EECE 5639 Computer Vision I

Lecture 10

Hough Transform, RANSAC, Snakes

Next Class

Region segmentation

More Image Features

(Grouping edges)

Contours: Lines and Curves

Edge detectors find “edgels” (pixel level)

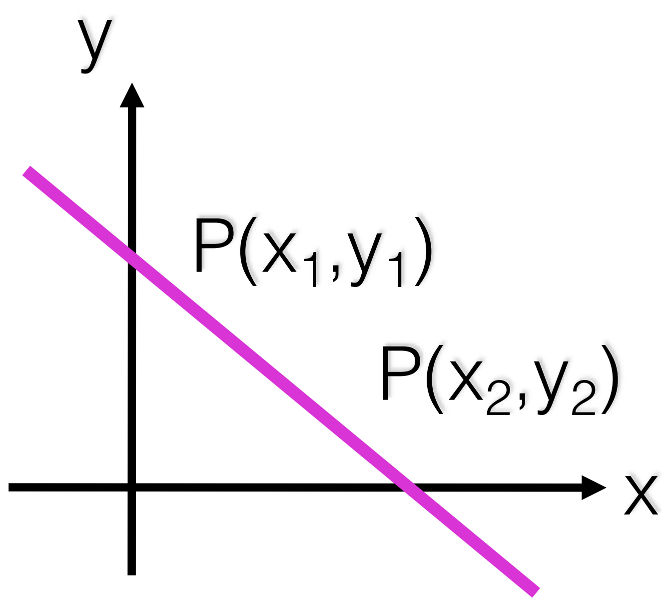
To perform image analysis :

- edgels must be grouped into entities such as contours (higher level).
- Canny does this to certain extent: the detector finds chains of edgels.

Line detection

Mathematical model of a line:

$$y = mx + n$$



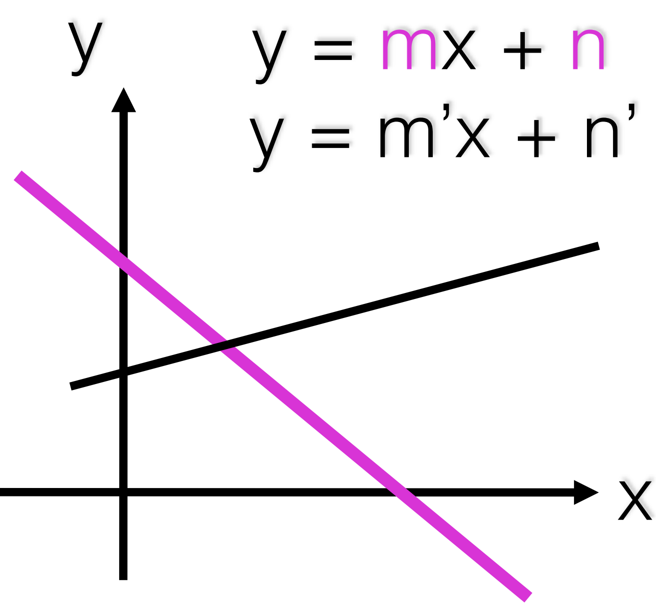
$$y_1 = m\ x_1 + n$$

$$y_2 = m\ x_2 + n$$

$$\Downarrow$$

$$y_N = m\ x_N + n$$

Image and Parameter Spaces



$$y_1 = m\,x_1 + n$$

$$y_2 = m\,x_2 + n$$

$$y_N = m\,x_N + n$$

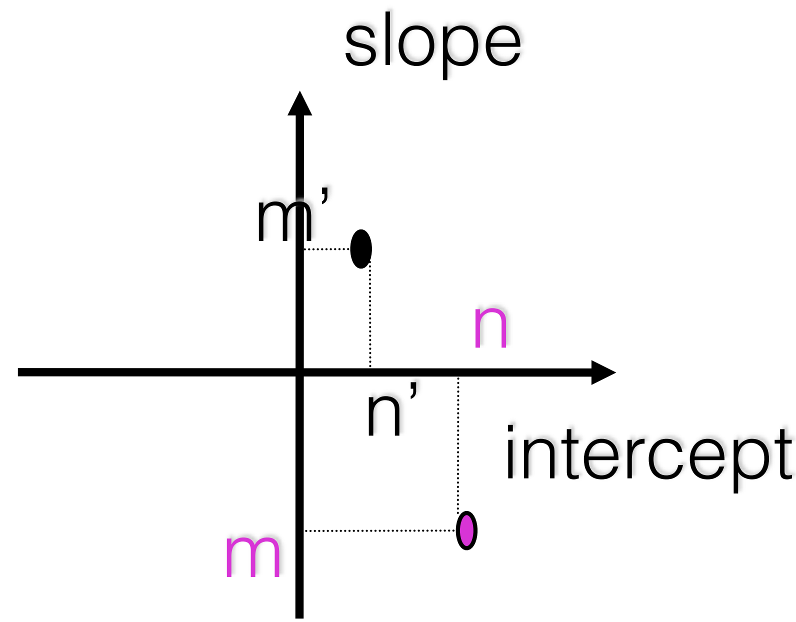


Image Space

Parameter Space

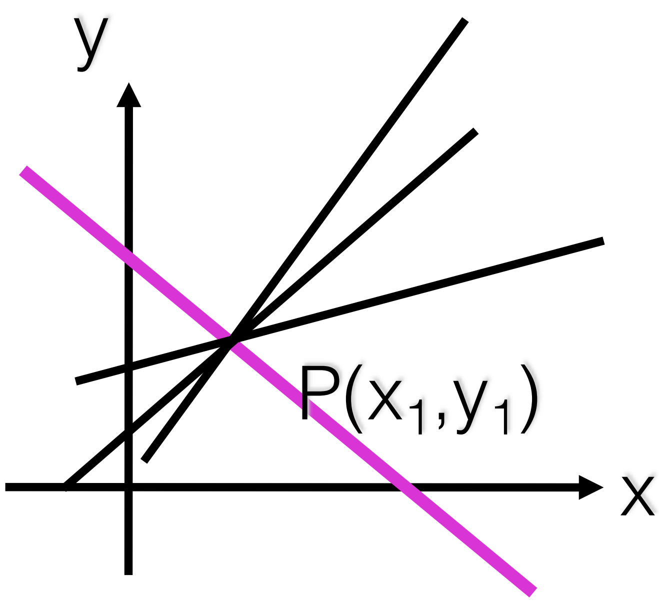
Line in Img. Space ~ Point in Param. Space

Looking at it backwards ...

Image space

Fix (m,n), Vary (x,y) - Line $y = mx + n$

Fix (x_1,y_1), Vary (m,n) – Lines thru a Point $y_1=m\,x_1+n$



Looking at it backwards …

Parameter space

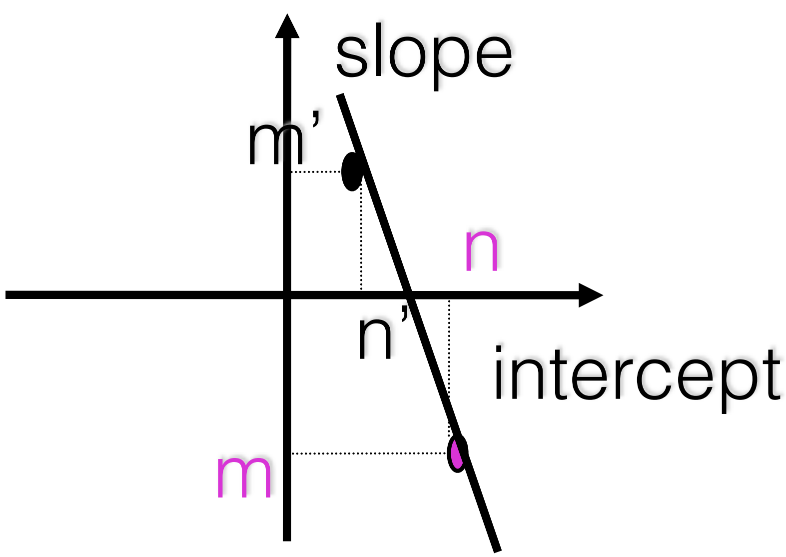
$y_1 = m\, x_1 + n$

Can be re-written as:

$$n = -x_1\, m + y_1$$

Fix $(-x_1, y_1)$, Vary (m,n) - Line

$$n = -x_1\, m + y_1$$



Img-Param Spaces

Image Space

- Lines
- Points
- Collinear points

Parameter Space

- Points
- Lines
- Intersecting lines

Hough Transform Technique

H.T. is a method for detecting straight lines (and curves) in images.

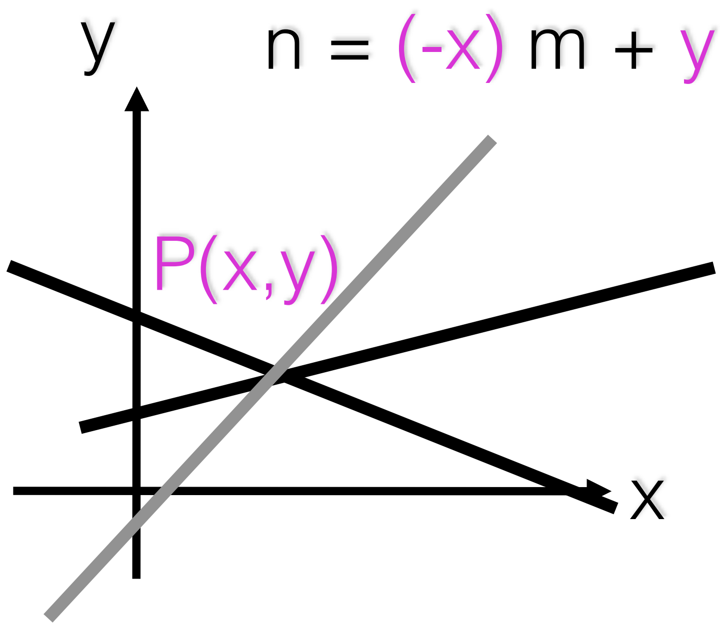
Main idea:

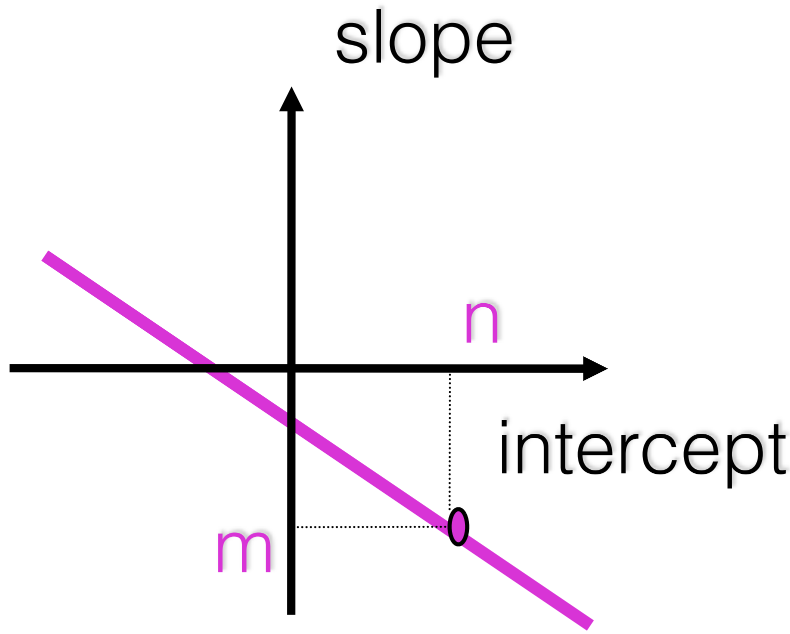
Map a difficult pattern problem into a simple peak detection problem

Hough Transform Technique

Given an edge point, there is an infinite number of lines passing through it (Vary m and n).

These lines can be represented as a line in parameter space.



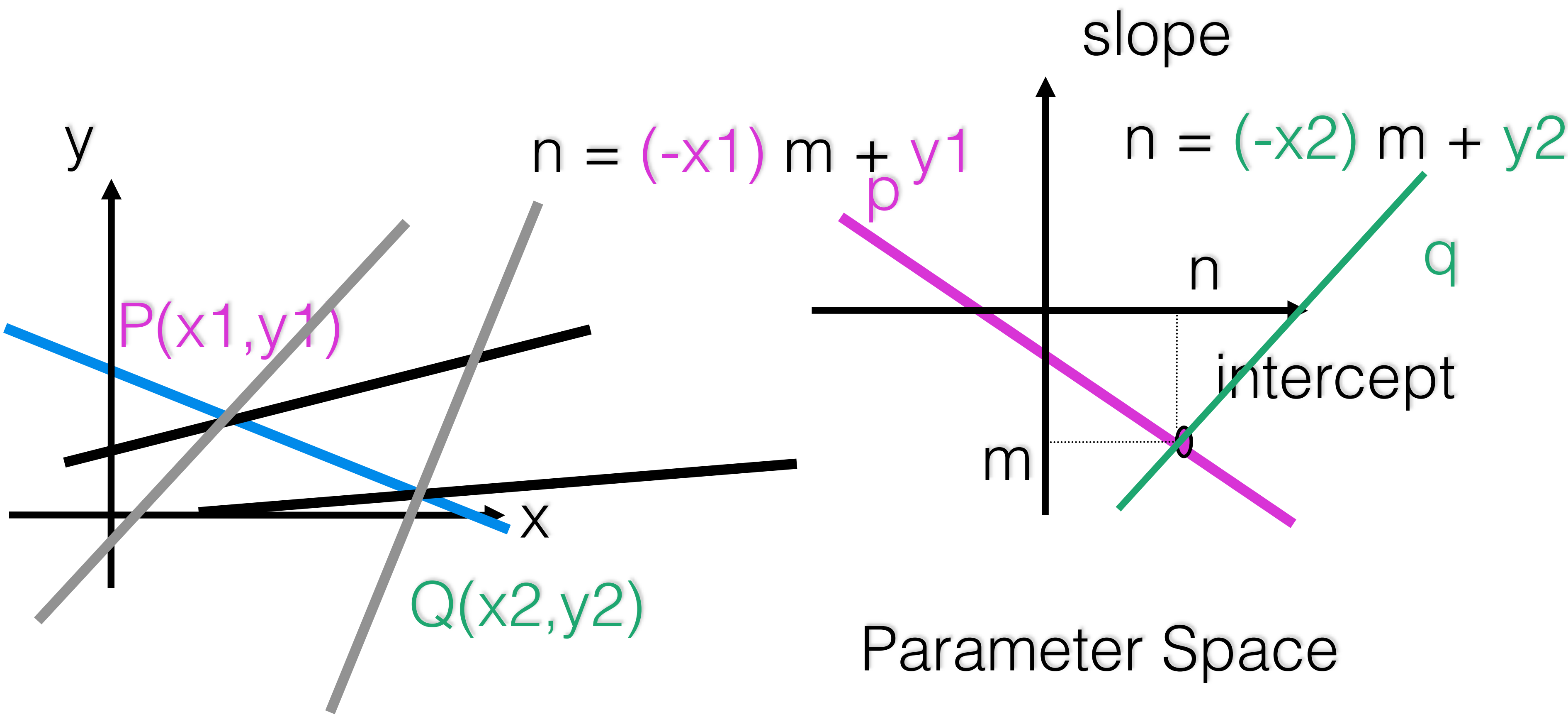


Parameter Space

Hough Transform Technique

Given a set of collinear edge points, each of them have associated a line in parameter space.

