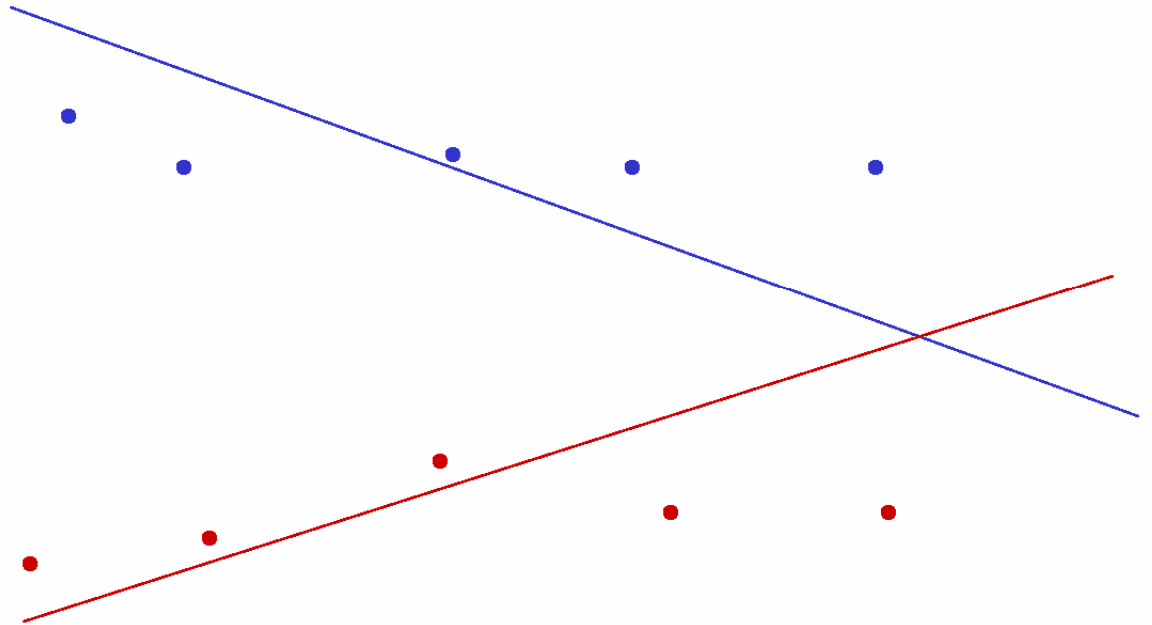
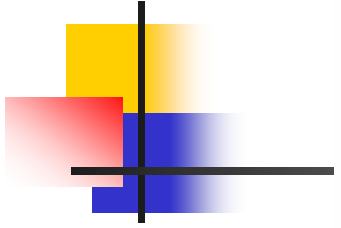
K-means line fitting



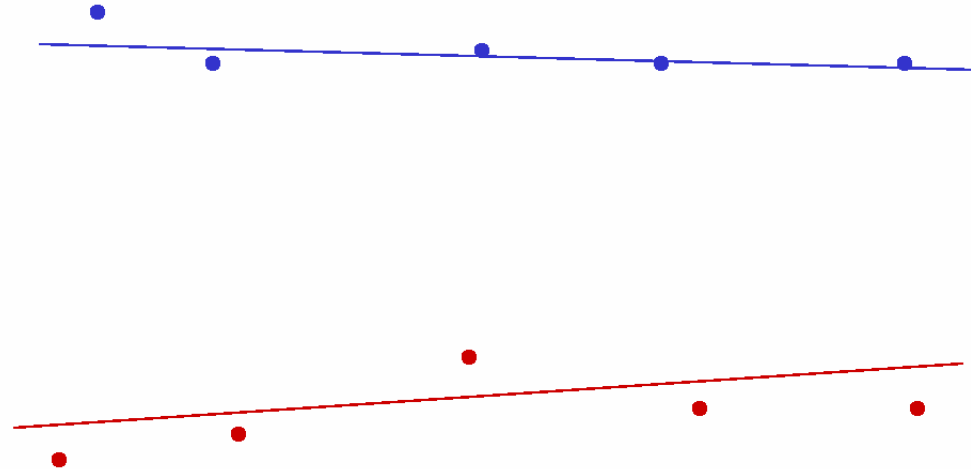
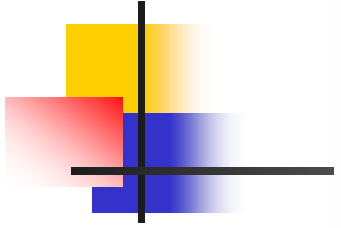
K-means line fitting



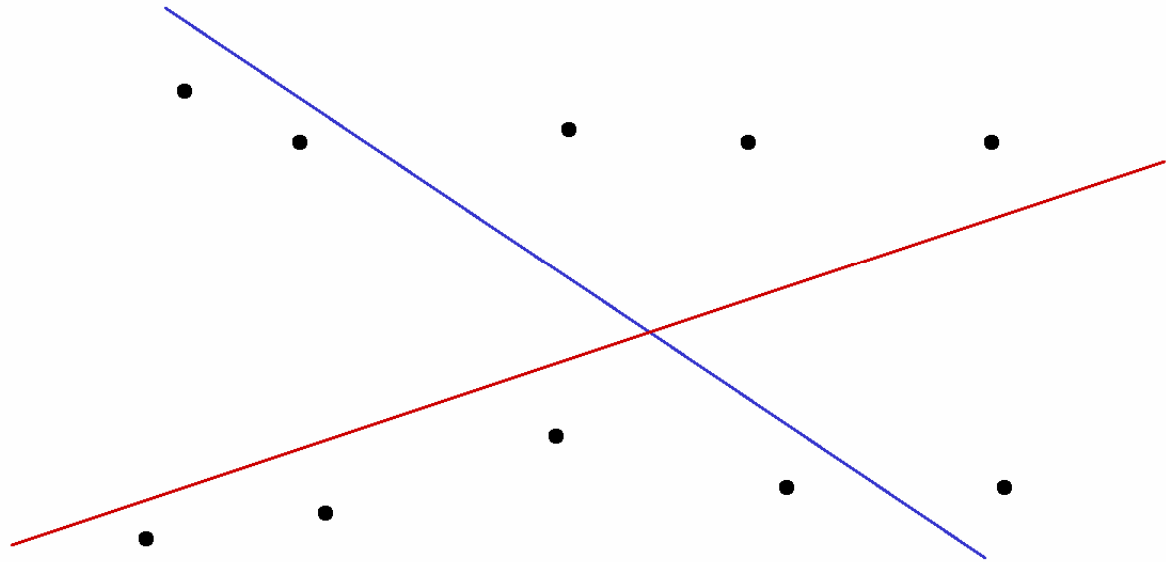
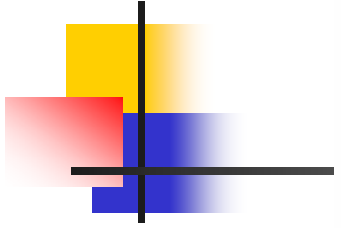
K-means line fitting



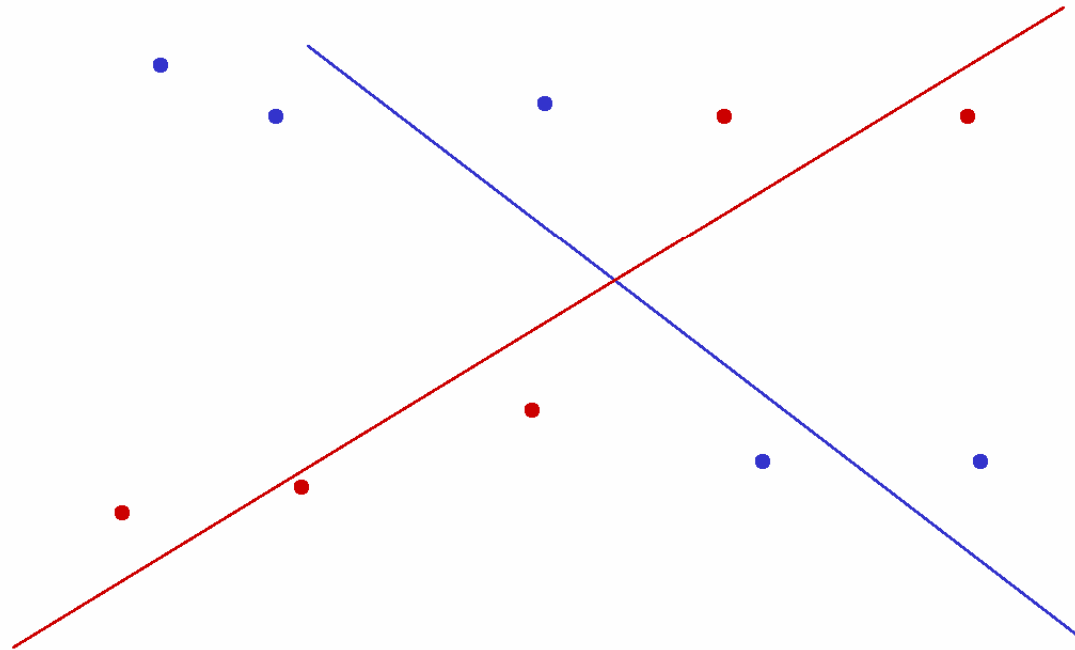
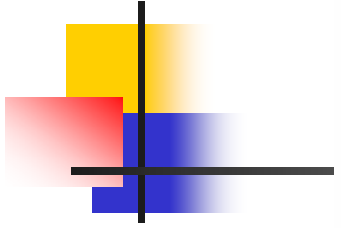
K-means line fitting