- These lines intersect at the point (m,n) corresponding to the parameters of the line in the image space.



Parameter Space

Hough Transform Technique

At each point of the (discrete) parameter space, count how many lines pass through it.

- Use an array of counters
- Can be thought as a " parameter **image**"

The higher the count, the more edges are collinear in the image space.

- Find a peak in the counter array
- This is a "bright" point in the parameter image
- It can be found by thresholding

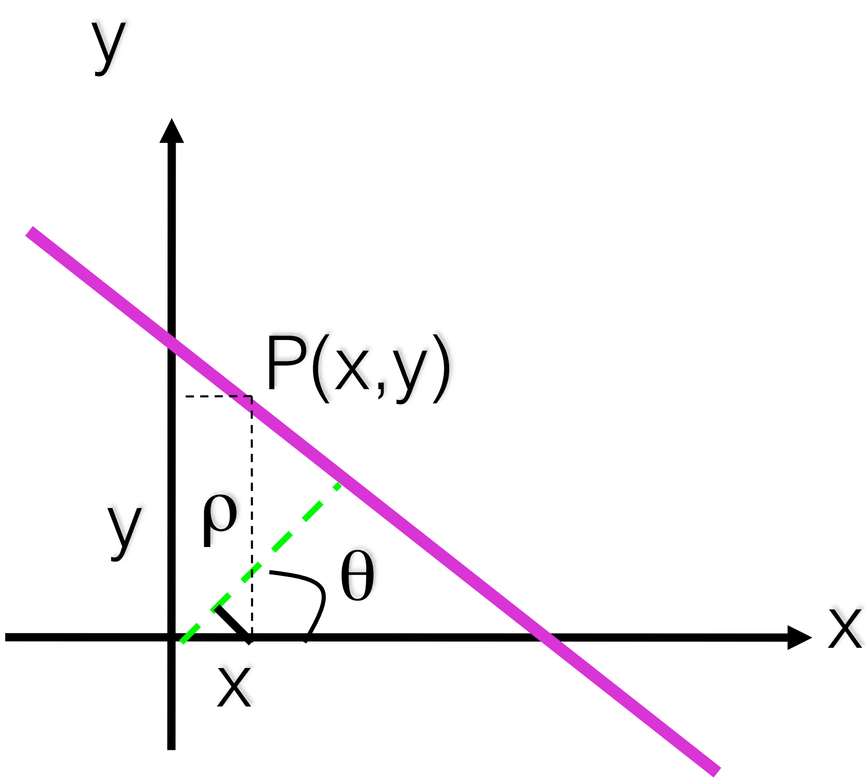
Practical Issues

- The slope of the line is $-\infty<m<\infty$
  - The parameter space is INFINITE
- The representation $y = mx + n$ does not express lines of the form $x = k$

Solution:

Use the “Normal” equation of a line:

$$y = mx + n$$



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

θ Is the line orientation

ρ Is the distance between the origin and the line

New Parameter Space

Use the parameter space (ρ, θ)

The new space is FINITE

- $0 < \rho < D$, where D is the image diagonal.
- $0 < \theta < \pi$

The new space can represent all lines

- $y = k$ is represented with $\rho = k$, $\theta=90$
- $x = k$ is represented with $\rho = k$, $\theta=0$

Consequence:

A Point in Image Space is now represented as a SINUSOID

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

Hough Transform Algorithm

Input is an edge image ($E(i,j)=1$ for edgels)

1. Discretize θ and ρ in increments of $d\theta$ and $d\rho$. Let $A(R,T)$ be an array of integer accumulators, initialized to 0.
2. For each pixel $E(i,j)=1$ and $h=1,2,\dots T$ do
   1. $\rho = i \cos(h * d\theta) + j \sin(h * d\theta)$
   2. Find closest integer k corresponding to ρ
   3. Increment counter $A(h,k)$ by one
3. Find local maxima in $A(R,T)$

Hough Transform Speed Up

If we know the orientation of the edge – usually available from the edge detection step

- We fix theta in the parameter space and increment **only one** counter!
- We can allow for orientation uncertainty by incrementing a few counters around the "nominal" counter.

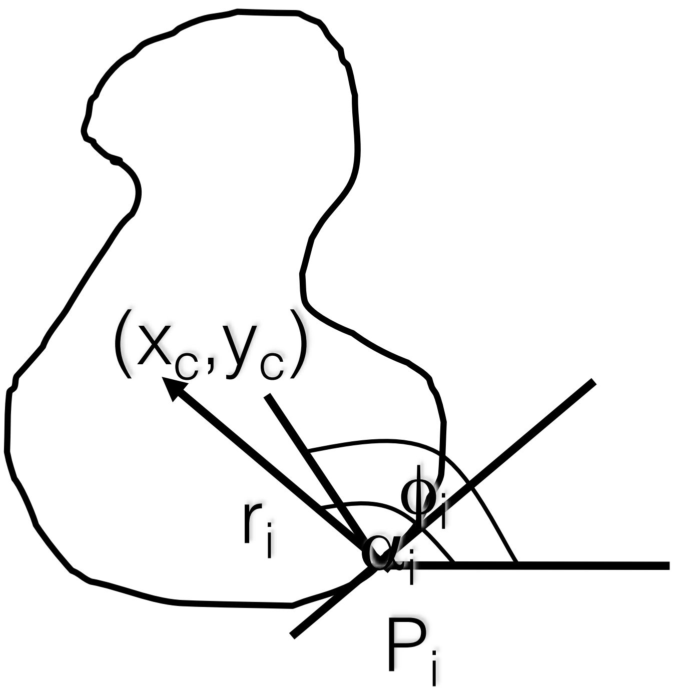
Hough Transform for Curves

The H.T. can be generalized to detect any curve that can be expressed in parametric form:

- $y = f(x, a1, a2, \ldots ap)$
- a1, a2, … ap are the parameters
- The parameter space is p-dimensional
- The accumulating array is LARGE!

Generalizing the H.T.

The H.T. can be used even if the curve has not a simple analytic form!



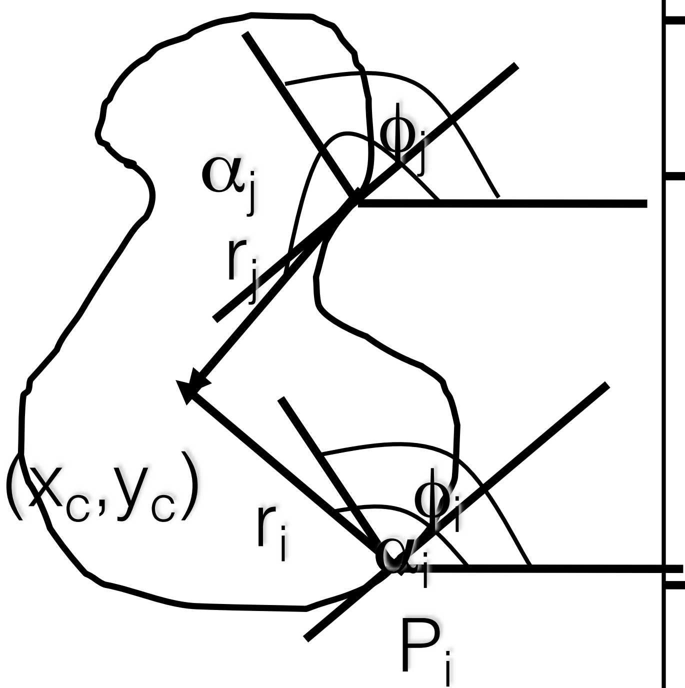
$$x_c = x_i + r_i\cos(\alpha_i)$$

$$y_c = y_i + r_i\sin(\alpha_i)$$

1. Pick a reference point (x_c,y_c)
2. For $i = 1,\ldots,n$:
   a. Draw segment to P_i on the boundary.
   b. Measure its length r_i, and its orientation α_i.
   c. Write the coordinates of (x_c,y_c) as a function of r_i and α_i
   d. Record the gradient orientation ϕ_i at P_i.
5. Build a table with the data, indexed by ϕ_i .

Generalizing the H.T.

Suppose, there were m **different** gradient orientations: (m <= n)



$$x_c = x_i + r_i\cos(\alpha_i)$$

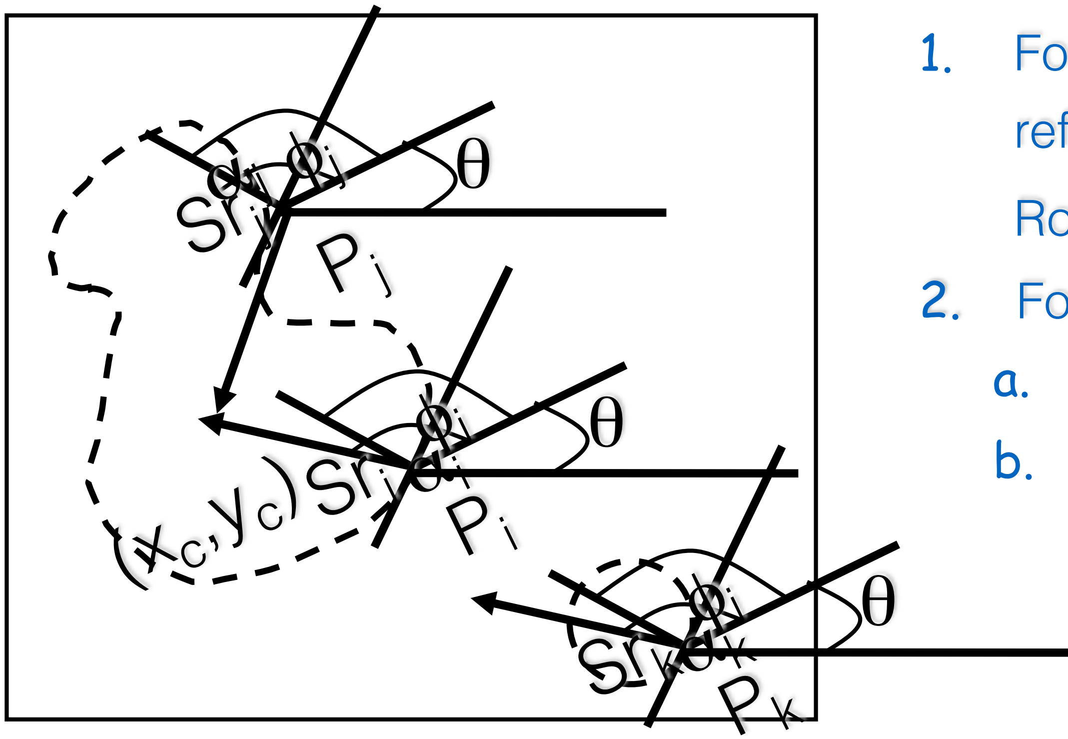
$$y_c = y_i + r_i\sin(\alpha_i)$$

ϕ_1	$(r^1{}_1,\alpha^1{}_1),(r^1{}_2,\alpha^1{}_2),\ldots,(r^1{}_{n1},\alpha^1{}_{n1})$
ϕ_2	$(r^2{}_1,\alpha^2{}_1),(r^2{}_2,\alpha^1{}_2),\ldots,(r^2{}_{n2},\alpha^1{}_{n2})$
.	.
.	.
.	.
ϕ_m	$(r^m{}_1,\alpha^m{}_1),(r^m{}_2,\alpha^m{}_2),\ldots,(r^m{}_{nm},\alpha^m{}_{nm})$

H.T. table

Generalized H.T. Algorithm:

Finds a rotated, scaled, and translated version of the curve:



$$x_c = x_i + r_i \cos(\alpha_i)$$

$$y_c = y_i + r_i \sin(\alpha_i)$$

1. Form an A accumulator array of possible reference points (x_c, y_c), scaling factor S and Rotation angle θ.
2. For each edge (x,y) in the image:
   a. Compute $\phi(x,y)$
   b. For each (r,α) corresponding to $\phi(x,y)$ do:
      1. For each S and θ:
         a. $x_c = x_i + r(\phi)\, S \cos[\alpha(\phi) + \theta]$
         b. $y_c = y_i + r(\phi)\, S \sin[\alpha(\phi) + \theta]$
         c. $A(x_c, y_c, S, \theta)$ ++
3. Find maxima of A.

H.T. Summary

H.T. is a “voting” scheme

points vote for a set of parameters describing a line or curve.

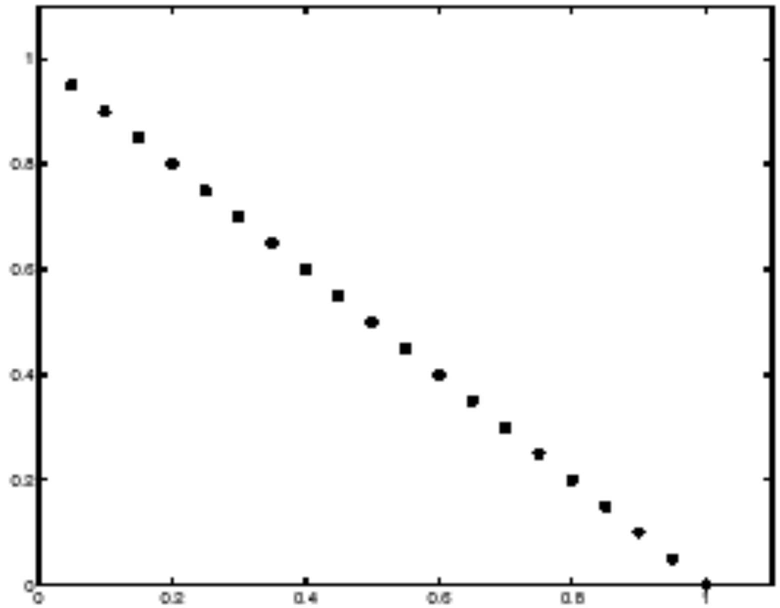
The more votes for a particular set

the more evidence that the corresponding curve is present in the image.

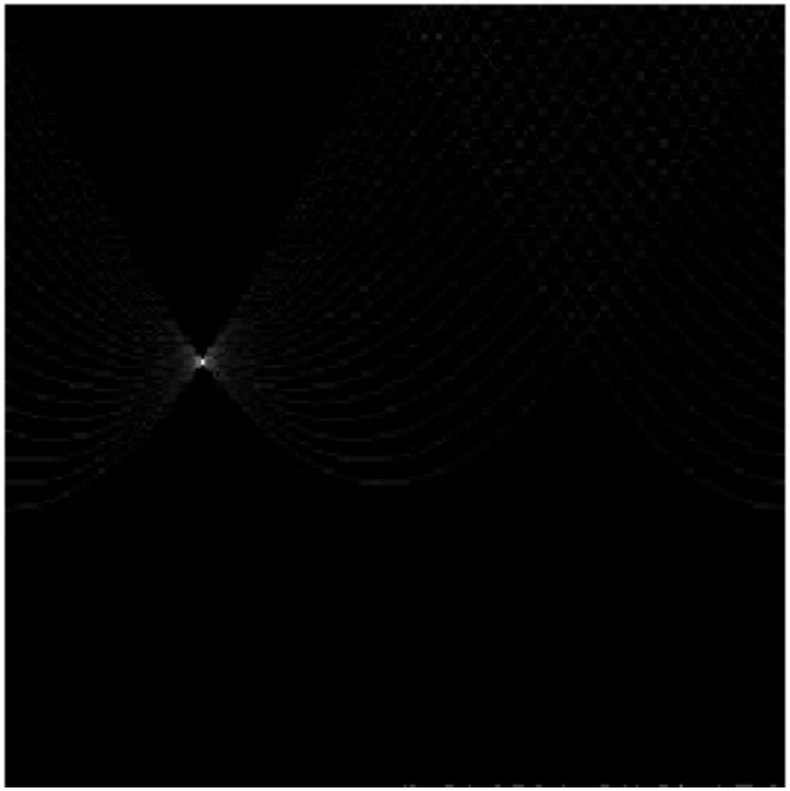
Can detect MULTIPLE curves in one shot.

Computational cost increases with the number of parameters describing the curve.

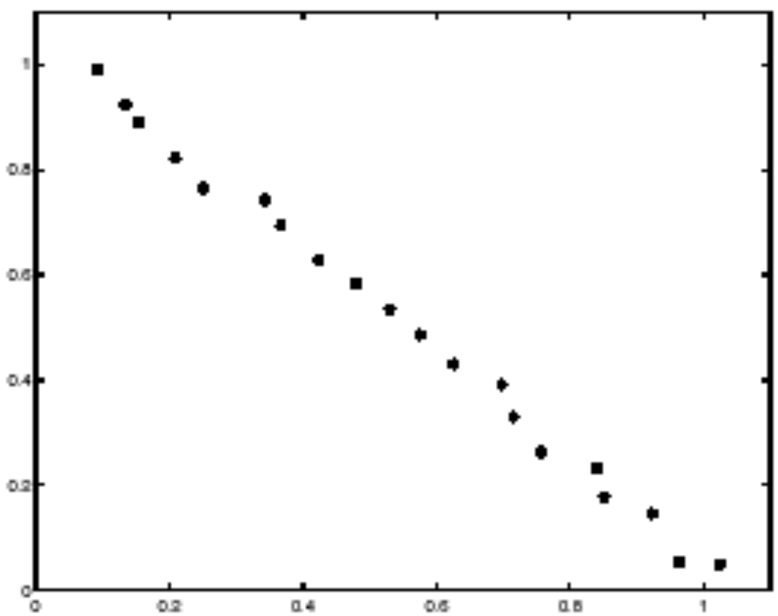
Hough Transf. & Noise



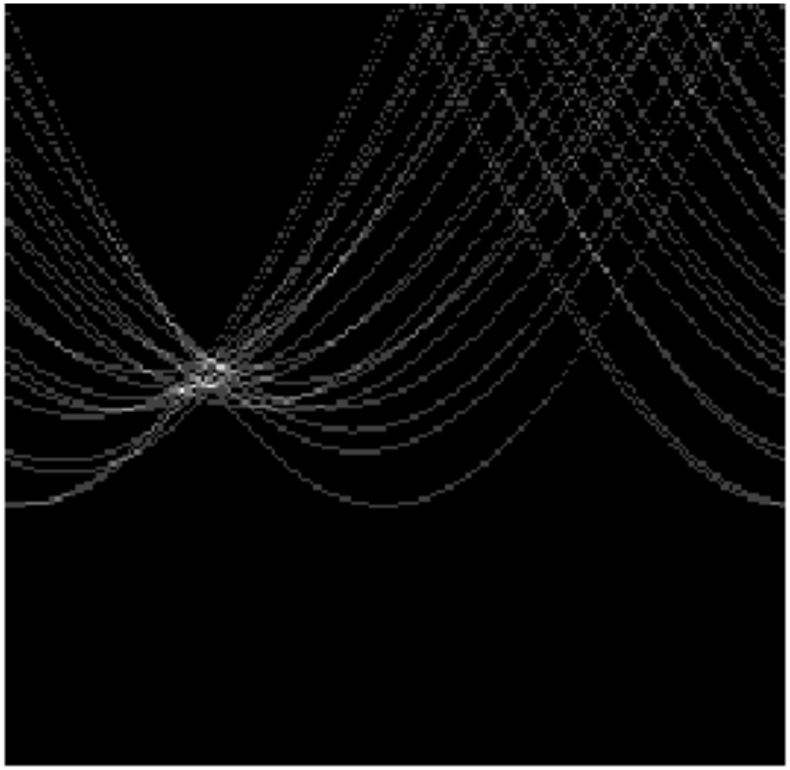
tokens



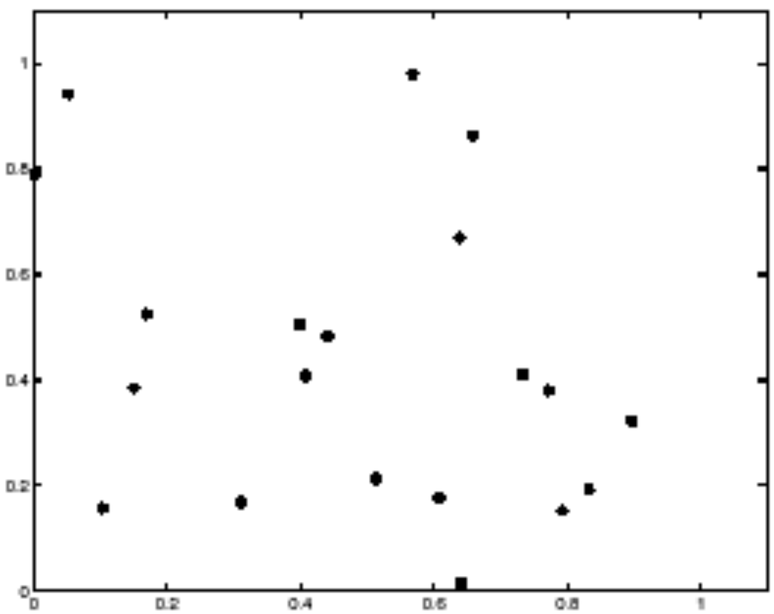
votes



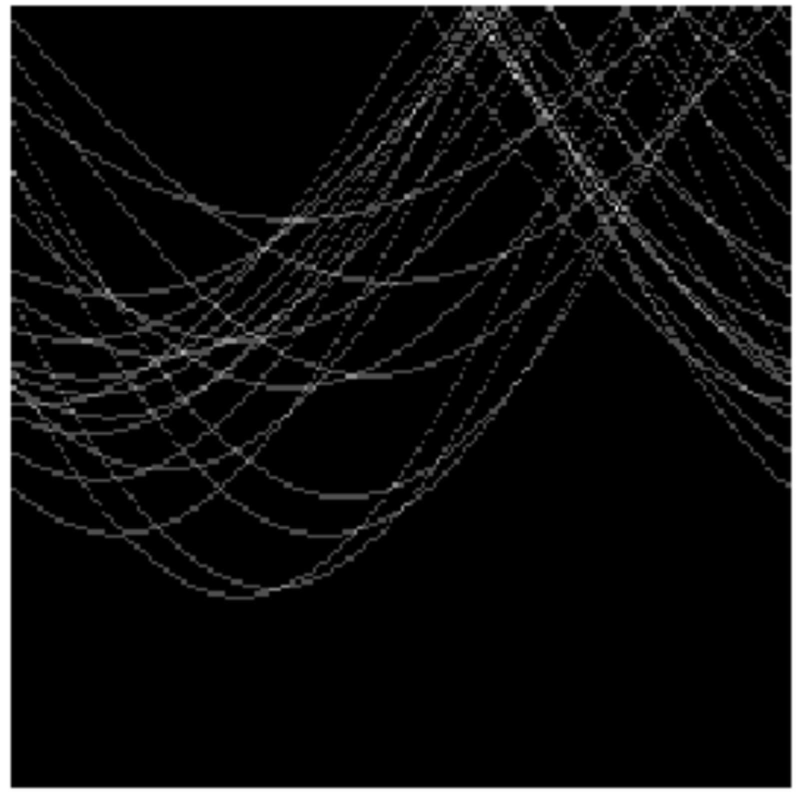
tokens

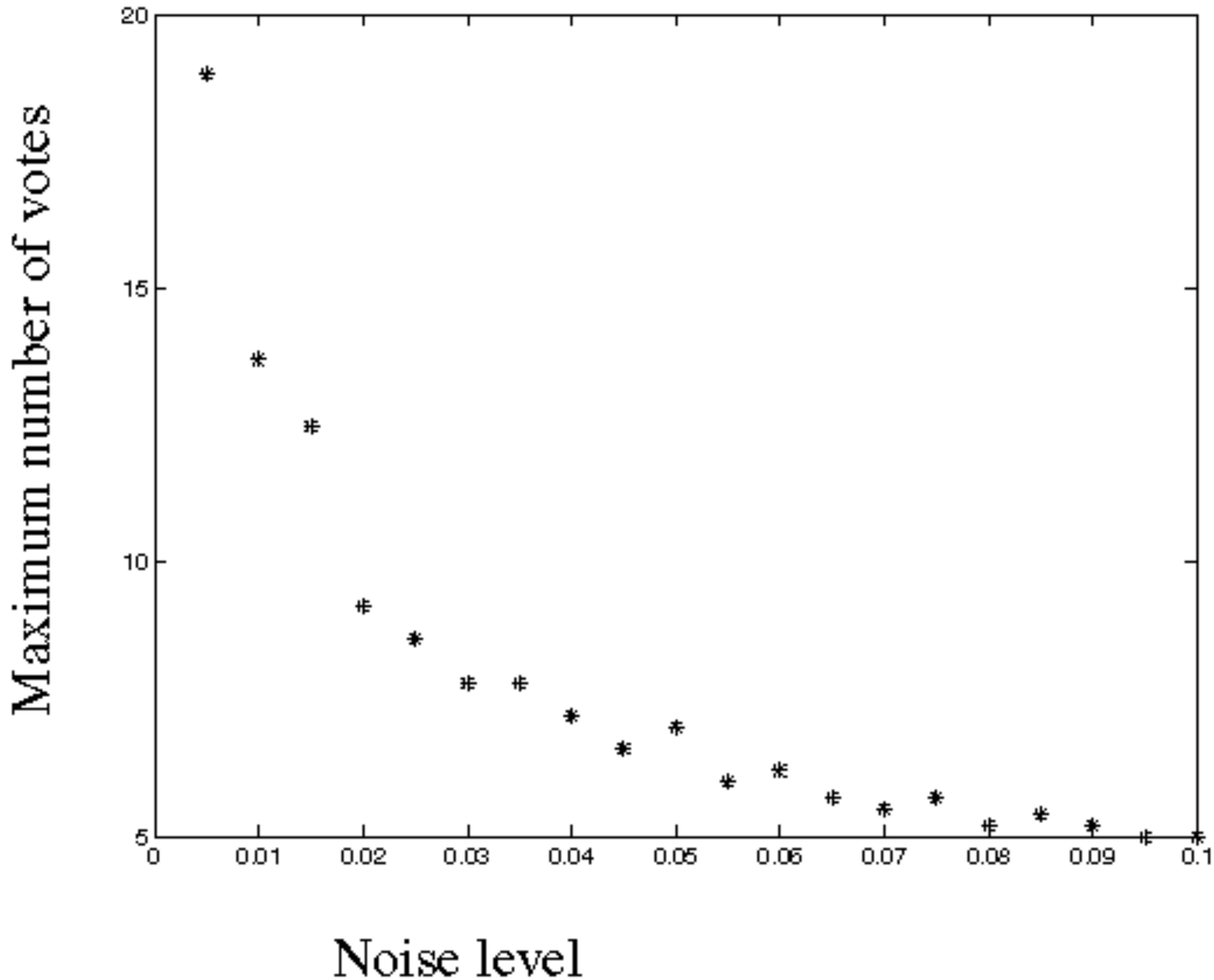


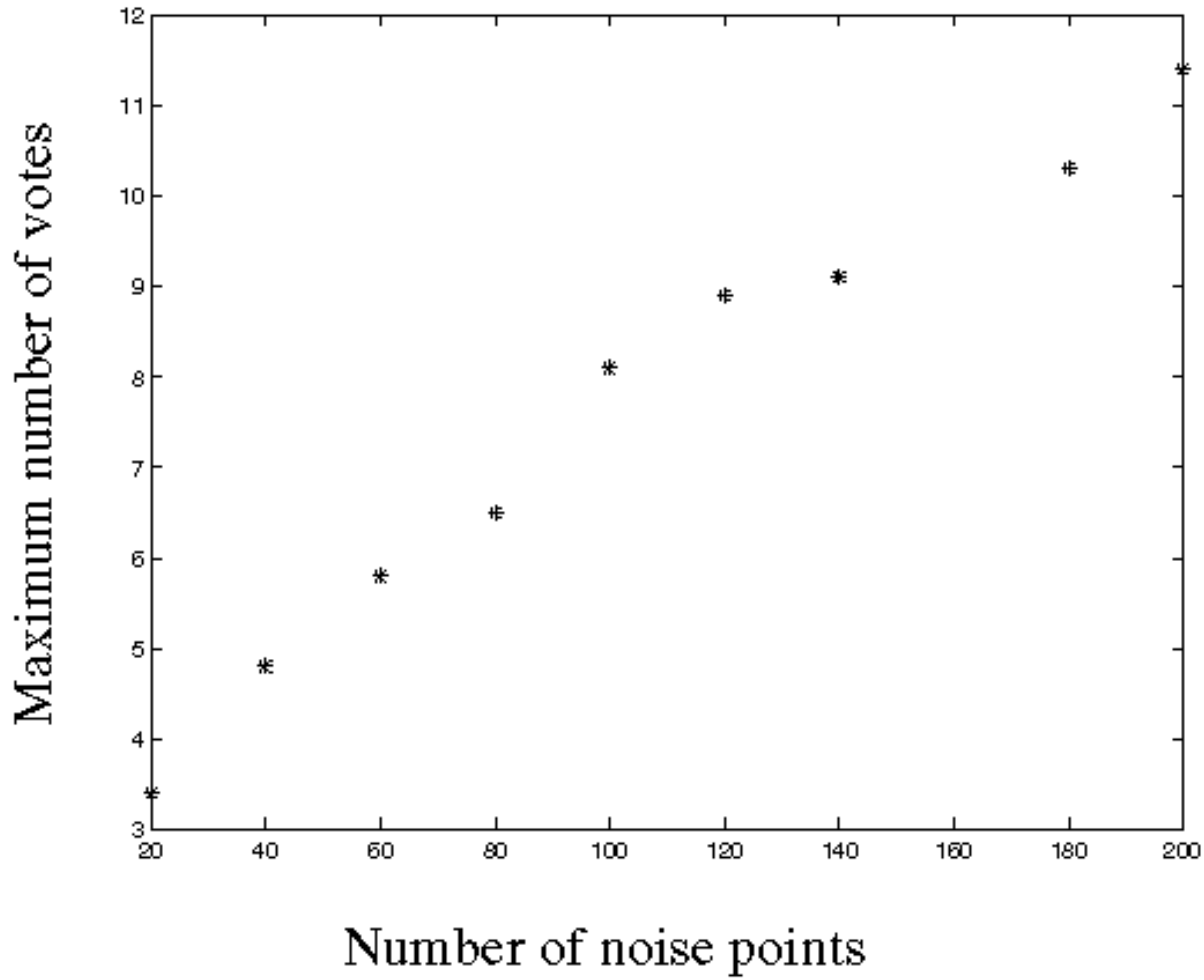
votes



0
0.2
0.4
0.6
0.8
1



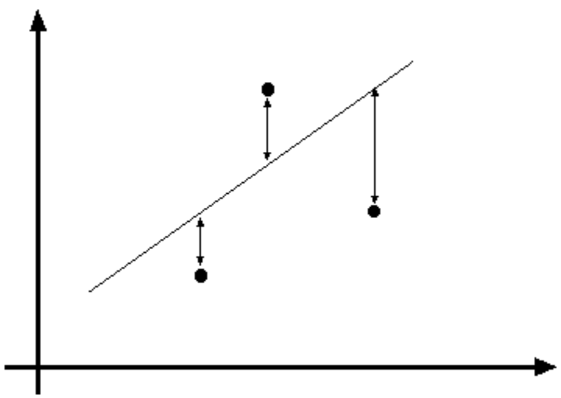

Maximum number of votes
20
15
10
5
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Noise level