K-means line fitting



K-means line fitting

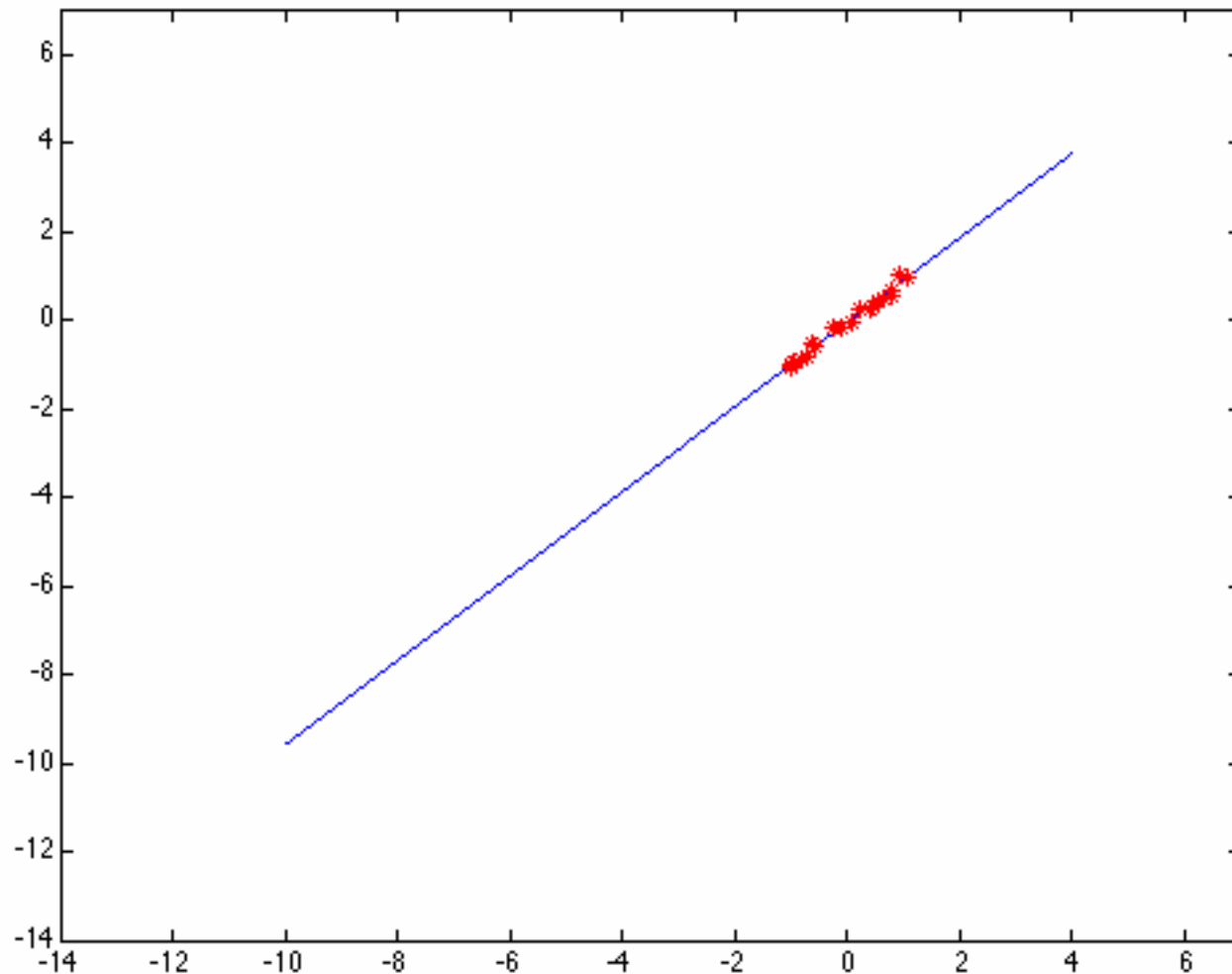




Robustness

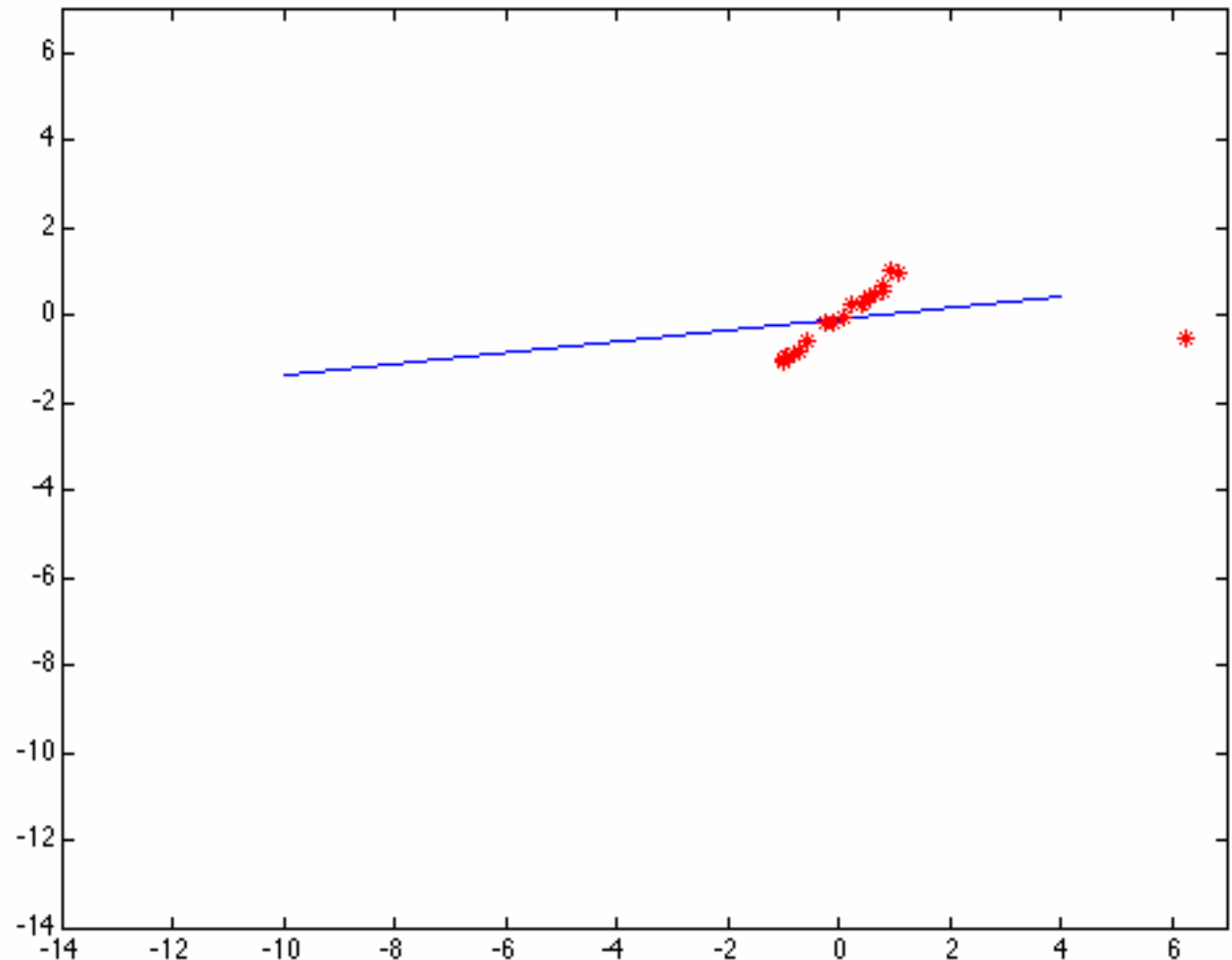
- As we have seen, squared error can be a source of bias in the presence of noise points
 - One fix is EM - we'll do this shortly
 - Another is an M-estimator
 - Square nearby, threshold far away
 - A third is RANSAC
 - Search for good points