Maximum number of votes
12
11
10
9
8
7
6
5
4
3
20
40
60
80
100
120
140
160
180
200
Number of noise points

Segmentation by Fitting

Fitting Lines

Using Least Squares Fitting Error:



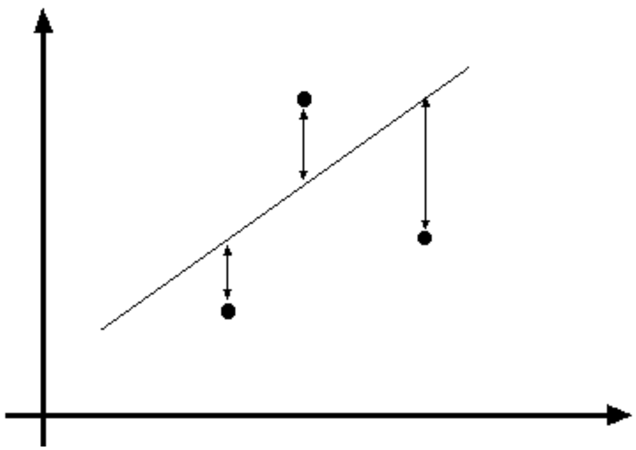
$$min\Phi = \sum_i (y_i - ax_i - b)^2$$

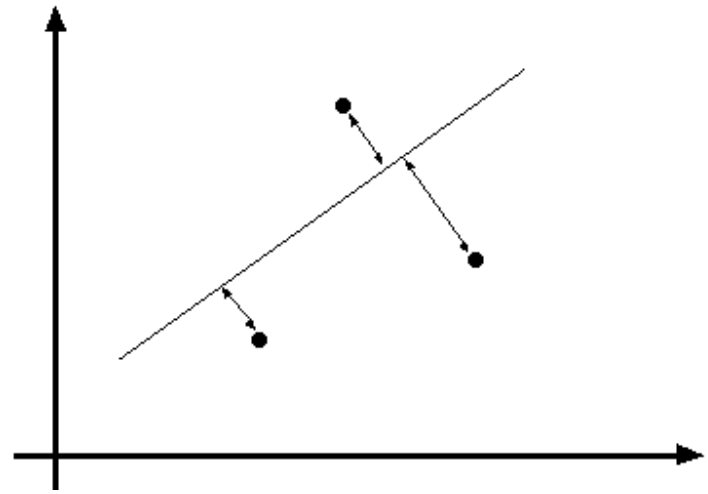
$$\frac{\partial \Phi}{\partial a} = 2a\sum_i x_i^2 - 2\sum_i x_i(y_i - b) = 0$$

$$\frac{\partial \Phi}{\partial b} = -2\sum_i (y_i - ax_i - b) = 0$$

$$\bar{xy} = \bar{x^2}a + \bar{x}b$$
$$\bar{y} = \bar{x}a + b$$

$$\begin{bmatrix} \bar{xy} \\ \bar{y} \end{bmatrix} = \begin{bmatrix} \bar{x^2} & \bar{x} \\ \bar{x} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

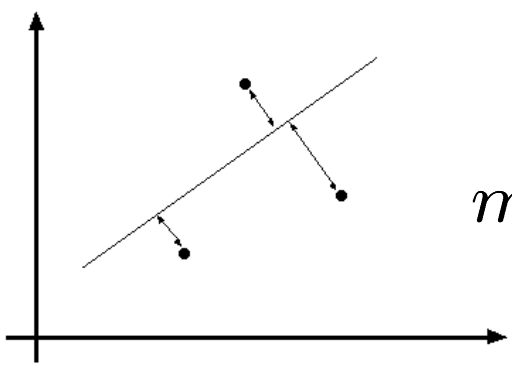




Line fitting can be max. likelihood - but choice of model is important

Fitting Lines

Using Total Least Squares Fitting Error:



$$min\Phi = \sum_i (ax_i + by_i + c)^2$$

$$\text{s.t.} a^2 + b^2 = 1$$

$$min\Lambda = \sum_i (ax_i + by_i + c)^2 + \lambda(a^2 + b^2 - 1)$$

$$\frac{\partial \Lambda}{\partial a} = 0 \quad \frac{\partial \Lambda}{\partial b} = 0 \quad \frac{\partial \Lambda}{\partial c} = 0 \quad \frac{\partial \Lambda}{\partial \lambda} = 0$$

$$\begin{bmatrix} \overline{x^2} - \bar{x}\bar{x} & \overline{xy} - \bar{x}\bar{y} \\ \overline{xy} - \bar{x}\bar{y} & \overline{y^2} - \bar{y}\bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \mu \begin{bmatrix} a \\ b \end{bmatrix}$$

Eigenvalue Problem!

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

Possible strategies

- Hough transform
- Incremental line fitting
- K-means

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
```

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

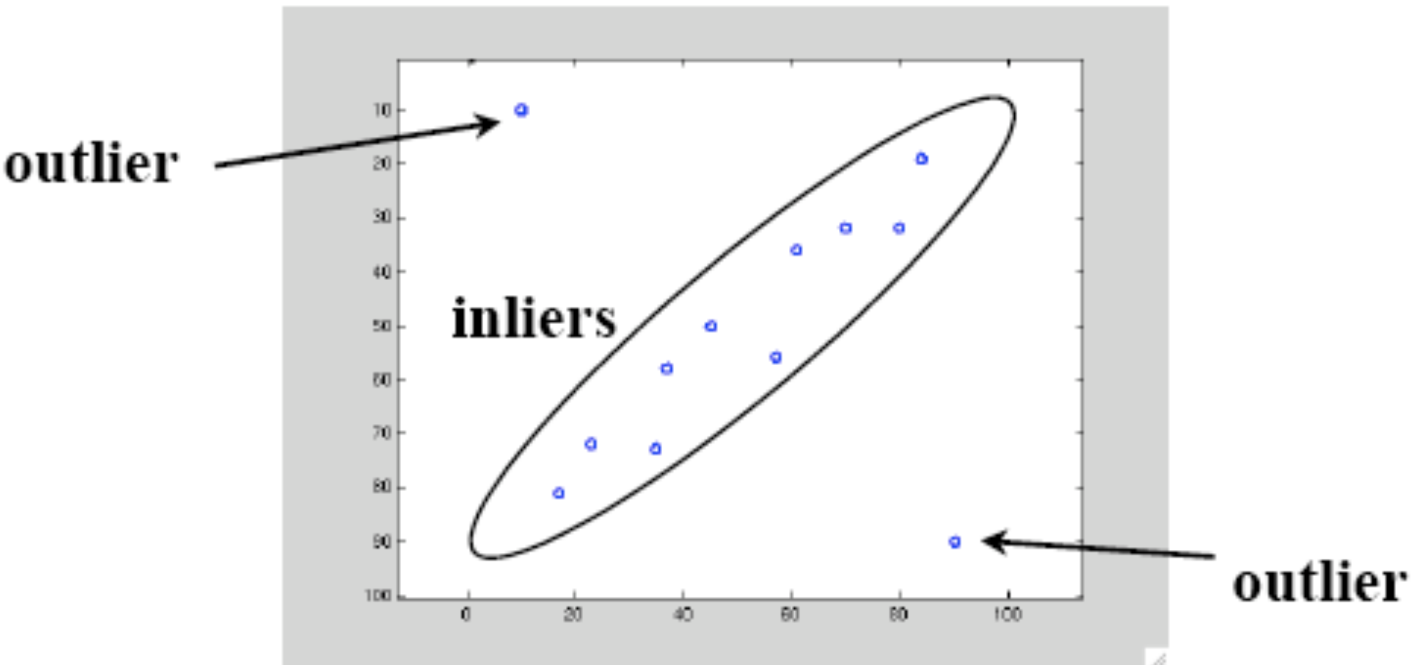
Robustness

As we have seen, squared error can be a source of bias in the presence of noise points

- One fix is EM - we'll not do this in this class
- Another is an M-estimator
  - Square nearby, threshold far away
- A third is RANSAC
  - Search for good points

Inliers-Outliers

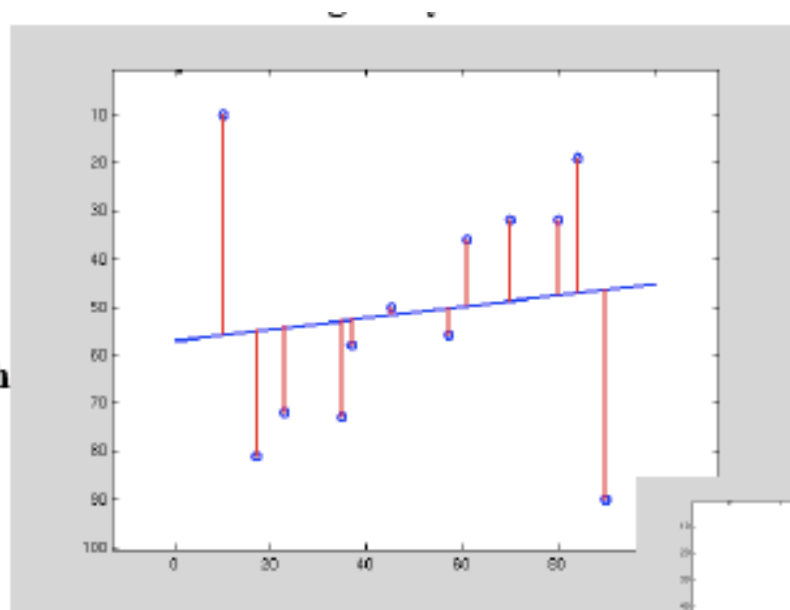
Loosely speaking, outliers are "bad data" points that do not fit the model. Points that fit the model are inliers.



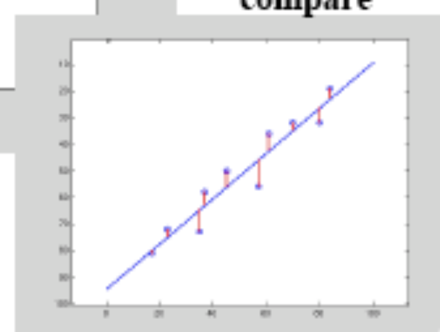
Problems with Outliers

Least square estimation is very sensitive to outliers! :
Few outliers can GREATLY skew the result.

Least squares regression with outliers



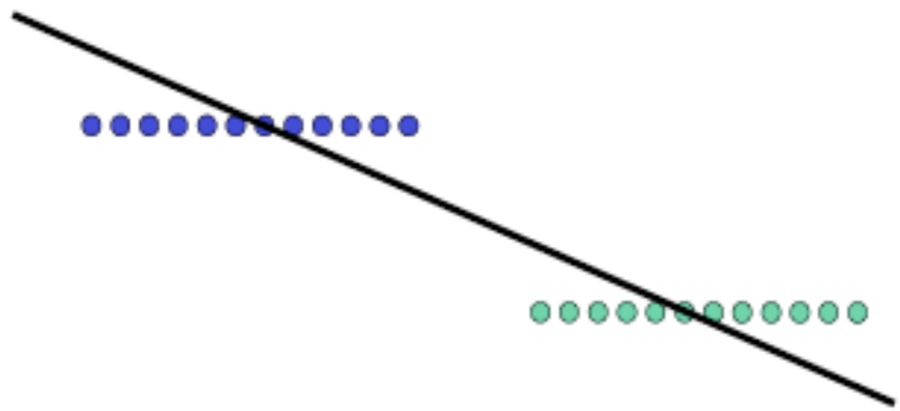
compare



Outliers are not the only problem

Multiple structures can also skew results.

The fitting procedure implicitly assumes ONE instance



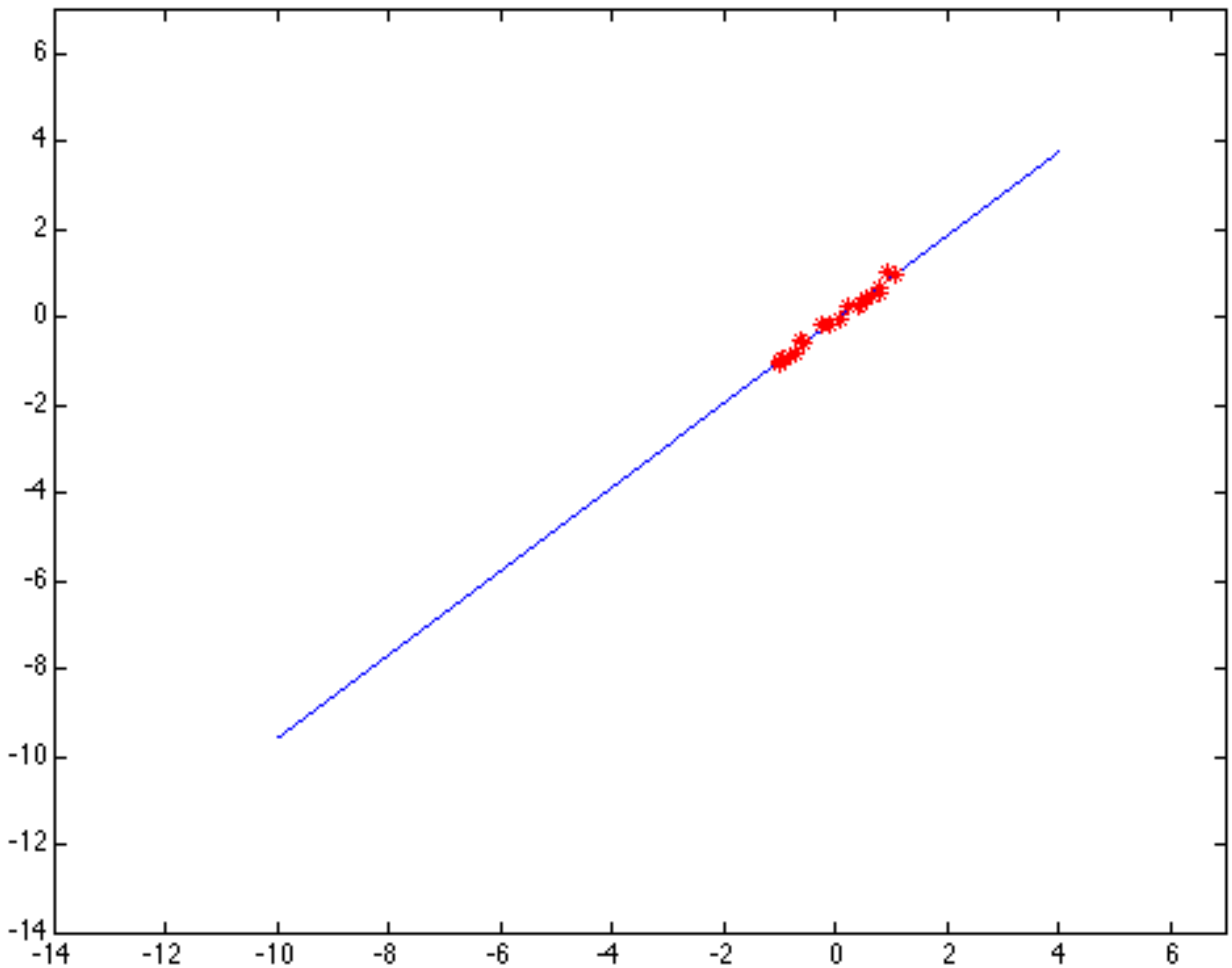
Robust Estimation

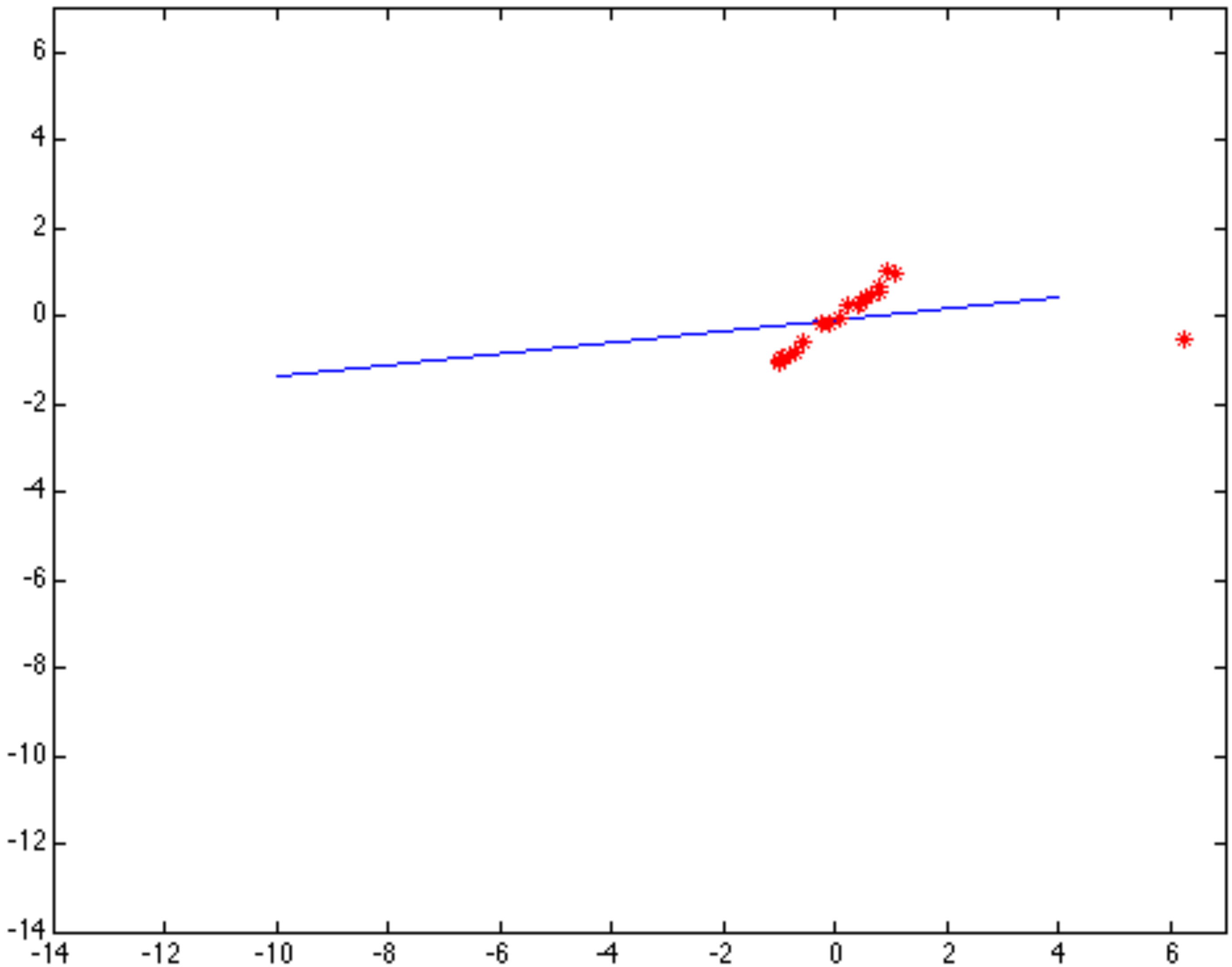
Two steps:

Classify data into INLIERS and OUTLIERS

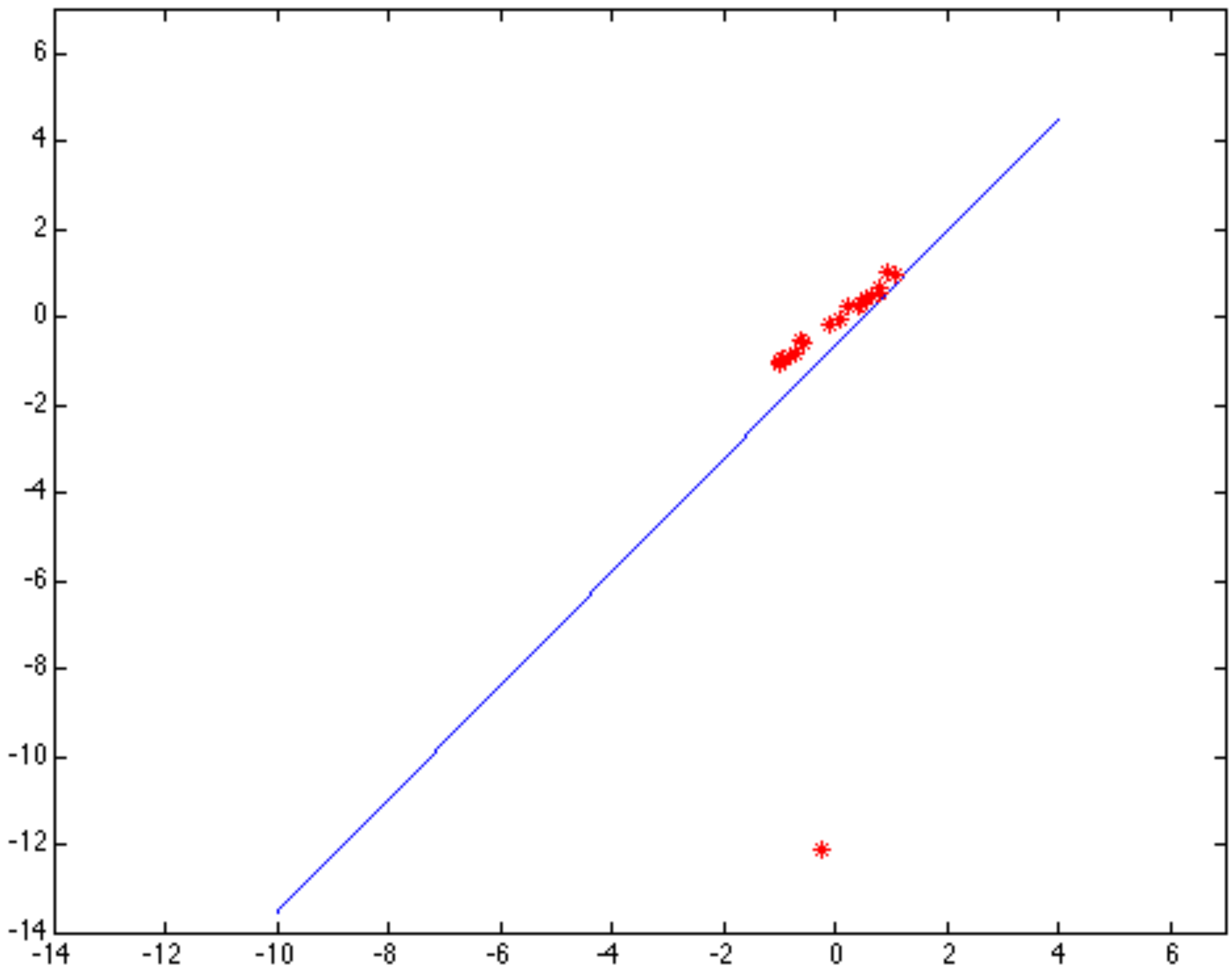
Use only INLIERS to fit the model

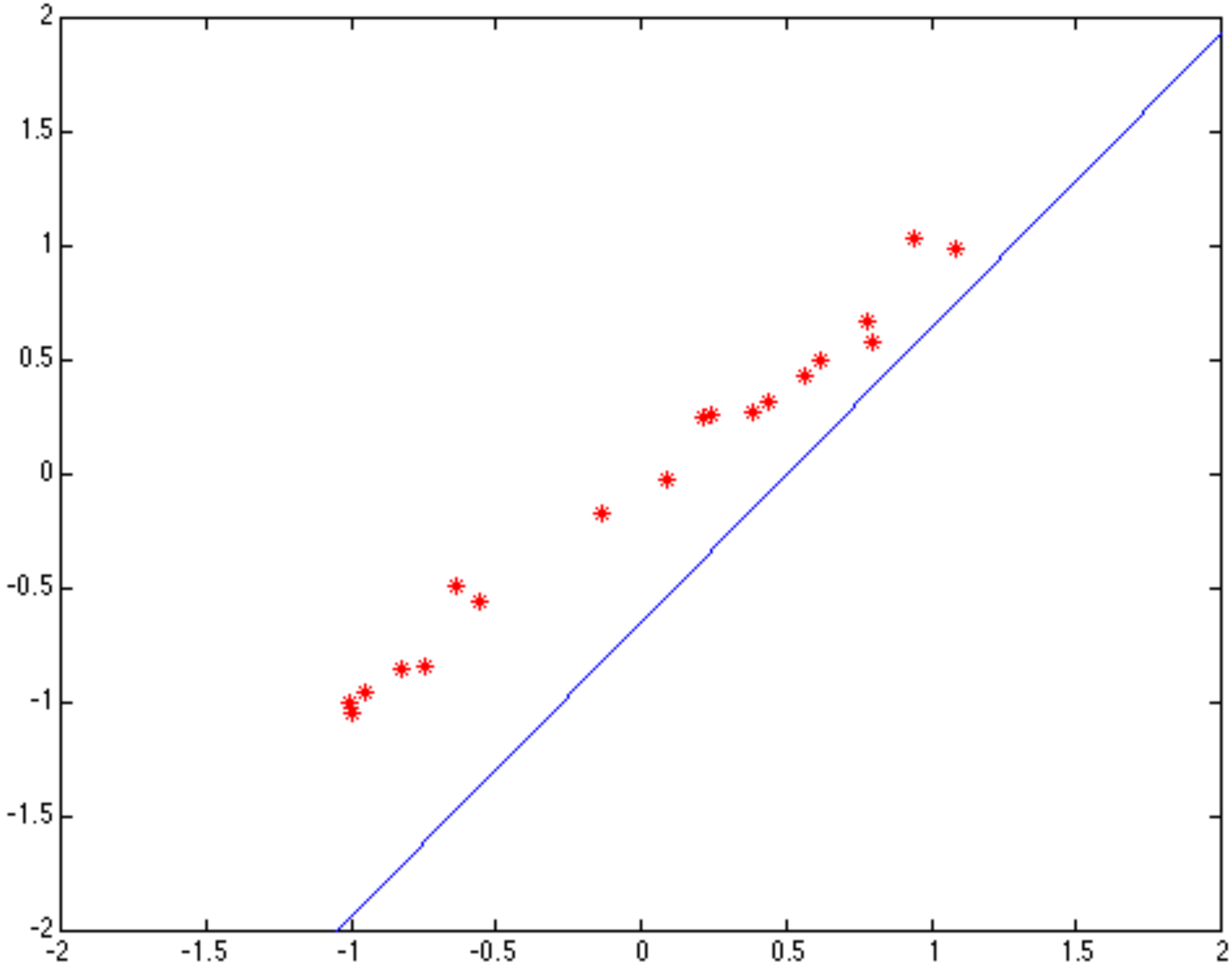
RANSAC is and example of this approach.


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6



6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Robustness and M-estimators

LSE methods are very sensitive to outliers: one bad point can have tremendous effect on the solution.

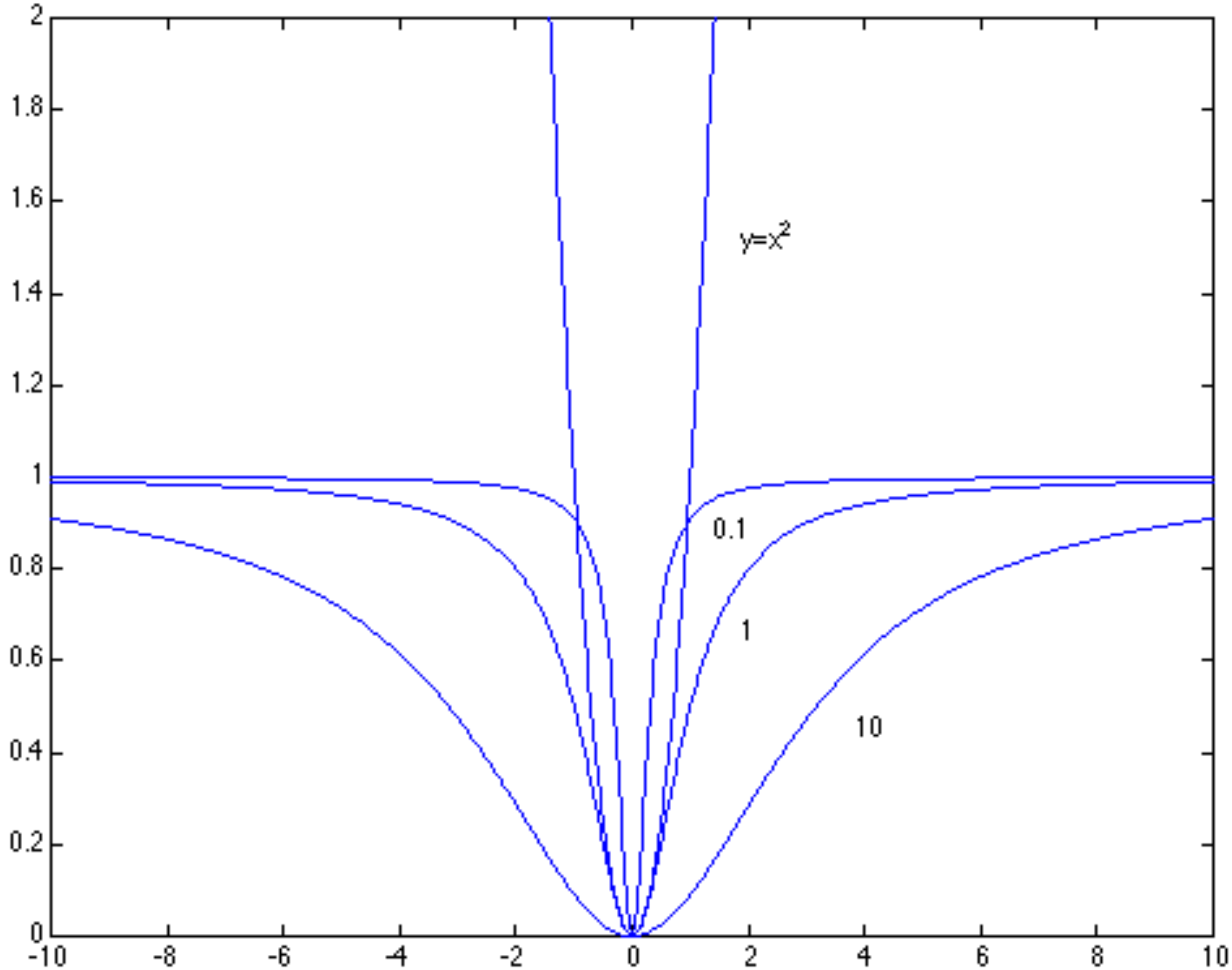
An M-estimator is used to give different weights to the errors:

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

Parameter Controlling Error influence

Residual error at xi

Set of parameters Being fitted



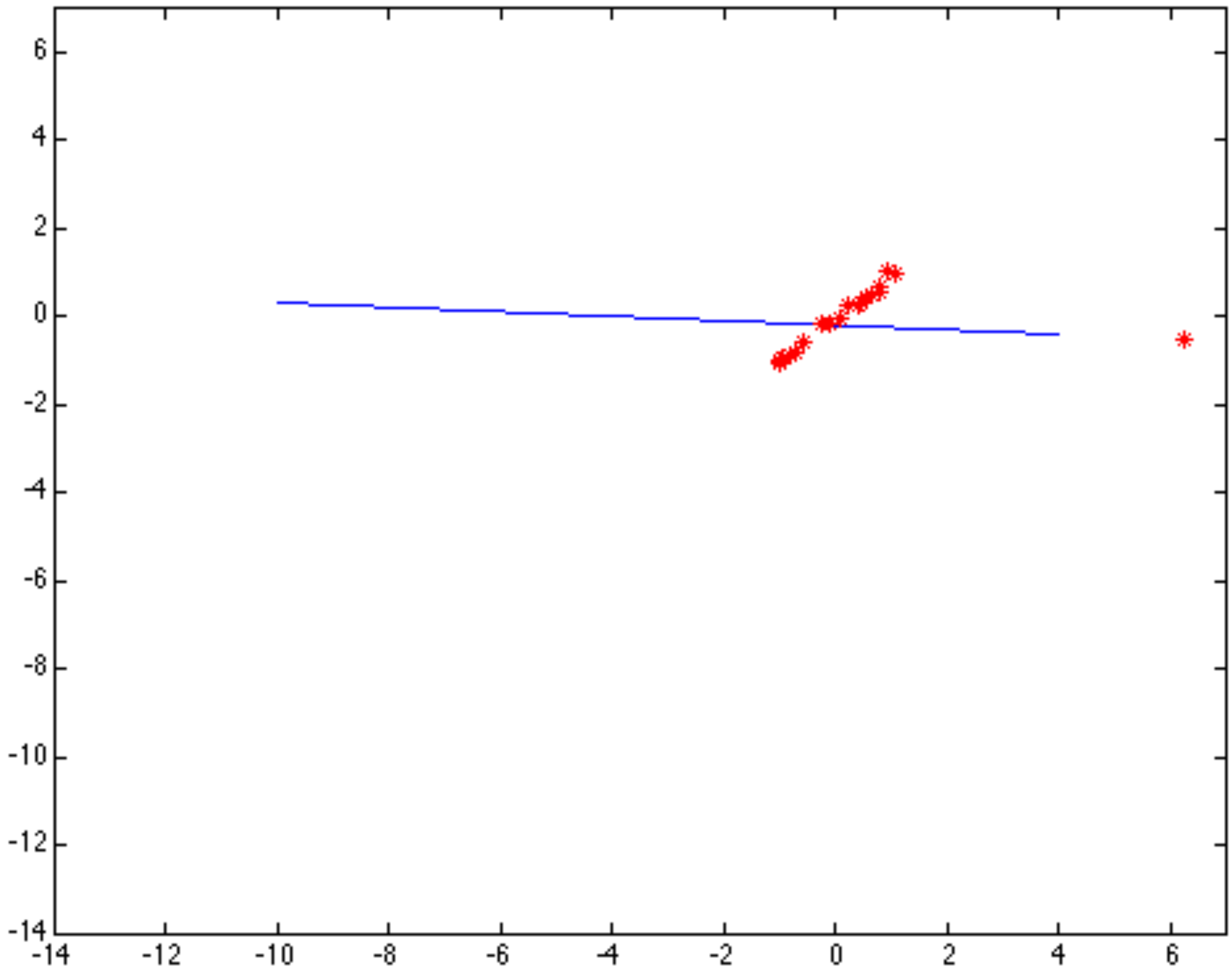
$$\rho(u;\sigma) = \frac{u^2}{u^2+\sigma^2}$$

Issues:

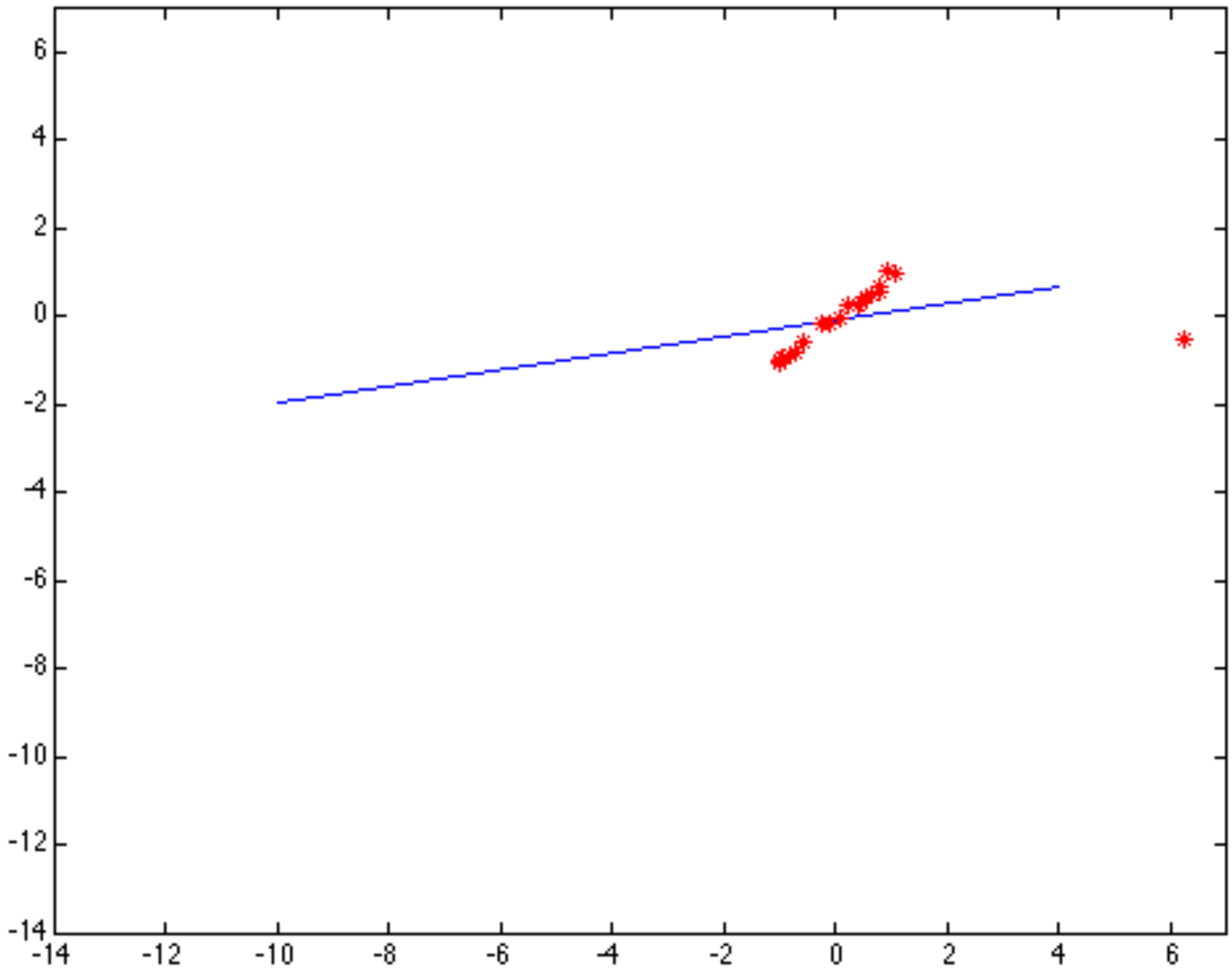
1. Minimizing objective is not longer linear:
   Solutions must be found interactively

2. Need to decide the scale parameter sigma.

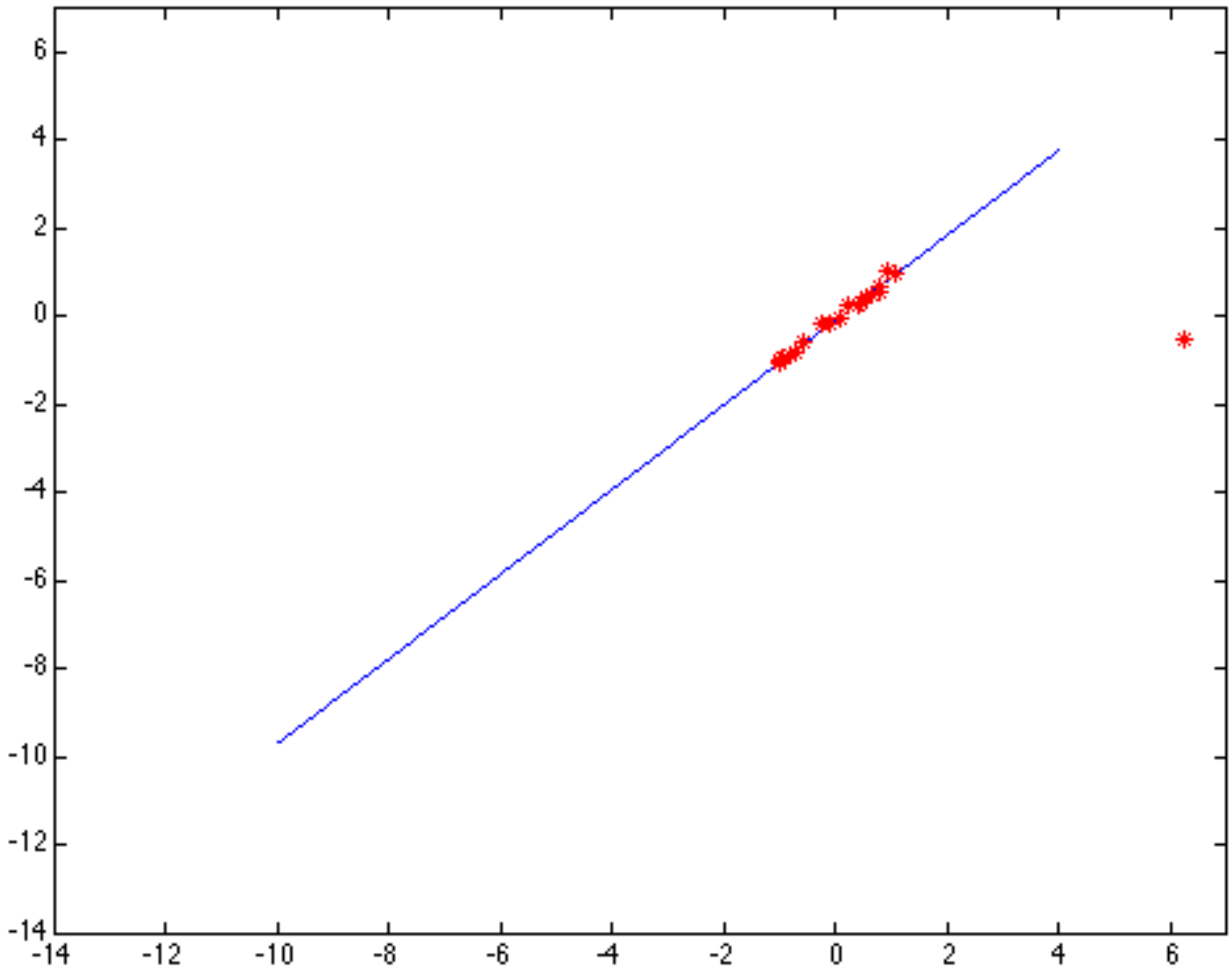
Too small sigma



Too large sigma



Good sigma value:

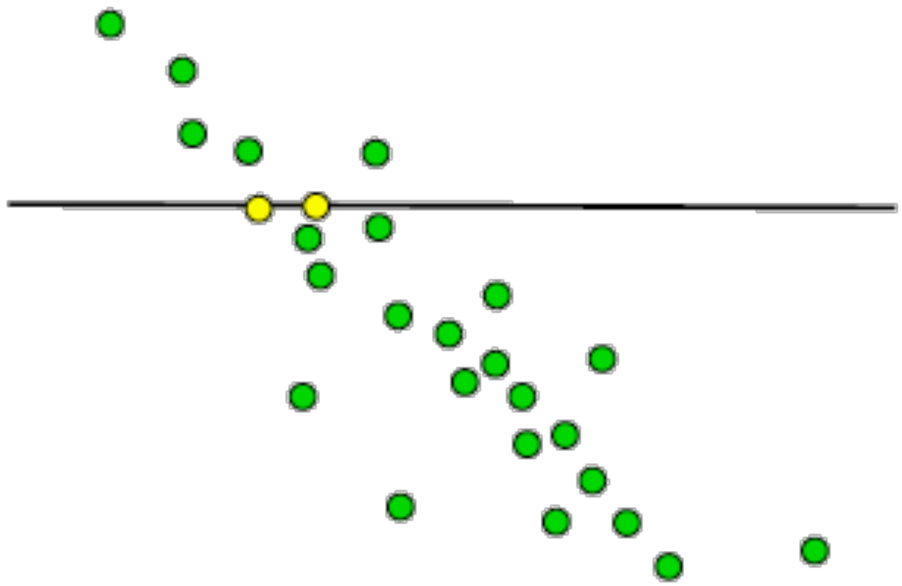


RANSAC

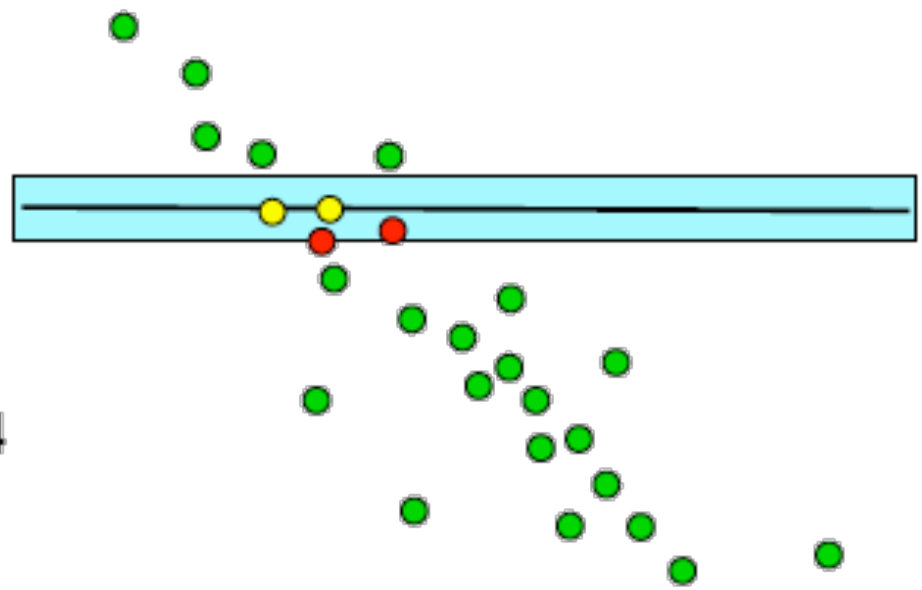
RANdom SAmple Consensus

RANSAC procedure (line example)

RANSAC procedure (line example)

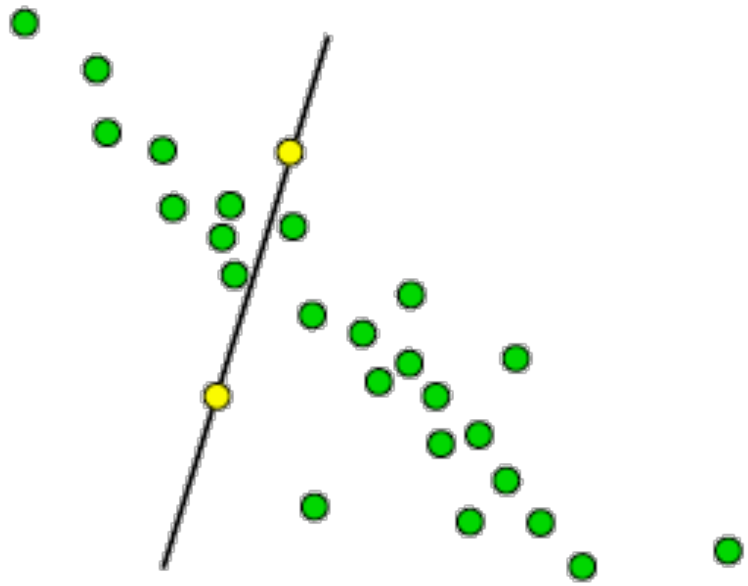


RANSAC procedure (line example)

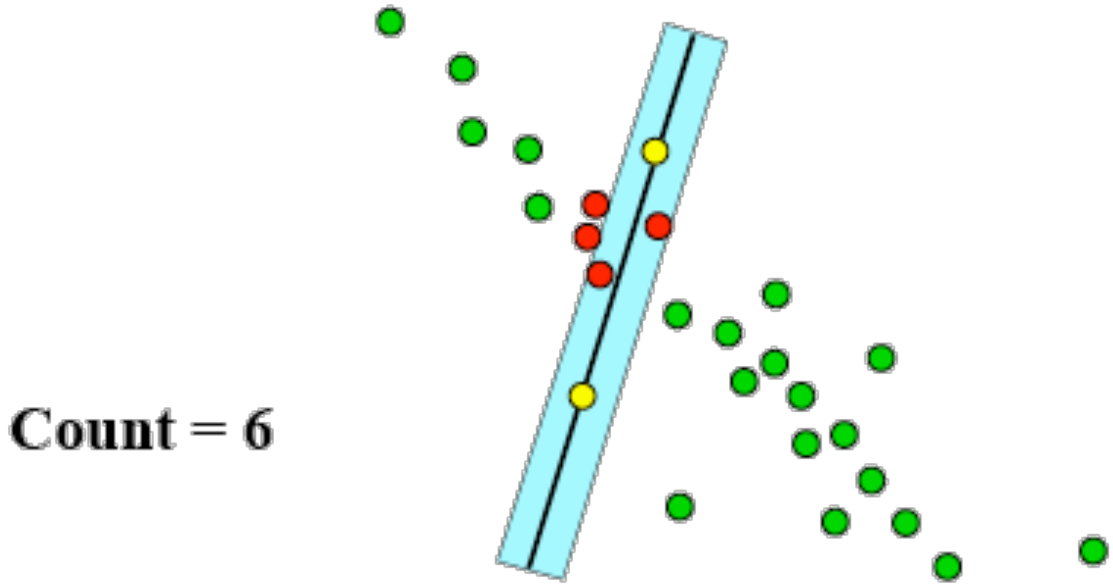


Count = 4

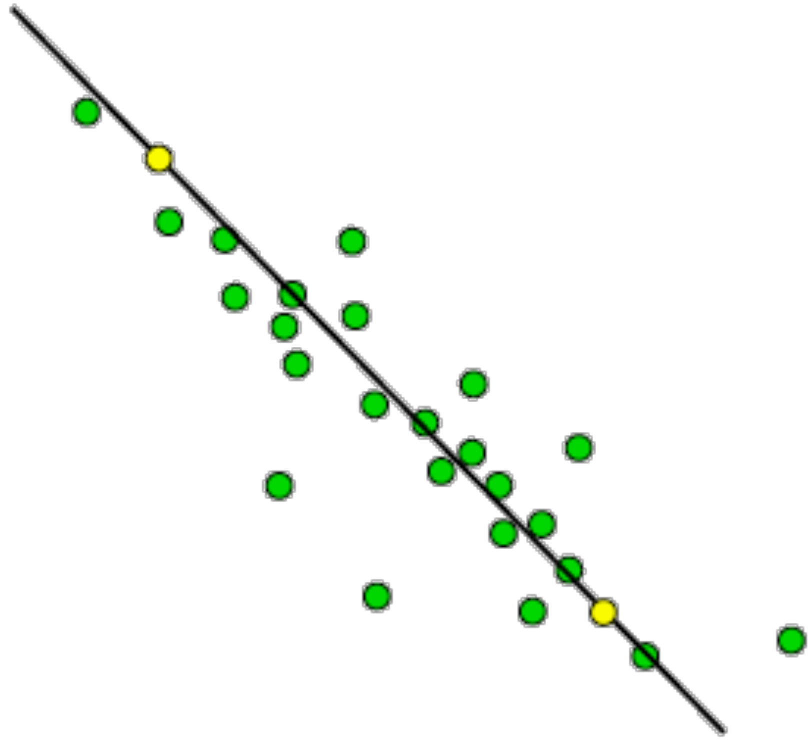
RANSAC procedure (line example)



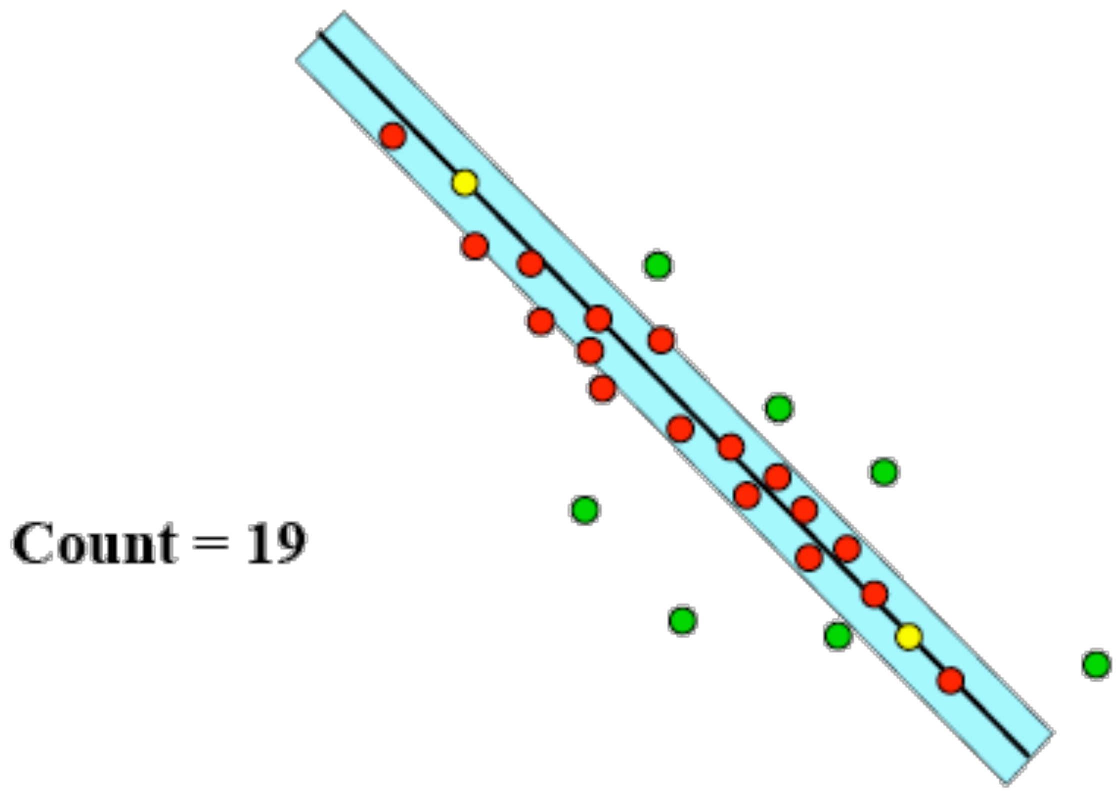