Least squares fit— no noise

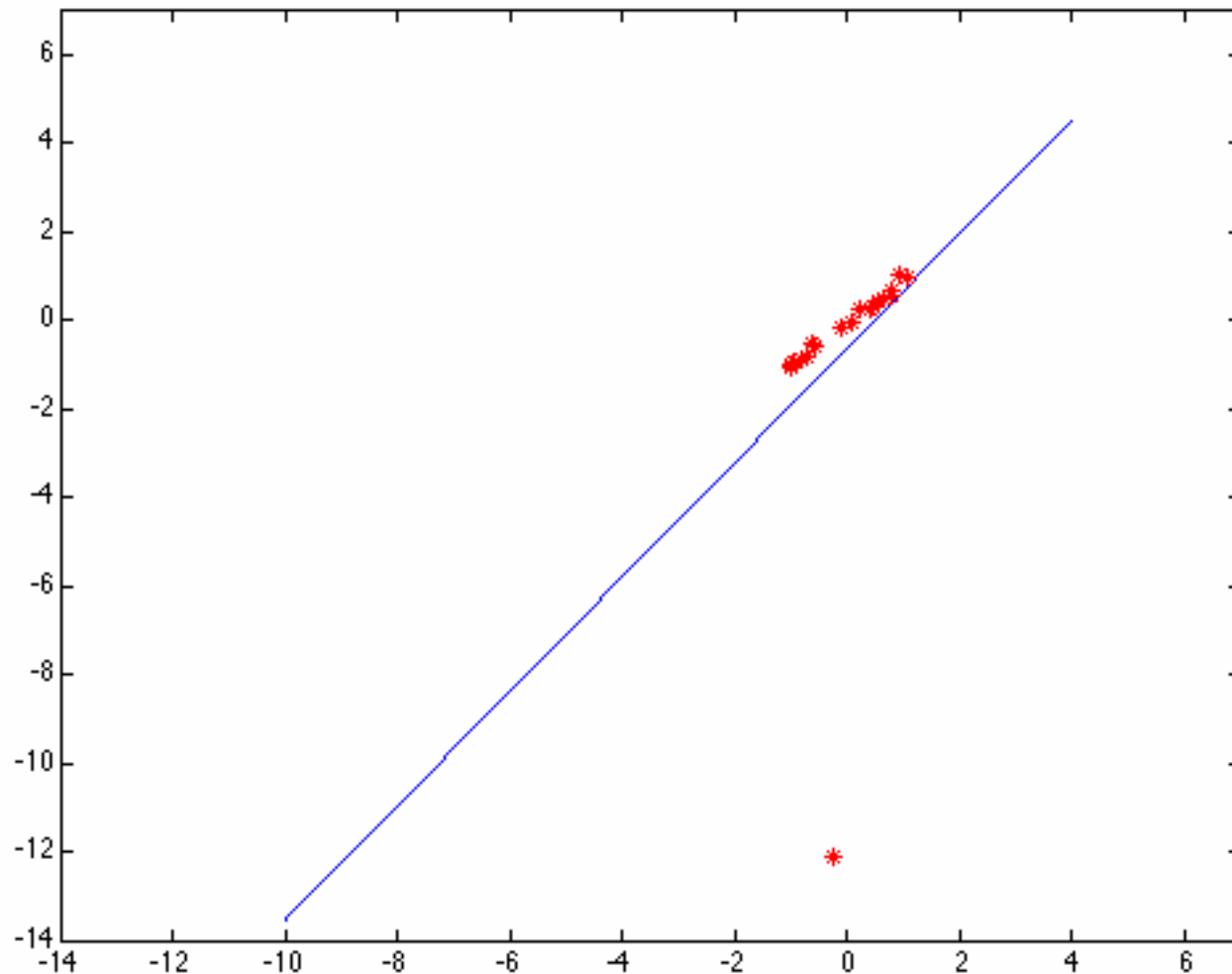


Least squares fit – with noise

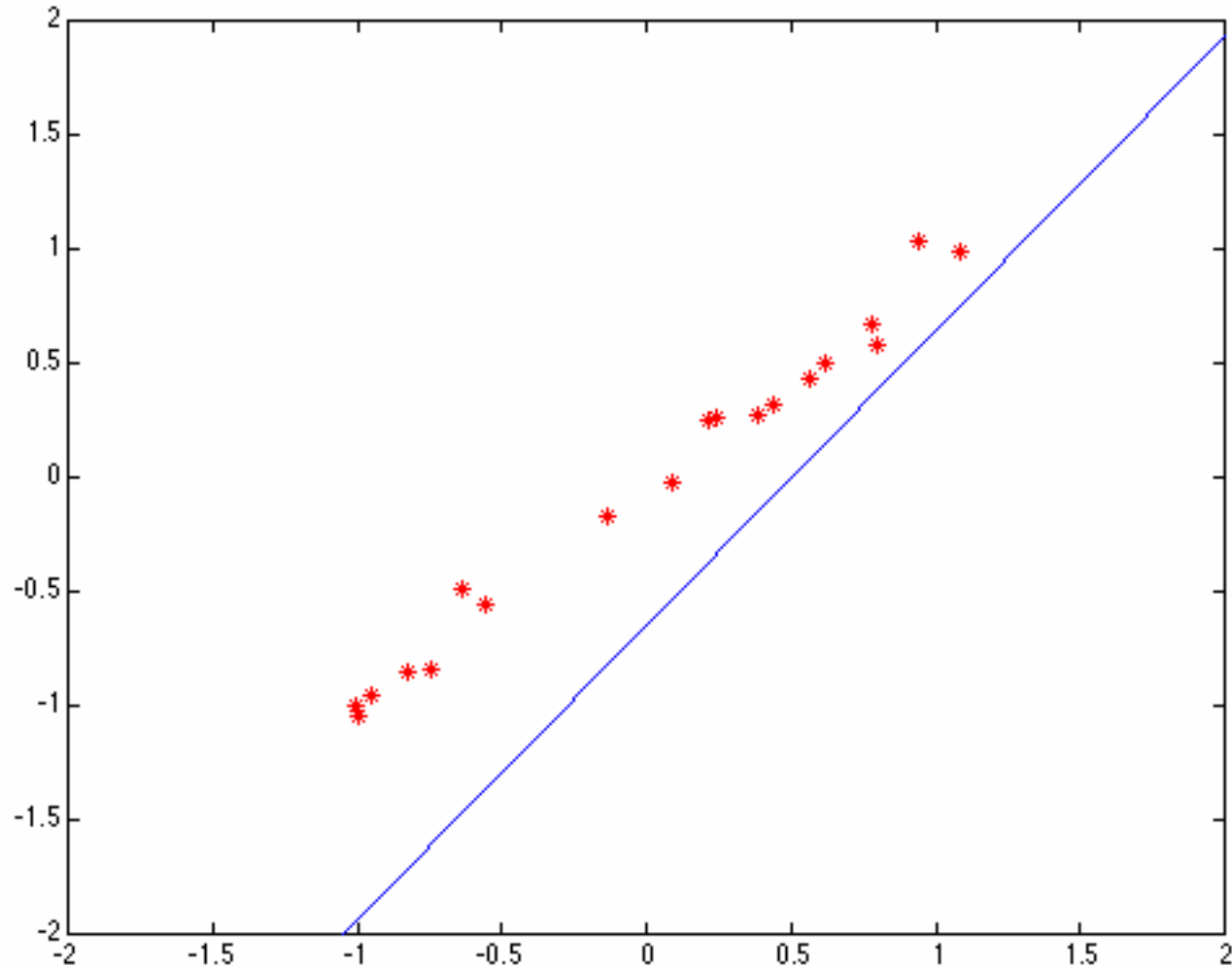
The problem is the single point on the right; the error for that point is so large that it drags the line away from the other points (the total error is lower if these points are some way from the line, because then the point on the line is not so dramatically far away).



Least squares fit in the presence of noise – another example



Let us zoom in...



Detail of the previous slide - the line is actually quite far from the points.

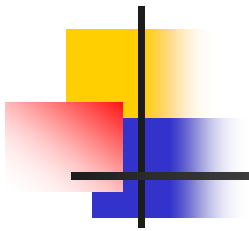


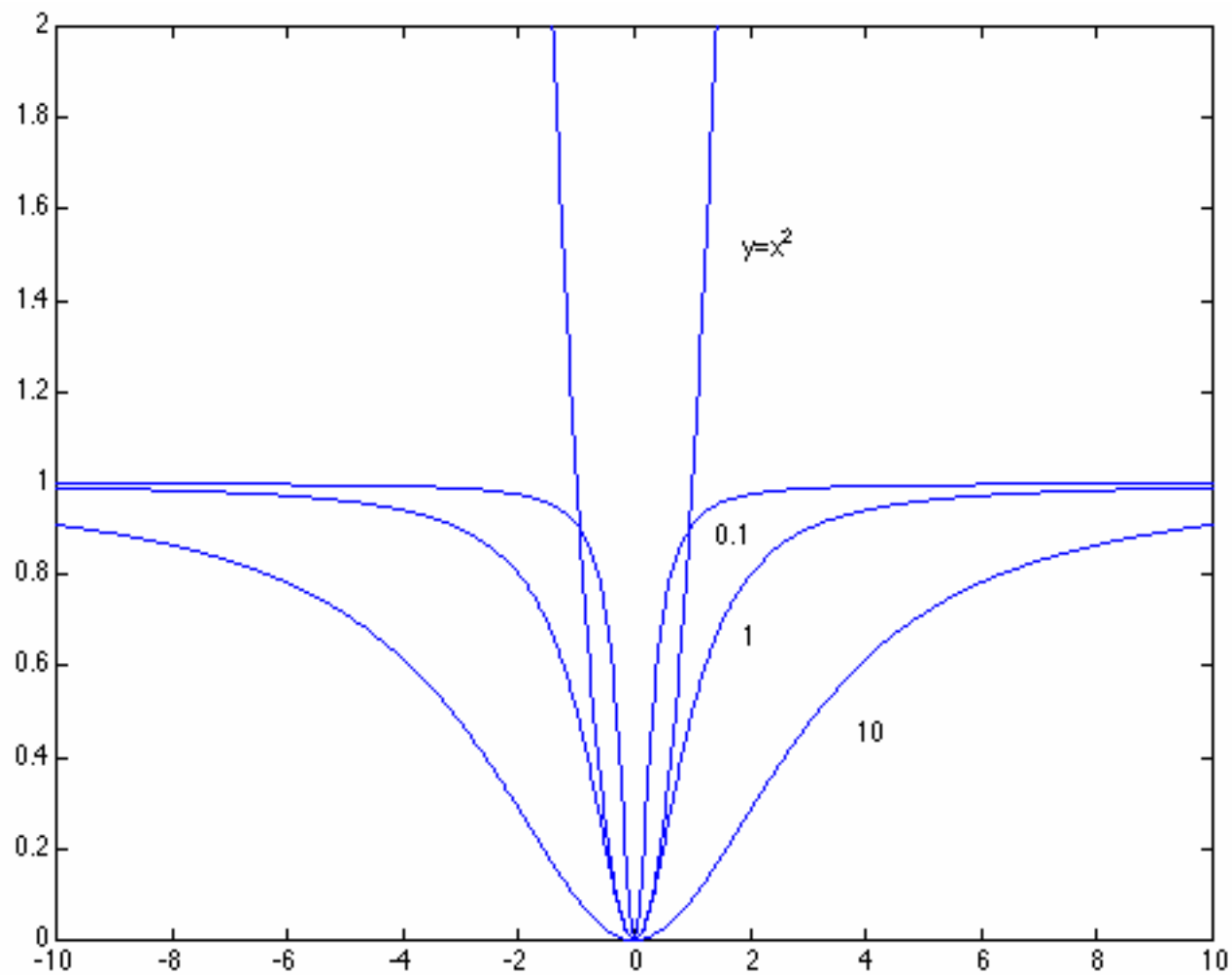
M-estimators

- Generally, minimize

$$\sum_i \rho(r_i(x_i, \theta); \sigma)$$

where $r_i(x_i, \theta)$ is the residual

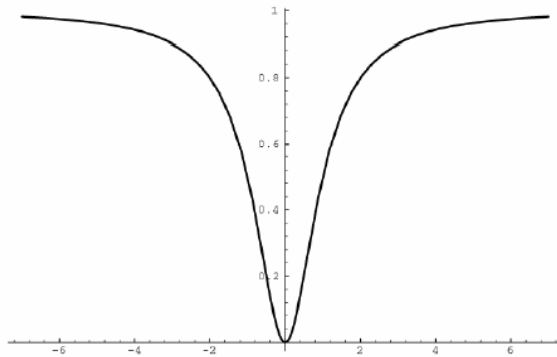

$$\rho(u; \sigma) = \frac{u^2}{\sigma^2 + u^2}$$





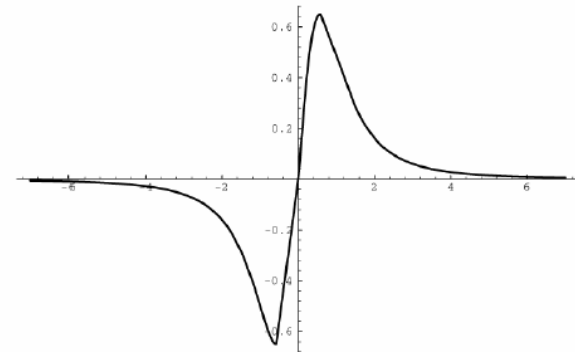
Robust Estimation

A quadratic ρ function gives too much weight to outliers
Instead, use robust norm:



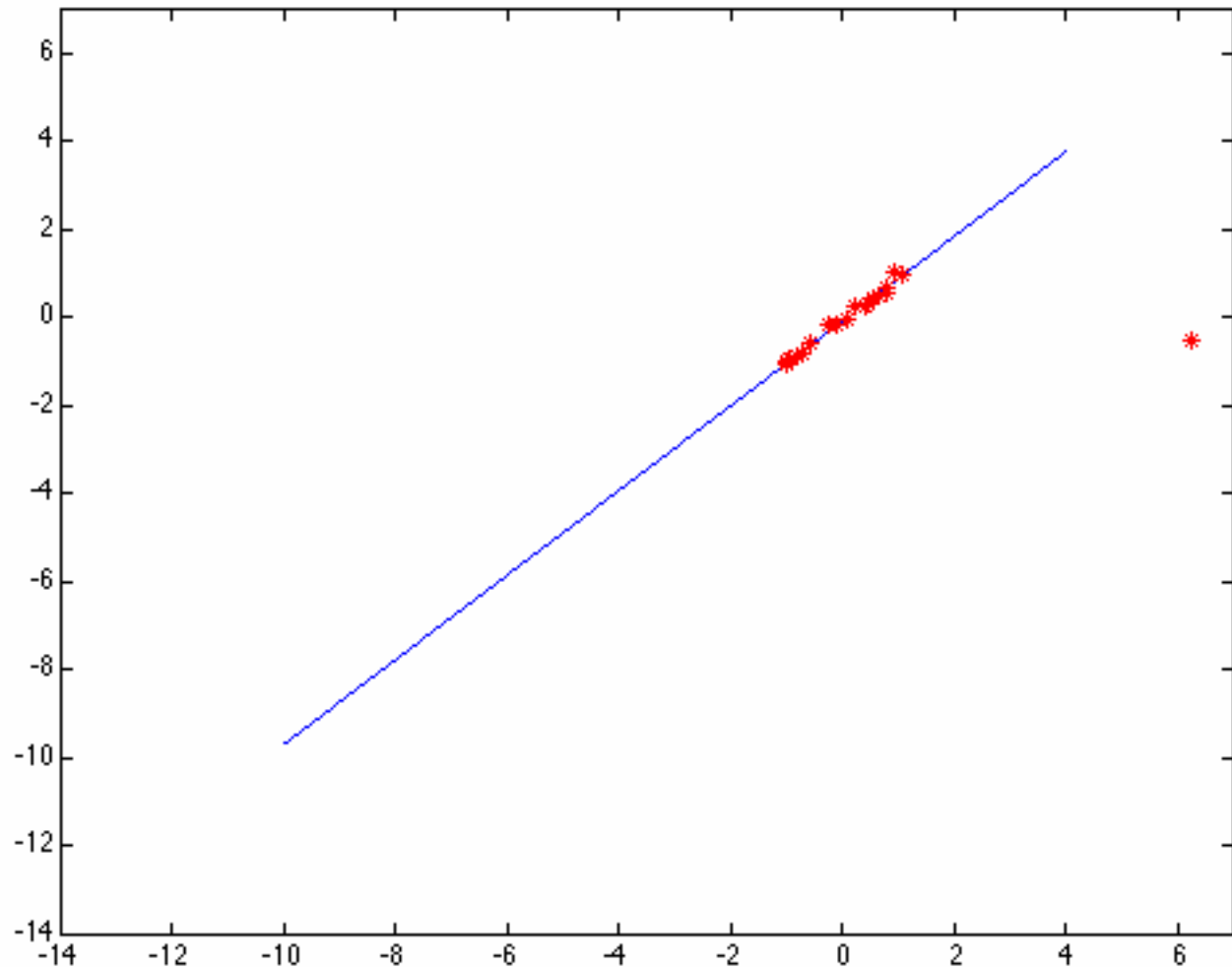
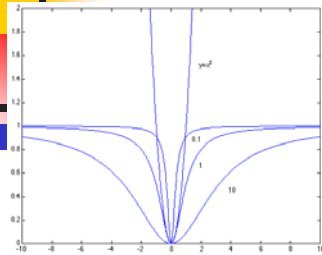
$$\rho(r, \sigma) = \frac{r^2}{\sigma^2 + r^2}$$

Influence function
(d/dr of norm):



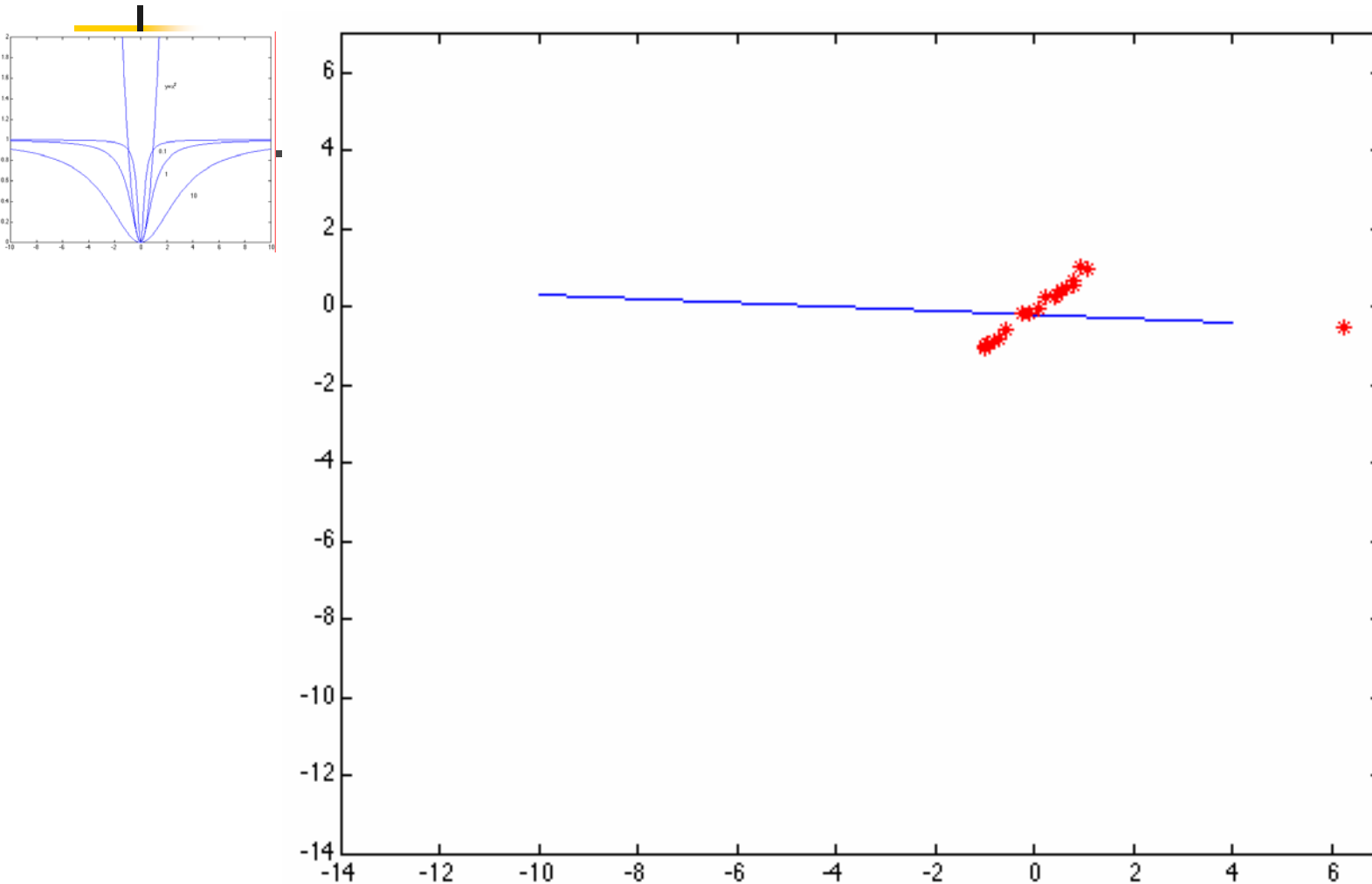
$$\psi(r, \sigma) = \frac{2r\sigma^2}{(\sigma^2 + r^2)^2}$$