RANSAC procedure (line example)



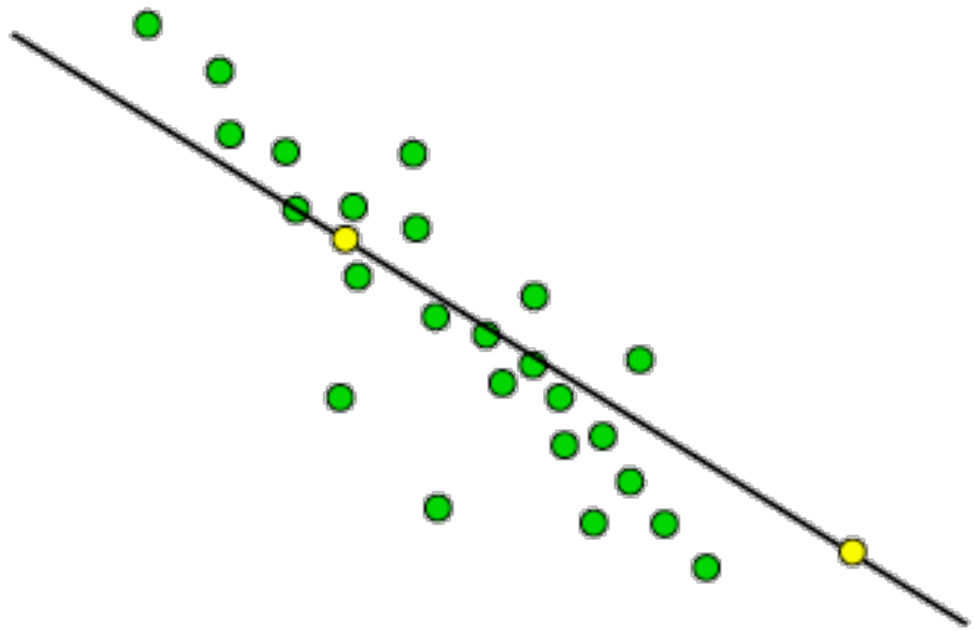
RANSAC procedure (line example)



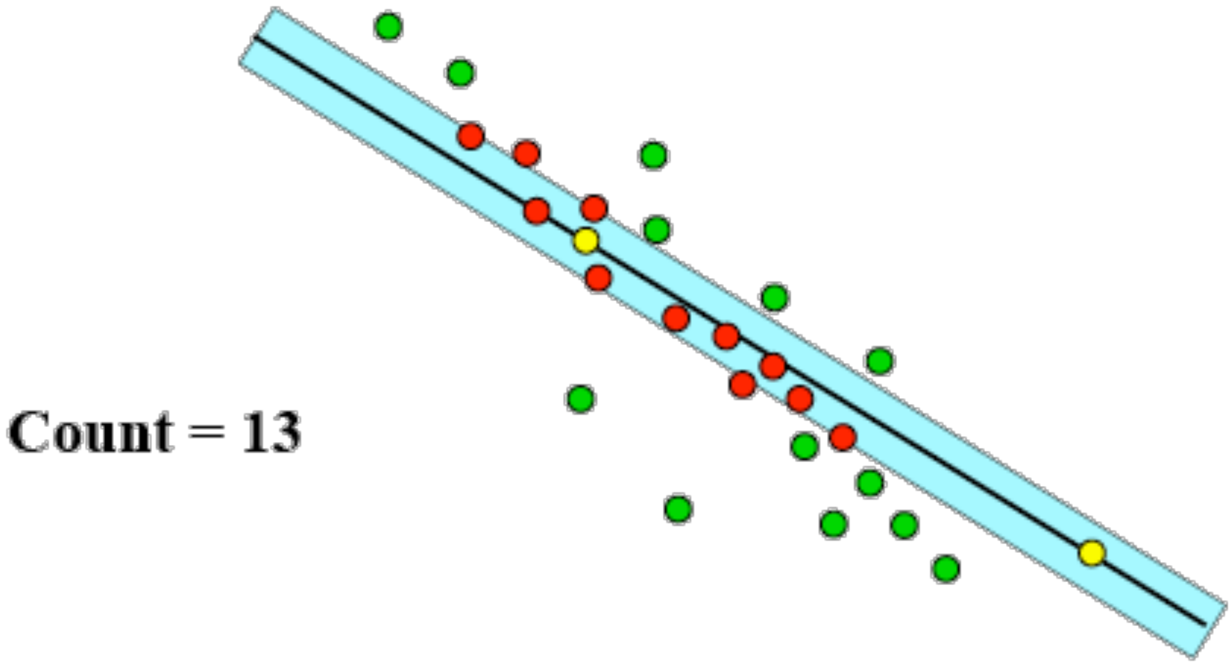
RANSAC procedure (line example)



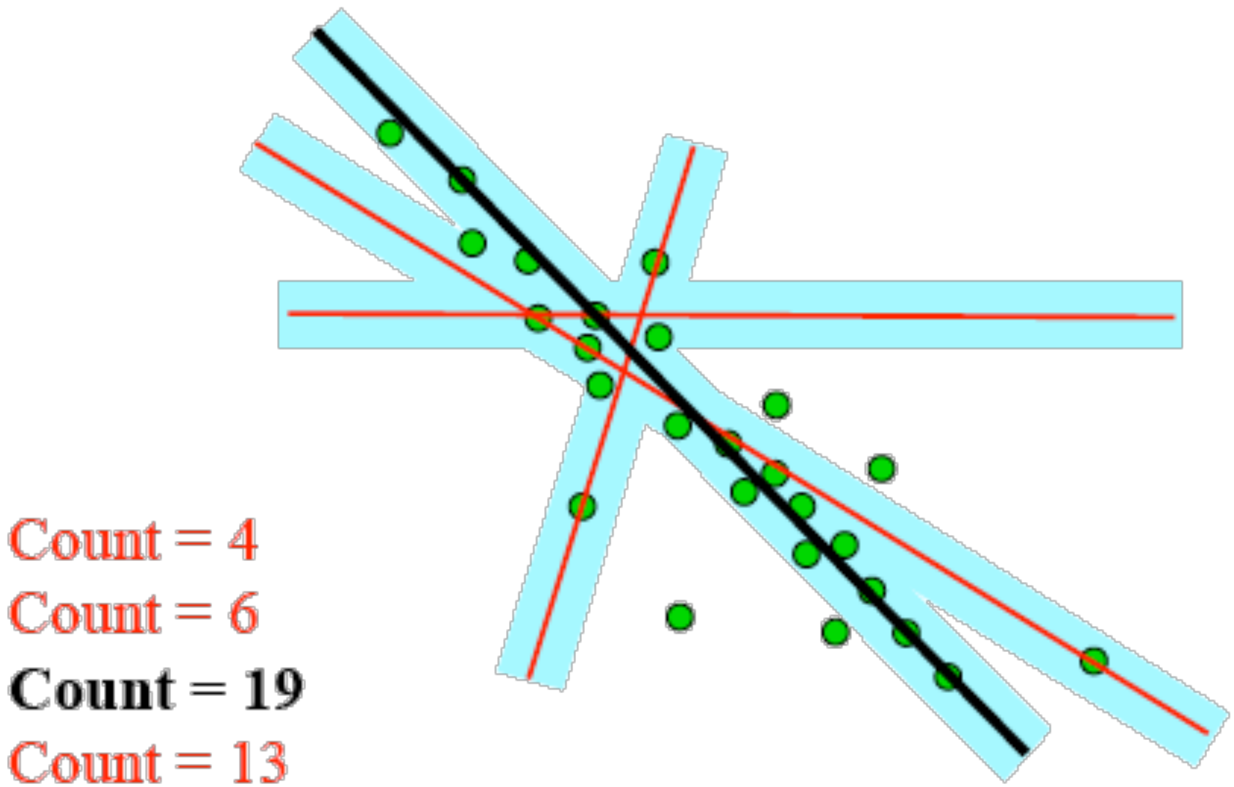
RANSAC procedure (line example)



RANSAC procedure (line example)



RANSAC procedure (line example)



RANSAC

Choose a small subset uniformly at random

Fit to that

Anything that is close to result is signal; all others are noise

Refit

Do this many times and choose the best

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?
  - One where the number of nearby points is so big it is unlikely to be all outliers

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

Forsyth & Ponce

How Many Samples to Choose?

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Let’s see why …

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Probability of choosing one inlier

Let's see why …

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Probability of choosing s inliers

Let's see why …

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Probability of that one or more points in the sample were outliers (contaminated sample)

Let's see why …

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Probability of that N Samples were contaminated

Let's see why …

- Probability that a point is an outlier: e
- Number of points in a sample: s
- Number of samples (we want to compute this): N
- Desired probability that we get a good sample: p

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

Probability of that AT LEAST one sample of N Samples was NOT contaminated

How Many Samples?

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

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

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

	proportion of outliers e						
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

Line example

12 pts: n = 12
Sample size: s = 2
Outliers 2: e=1/6 -> 20%
N = 5 gives 99% chance of getting a good sample
(trying every possible pair requires 66 trials!)

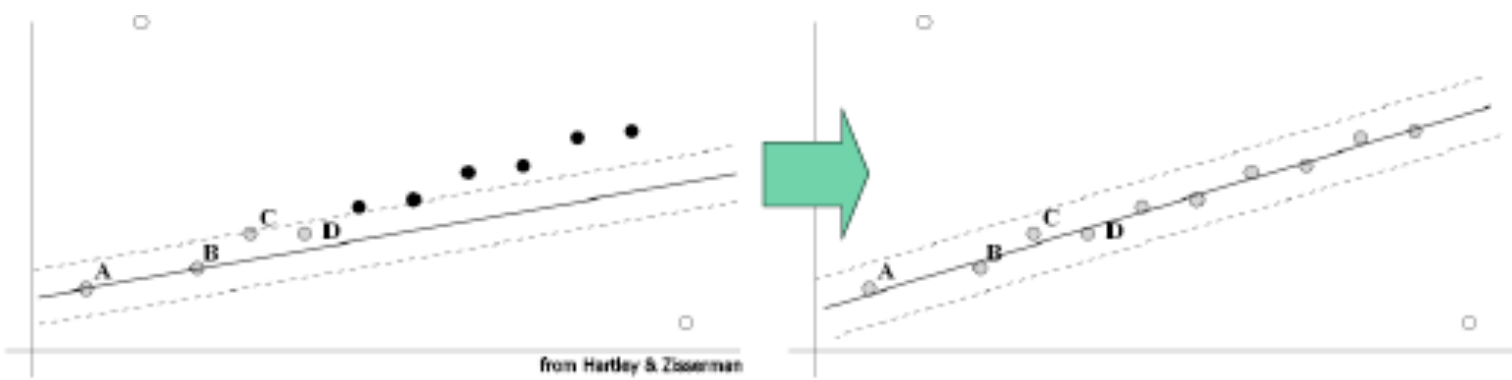
Acceptable Consensus Set

Typically, terminate when inlier ratio reaches expected ratio of inliers

$$T = (1 - e) \times \text{ total number of data points}$$

After RANSAC

- RANSAC divides the data into inliers and outliers
- But it computes the estimate with the MINIMAL samples
- We can improve the result by estimating the model using all inliers:
- After RANSAC: estimate once more!



Fitting curves other than lines

In principle, an easy generalization

The probability of obtaining a point, given a curve, is given by a negative exponential of distance squared

- In practice, rather hard