Parameter selection



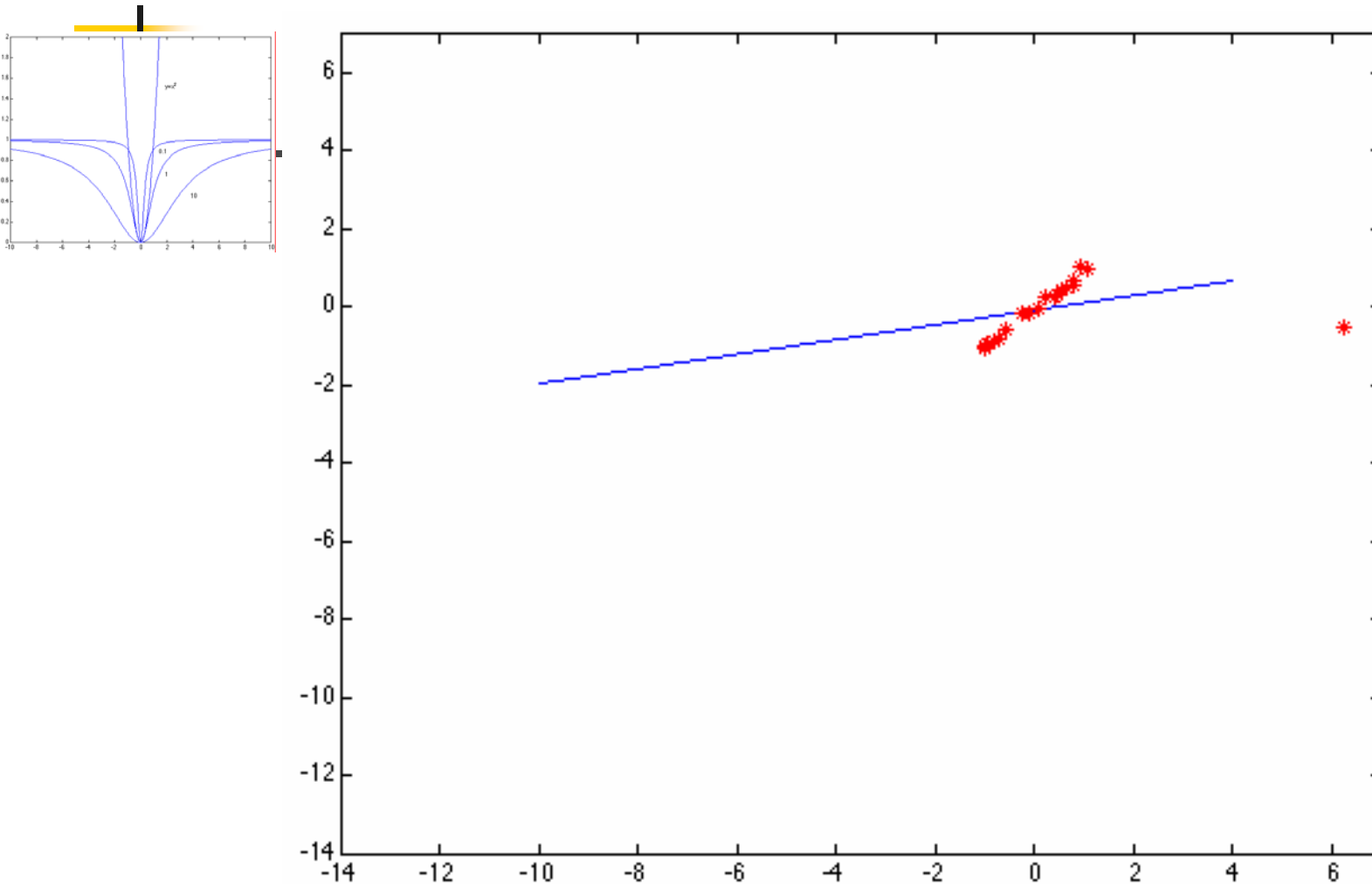
Fit to earlier data with an appropriate choice of s

Parameter selection



Here the parameter is too small, and the error for every point looks like distance^2

Parameter selection



Here the parameter is too large, meaning that the error value is about constant for every point, and there is very little relationship between the line and the points.



RANSAC



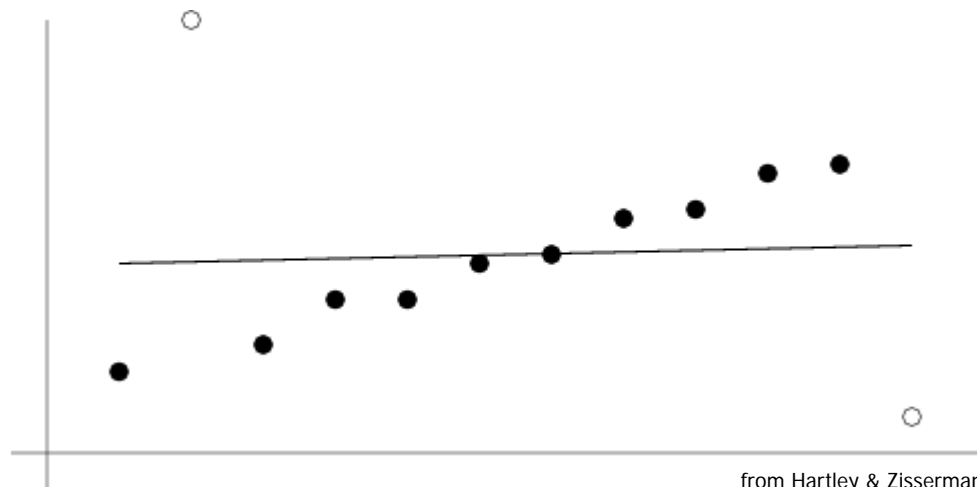
Motivation

- Estimating motion models
- Typically: points in two images
- Candidates:
 - Translation
 - Homography
 - Fundamental matrix

The Problem with Outliers

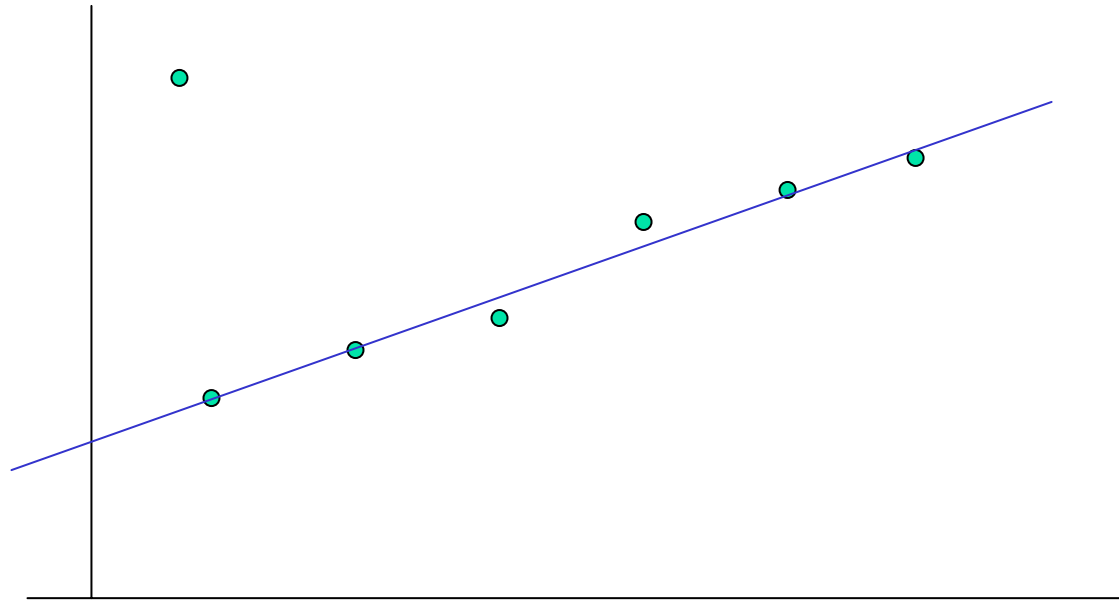
- Least squares is a technique for fitting a model to data that exhibit a Gaussian error distribution
- When there are *outliers*—data points that are not drawn from the same distribution—the estimation result can be biased

Line fitting using
regression is
biased by outliers



Simpler Example

- Fitting a straight line





Robust Estimation

- View estimation as a two-stage process:
 - Classify data points as outliers or inliers
 - Fit model to inliers



Discard Outliers

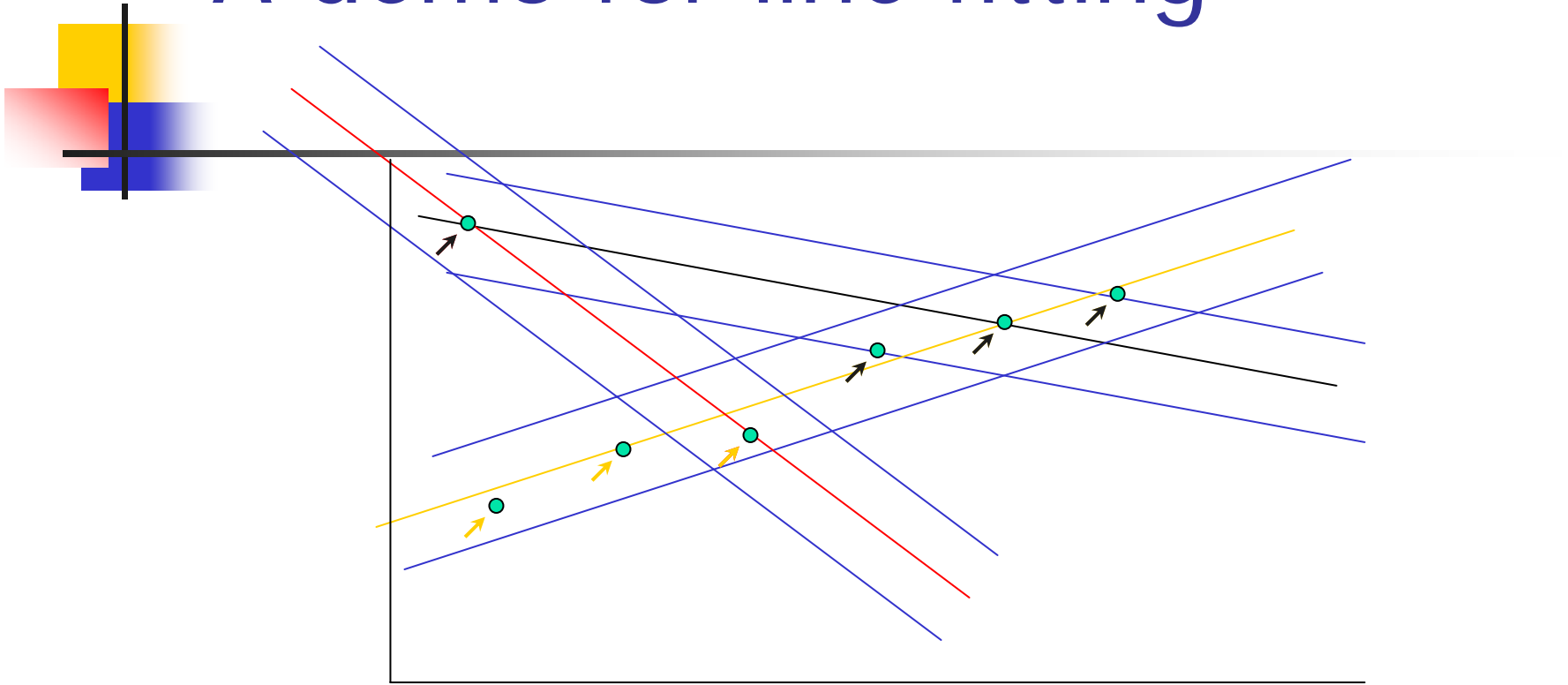
- No point with $d > t$
- RANSAC:
 - RANdom SAmple Consensus
 - Fischler & Bolles 1981
 - Copes with a large proportion of outliers



RANSAC

- Main idea
 - Select 2 points at random
 - Fit a line
 - “Support” = number of inliers
 - Line with most supports wins
- General algorithm
 - Randomly select s points
 - Instantiate a model
 - Get consensus set S_i (supports)
 - Repeat for N trials, return model with $\max |S_i|$

A demo for line fitting



Trial 1: Support number=2

Trial 2: Support number=6

Trial 3: Support number=4

Trial 2 Wins with maximal support number !



RANSAC

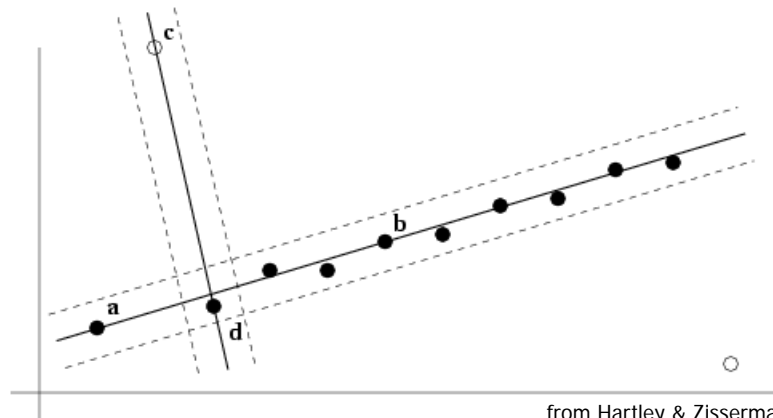
- Why will this work?
 - More support -> better fit
 - Best Line has most support
- Fit a more general model
 - Robust fit of a model to data S
- **RAN**dom **SA**mples **C**onsensus: designed for bad data (in best case, up to 50% outliers)

RANSAC

(RANDOM Sample Consensus)

1. Randomly choose minimal subset of data points necessary to fit model (a *sample*)
2. Points within some distance threshold t of model are a *consensus set*. Size of consensus set is model's *support*
3. Repeat for N samples; model with biggest support is most robust fit
 - Points within distance t of best model are inliers
 - Fit final model to all inliers

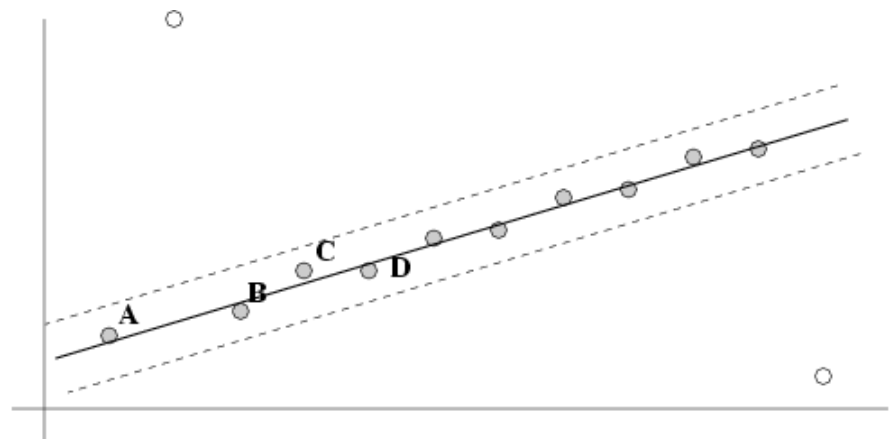
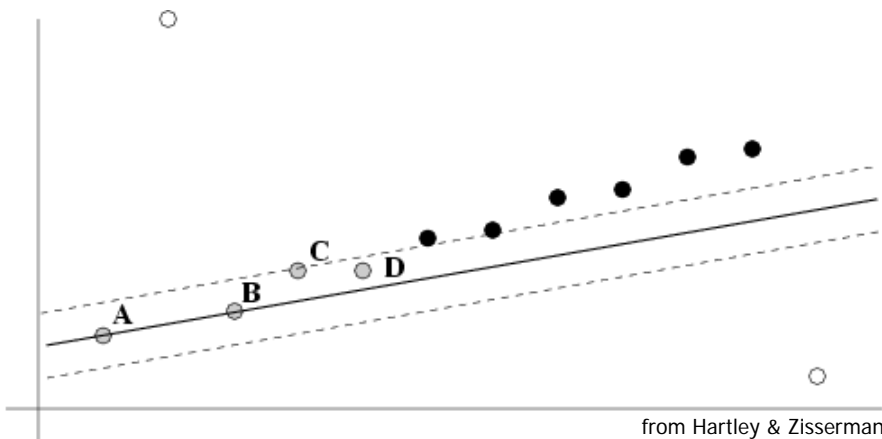
Two samples
and their supports
for line-fitting



After RANSAC

RANSAC divides data into inliers and outliers and yields estimate computed from minimal set of inliers with greatest support

- Improve this initial estimate with estimation over all inliers (i.e., standard minimization)
- But this may change inliers, so alternate fitting with re-classification as inlier/outlier





RANSAC

■ Issues

- How many times?
 - Often enough that we are likely to have a good line
- How big a subset?
 - Smallest possible
- What does close mean?
 - Depends on the problem
- What is a good line?
 - The number of nearby points is so big it is unlikely to be all outliers



RANSAC: How many samples?

- We want: at least one trial with a sample of all inliers
- Can't guarantee: use probability
- Using all possible samples is often infeasible
- Instead, pick **N** so that, with probability **p** of at least one sample is free from outliers (or being all inliers)

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

where e is probability that point is an outlier

- Typically **p = 0.99**



Calculate N

- If w = proportion of inliers = $1-e$
- $P(\text{sample with all inliers}) = w^s$
- $P(\text{sample with an outlier}) = 1-w^s$
- $P(N \text{ samples an outlier}) = (1-w^s)^N$
- We want $P(N \text{ samples an outlier}) < 1-p$
- $(1-w^s)^N < 1-p$
- $N > \log(1-p) / \log(1-w^s)$

$w=1-e$:

$$N > \log(1-p) / \log(1-(1-e)^s)$$

RANSAC: Computed N ($p = 0.99$)

Sample size	Proportion of outliers ²						
S	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

adapted from Hartley & Zisserman



Example

- $P=0.99$
- $s=2, e=5\%$ $\Rightarrow N=2$
- $s=2, e=50\%$ $\Rightarrow N=17$
- $s=4, e=5\%$ $\Rightarrow N=3$
- $s=4, e=50\%$ $\Rightarrow N=72$
- $s=8, e=5\%$ $\Rightarrow N=5$
- $s=8, e=50\%$ $\Rightarrow N=1177$

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:

- n — the smallest number of points required
- k — the number of iterations required
- t — the threshold used to identify a point that fits well
- d — the number of nearby points required
to assert a model fits well

Until k iterations have occurred

Draw a sample of n points from the data
uniformly and at random

Fit to that set of n points

For each data point outside the sample

Test the distance from the point to the line
against t ; if the distance from the point to the line
is less than t , the point is close

end

If there are d or more points close to the line
then there is a good fit. Refit the line using all
these points.

end

Use the best fit from this collection, using the
fitting error as a criterion



Distance threshold

Choose t so probability for inlier is α (e.g. 0.95)

- Often empirically
- Zero-mean Gaussian noise σ then d_{\perp}^2 follows χ_m^2 distribution with m =codimension of model

(dimension+codimension=dimension space)

Codimension	Model	t^2
1	line,F	$3.84 \sigma^2$
2	H,P	$5.99 \sigma^2$
3	T	$7.81 \sigma^2$



Adaptively determining the number of samples

e is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield $e=0.2$

- $N = \infty$, $sample_count = 0$
- While $N > sample_count$ repeat
 - Choose a sample and count the number of inliers
 - Set $e = 1 - (\text{number of inliers}) / (\text{total number of points})$
 - Recompute N from e
 - Increment the $sample_count$ by 1
- Terminate

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



Remarks

- $N = f(e)$, not the number of points
- N increases steeply with s



Adaptive N

- When ϵ_{th} is unknown ?
- Start with $\epsilon_{\text{th}}=50\%$, $N=\infty$
- Repeat:
 - Sample s , fit model
 - \rightarrow update ϵ_{th} as $|\text{outliers}|/n$
 - \rightarrow set $N=f(\epsilon_{\text{th}}, s, p)$
- Terminate when N samples seen



RANSAC—Alternative Scheme

- Algorithm
 - Randomly select s points
 - Instantiate a model
 - Get consensus set S_i
 - If $|S_i| > T$, terminate and return model
 - Repeat for N trials, return model with $\max |S_i|$



RANSAC—Alternative Scheme

- About Threshold T
 - Remember: terminate if $|S_i| > T$
 - Rule of thumb: $T \approx \#inliers$
 - So, $T = (1-e)n$



RANSAC—Summary

- Choose a small subset uniformly at random
- Least squares fit to that subset
- Compute the fitting error
- Determine the consensus set
 - comparing each error with the threshold;
 - Anything close to result is inliers;
 - all others are outliers
- Repeat the above steps for many trials
- Choose the fit that agreed with most points
 - Can perform one final LS with all inliers



RANSAC—Discussion

- Advantages:

- General method suited for a wide range of model fitting problems;
- Easy to implement and easy to calculate its failure rate;

- Disadvantages:

- Only handles a moderate percentage of outliers without cost blowing up
 - Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)
- Hough transform can handle high percentage of outliers, but false collisions increases with large bins



More Advanced Example

Automatic Homography H Estimation

- How to get correct correspondences without human intervention?