  - It is generally difficult to compute the distance between a point and a curve

Active Contours

Deformable Contours

- They are also called
  - Snakes
  - Active contours
- Think of a snake as an elastic band:
  - of arbitrary shape
  - sensitive to image gradient
  - that can wiggle in the image
  - represented as a necklace of points

Active Contour Models

An important class of algorithms to find boundaries

Usually does not use prior knowledge of the shape

Poses the problem as an “optimization” problem

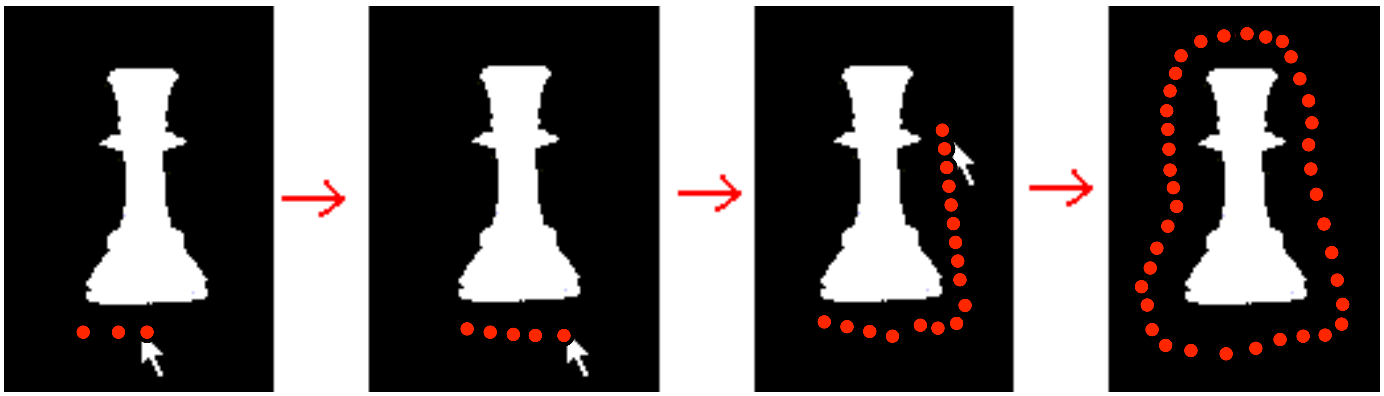
Introduction

How can we find the boundary of an object in an image?

- One approach could be:
  - Find edges
  - Link the edges
- Another possibility is to search for “smooth” boundaries:
  - The boundary should “match” the image
  - Can iteratively “improve”

Main Idea:

"Drop" a snake



Let the snake "wiggle", attracted by image gradient, until it glues itself against a contour

The Energy Functional

- Associate to each possible shape and location of the snake a value E.
  - Values should be s.t. the image contour to be detected has the minimum value.
  - E is called the energy of the snake.
- Keep wiggling the snake towards smaller values of E.

Energy Functional Design

We need a function that given a snake state, associates to it an Energy value E.

The function should be designed so that the snake moves towards the contour that we are seeking!

What moves the snake?

"Forces" applied to its points

Snake Energy

The total energy of the snake is defined as:

$$E_{total} = E_{internal} + E_{external}$$

The internal energy encourages smoothness
The external energy encourages closeness to edges

Forces moving the snake (External)

It needs to be attracted to contours:

- Edge pixels must “pull” the snake points.
- The stronger the edge, the stronger the pull.
- The force is proportional to $|\nabla I|$

Forces preserving the snake (Internal)

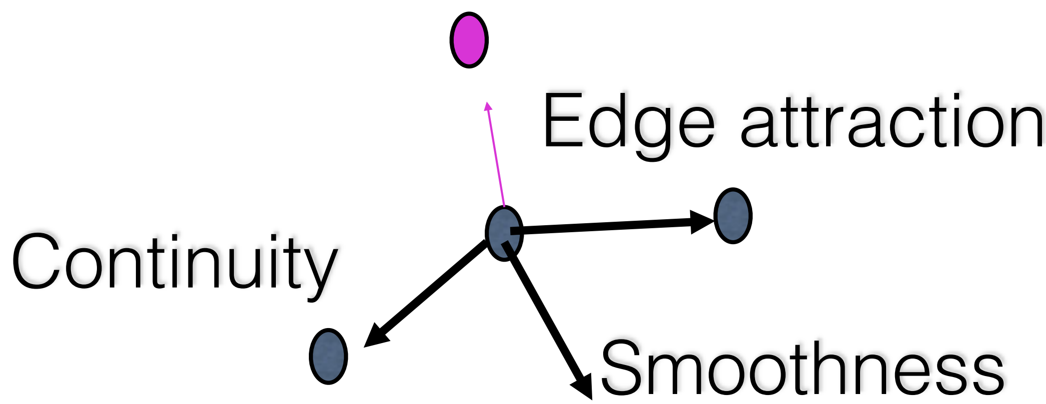
The snake should not break apart!

- Points on the snake must stay close to each other
- Each point on the snake pulls its neighbors
- The farther the neighbors, the stronger the force
- The force is proportional to the distance $|P_i - P_{i-1}|$

Forces preserving the snake (Internal)

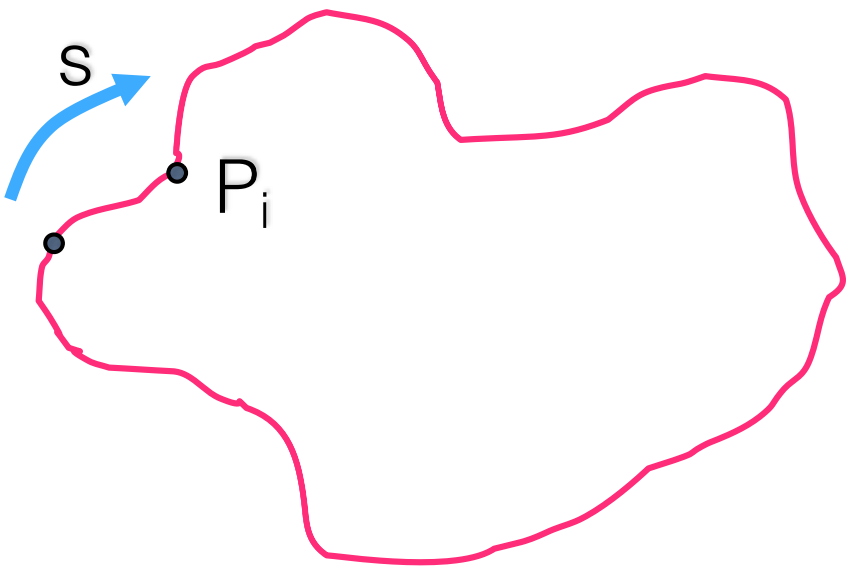
- The snake should avoid “oscillations”
  - Penalize high curvature
  - Force proportional to snake curvature

Snake Forces



Snake "state": Contour Parametrization

Consider a contour parametrization $c=c(s)$ where s is the "arc length"



Each point P_i on the contour has coordinates $(x_i(s),y_i(s))$

$$E = \int E_{int}(s) + E_{ext}(s)ds$$

Snake Energy Functional

Given a snake with N points $p_1, p_2, \ldots, p_N$

$$E = \sum_{i-1}^{N} a_i E_c(p_i) + b_i E_s(p_i) + c_i E_g(p_i)$$

"Continuity" "Smoothness" "Edgeness"

a_i, b_i, c_i are "weights" to control influence

Continuity Term

Given a snake with N points $p_1, p_2, \ldots, p_N$

Let d be the average distance between points

Distance between points should be kept close to average

Define the continuity term of the Energy Functional:

$$E_c(p_i) = (d - |p_i - p_{i-1}|)^2$$

$$p_i = [x_i \; y_i]$$

$$E_c = \left(d - \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right)^2$$

Smoothness Term

Given a snake with N points $p_1, p_2, \ldots, p_N$

Curvature should be kept small

Define the smoothness term of the Energy Functional:

$$E_s(p_i) = \underbrace{|p_{i-1} - 2p_i + p_{i+1}|^2}_{\text{Second derivative}}$$

$$E_s = (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2$$

Edgeness Term

Given a snake with N points $p_1, p_2, \ldots, p_N$

Define the edgeness term of the Energy Functional:

$$E_g(p_i) = -|\nabla I(p_i)|$$

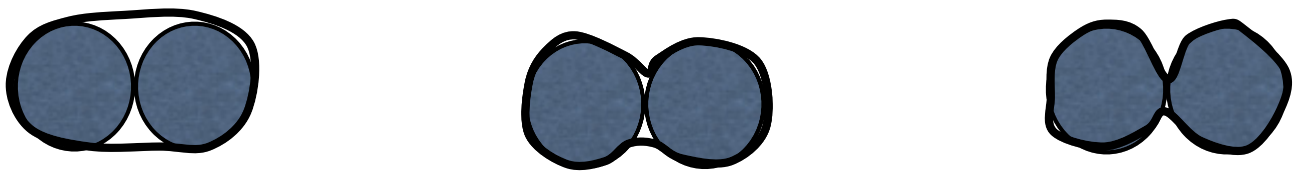
$$\nabla I(p_i) = [G_x(p_i)\ G_y(p_i)]$$

$$|\nabla I(p_i)| = \sqrt{G_x(p_i)^2 + G_y(p_i)^2}$$

Magnitude of the gradient should be LARGE

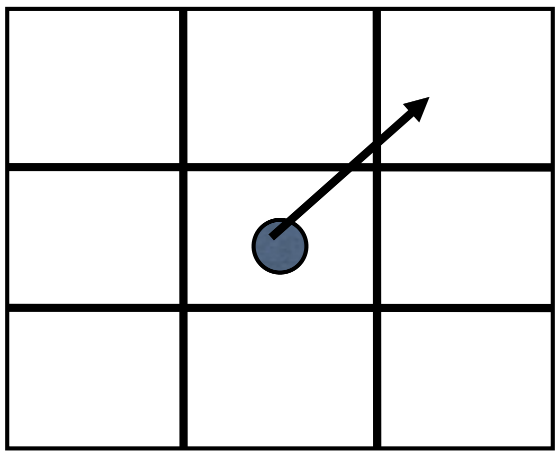
Relative Weighting

The weights control the smoothness and stiffness of the snake



Greedy Algorithm

Each point moves within a small window to minimize the energy



Compute the new energy for each candidate location
Move the point to the one with the minimum value

Keeping corners …

Before starting a new iteration:

- Search for “corners”:
  - max curvature
  - large gradient
- Corner points should not contribute to the energy (set $b_i = 0$)

Implementation Considerations

To avoid numerical problems, the terms of the energy function should be normalized.

- E_c and E_s are normalized by their maximum in the neighborhood
- E_g is normalized as $|\nabla I - m| / (M - m)$
  - M and m are the max and min value of the gradient magnitude in the neighborhood

Snake Algorithm

Input:

gray scale image I

a chain of points $p_1, p_2, \ldots, p_N$

f is the fraction of points that must move to start a new iteration

$U(p)$ is a neighborhood around p

d is the average distance between snake points.

Snake Algorithm

1. While the fraction of moved points > f
   1. For i=1,2,…,N
      1. find a point in $U(p_i)$ s.t. the energy is minimum,
      2. move p_i to this location
   2. For i=1,2,…,N
      1. Estimate the curvature $k=|p_{i-1}-2p_i+p_{i+1}|$
      2. Look for local max, and set $b_{max} = 0$
   3. Update d

Problems with Snakes

Smoothness does not always capture all prior knowledge

User must define the weights

Snakes might oversmooth boundaries

Not trivial to prevent curve self intersecting