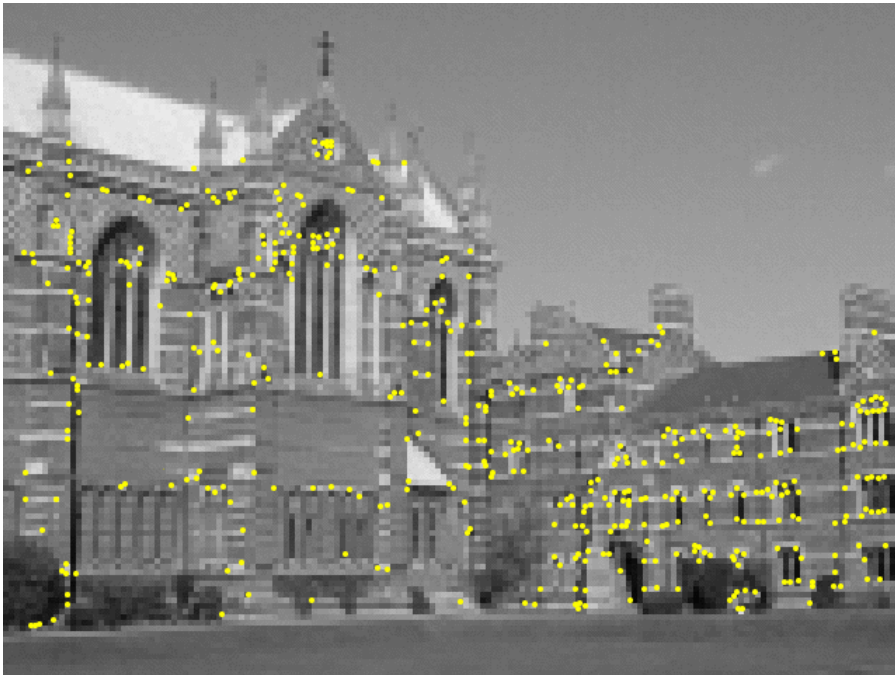
from Hartley & Zisserman



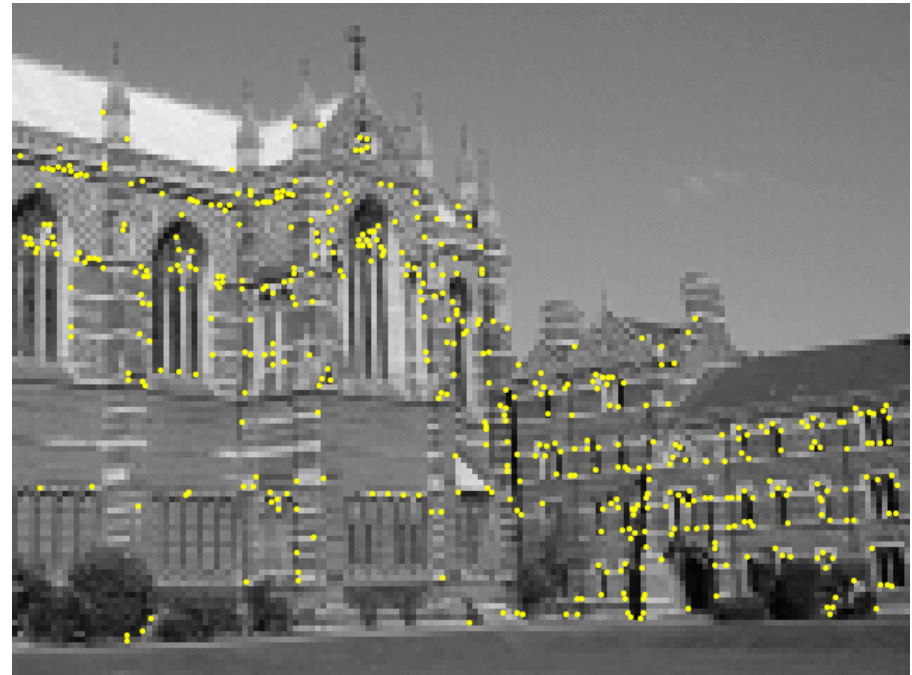
Automatic **H** Estimation: Feature Extraction

- Find features in pair of images using corner detection—e.g., minimum eigenvalue over threshold of:

$$\mathbf{C} = \begin{pmatrix} \sum \mathbf{I}_x^2 & \sum \mathbf{I}_x \mathbf{I}_y \\ \sum \mathbf{I}_x \mathbf{I}_y & \sum \mathbf{I}_y^2 \end{pmatrix}$$



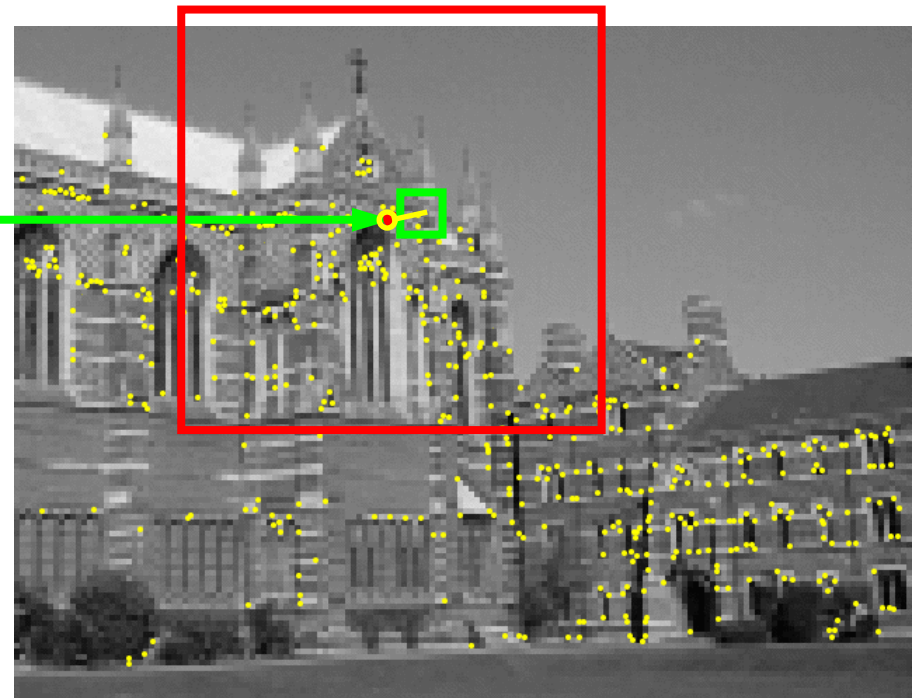
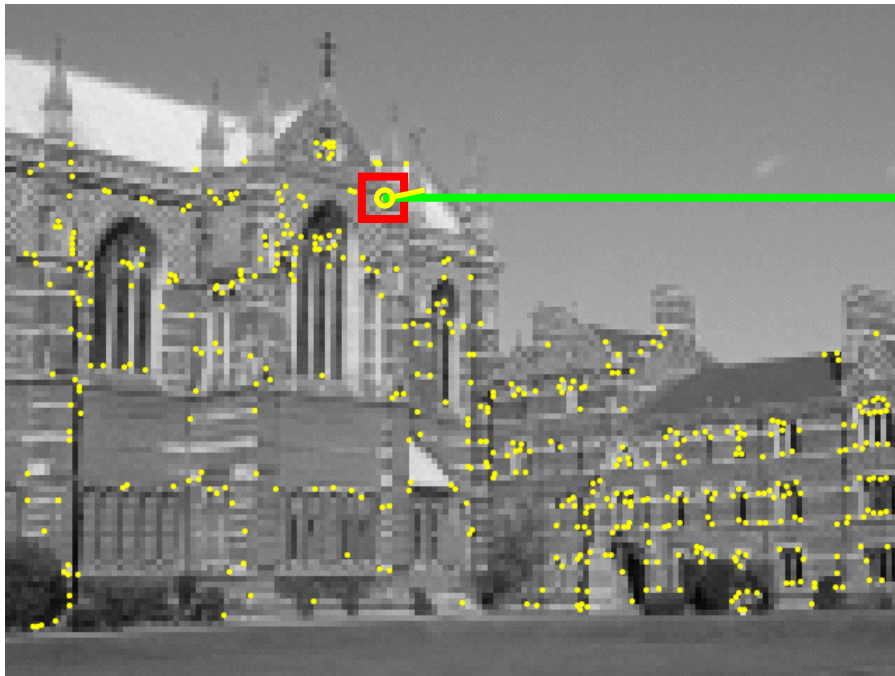
from Hartley & Zisserman



~500 features found

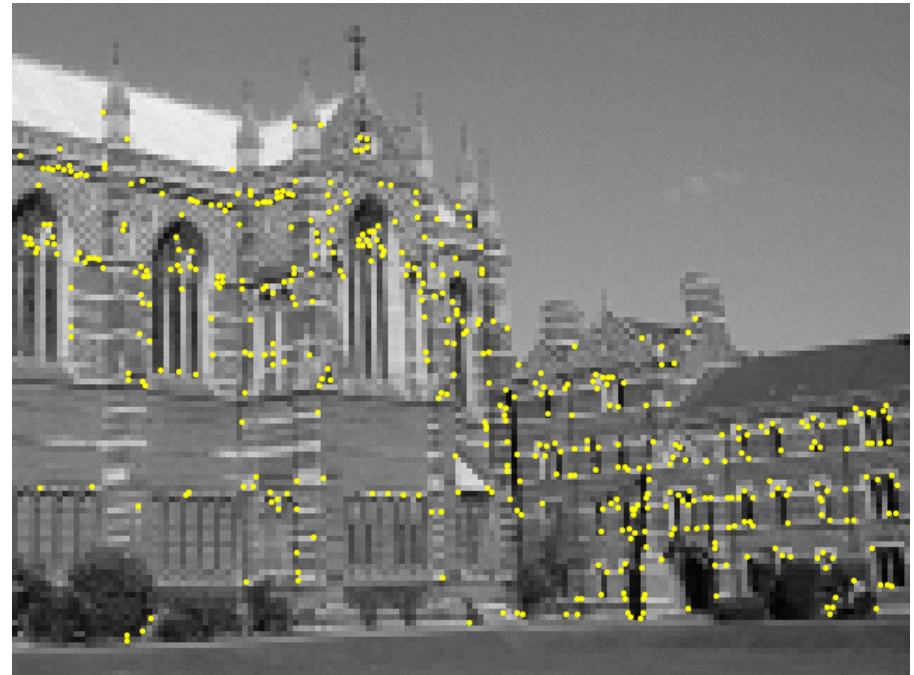
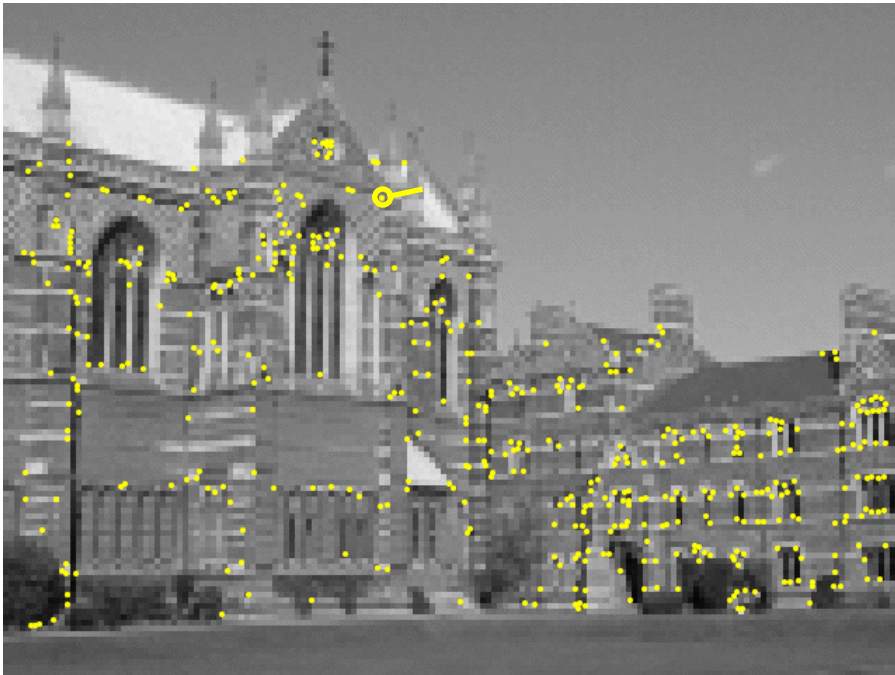
Automatic **H** Estimation: Finding Feature Matches

- Best match over threshold within square search window (here ≈ 300 pixels) using SSD or normalized cross-correlation



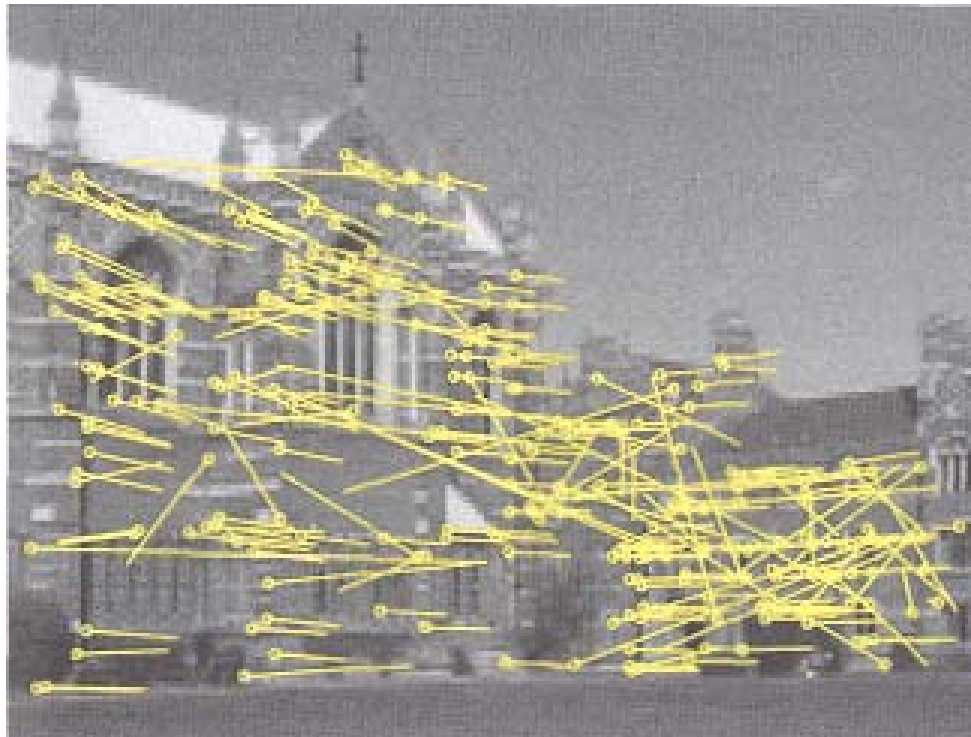
Automatic **H** Estimation: Finding Feature Matches

- Best match over threshold within square search window (here ≈ 300 pixels) using SSD or normalized cross-correlation



from Hartley & Zisserman

Automatic H Estimation: Initial Match Hypotheses



268 matched features (over SSD threshold) in left image
pointing to locations of corresponding right image features

Automatic **H** Estimation:

Applying RANSAC

- Sampling

- Size: Recall that 4 correspondences suffice to define homography, so sample size **S = 4**
- Choice
 - Pick SSD threshold conservatively to minimize bad matches
 - Disregard degenerate configurations
 - Ensure points have good spatial distribution over image

- Distance measure

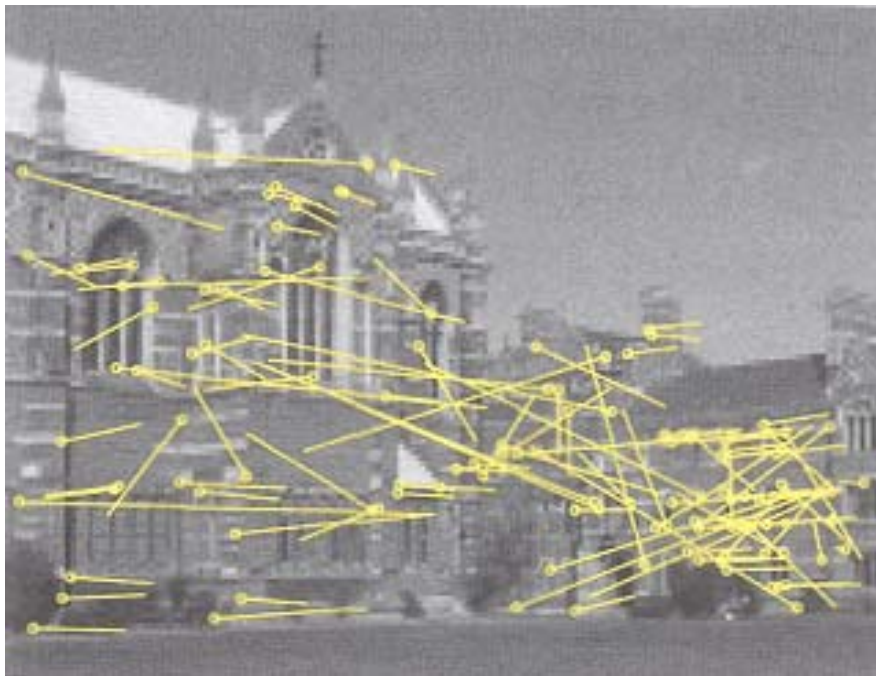
- Obvious choice is symmetric transfer error:

$$d_{transfer}^2 = d(\mathbf{x}, \mathbf{H}^{-1}\mathbf{x}')^2 + d(\mathbf{x}', \mathbf{H}\mathbf{x})^2$$

Automatic **H** Estimation:

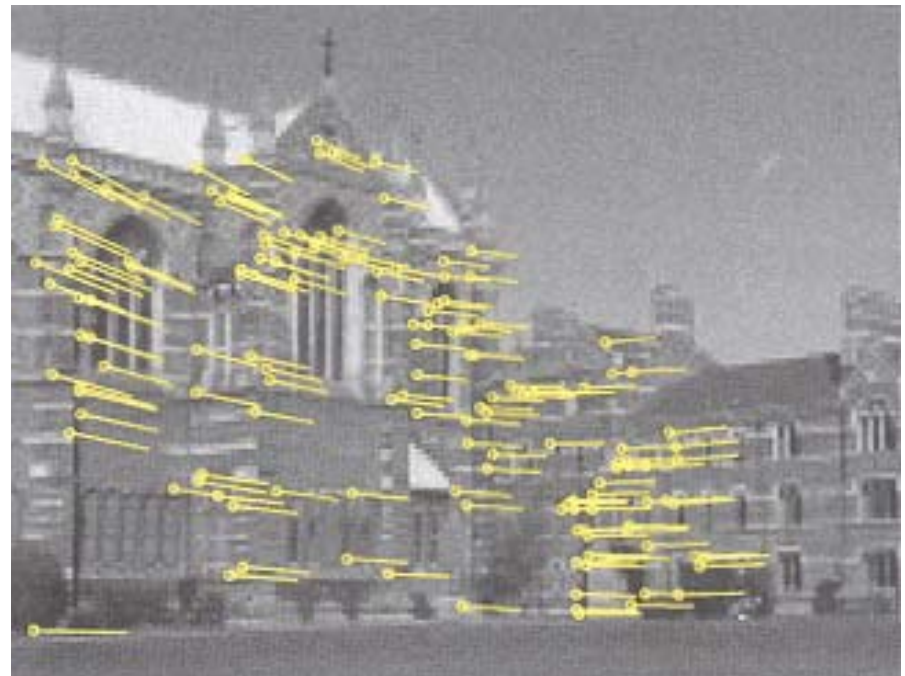
Outliers & Inliers after RANSAC

- 43 samples used with $t = 1.25$ pixels



from Hartley & Zisserman

117 outliers ($\sigma^2 = 0.44$)



151 inliers



RANSAC

- **RAN**dom **SA**mples **C**onsensus: designed for bad data (in best case, up to 50% outliers)
- Idea: Take many random subsets of data to improve robustness
 - Randomly select s points
 - Instantiate a model
 - Get consensus set S_i (supports)
 - Repeat for N trials, return model with $\max |S_i|$