



LUND  
UNIVERSITY

350

# Image Analysis (FMAN20)

## Lecture 12, 2019

MAGNUS OSKARSSON

5 Decker-München

COMPU

2.9

4

8

11

16

22





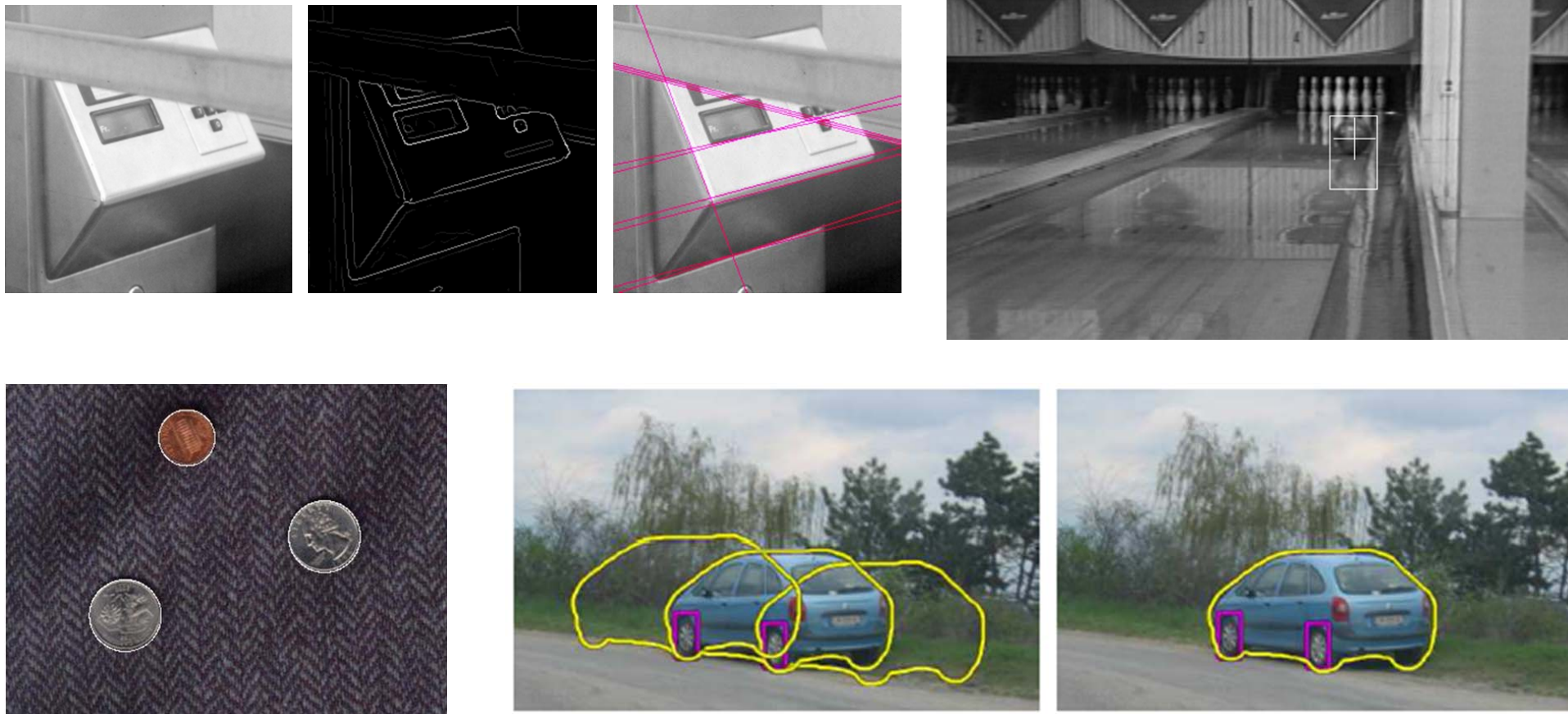
# Image Analysis - Motivation

# Overview – Parameter Estimation, fit

1. Voting, The Hough Transform
2. Fitting one line, least squares, total least squares, svd
3. Curve Fitting
4. Robust Fitting
  1. M-estimator
  2. RANSAC
5. Fitting Multiple Objects
  1. K-means
  2. Remove and fit again

# Fitting

We want to **associate** a **model** with **observed features**



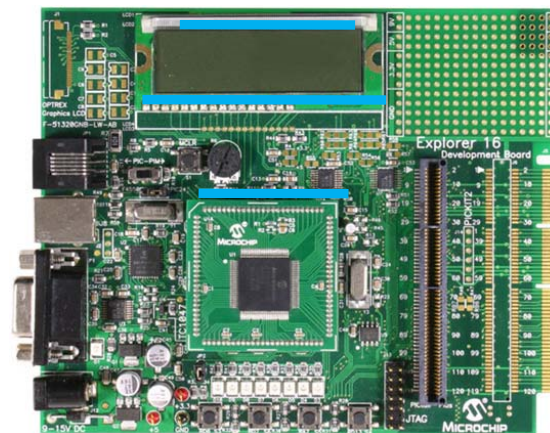
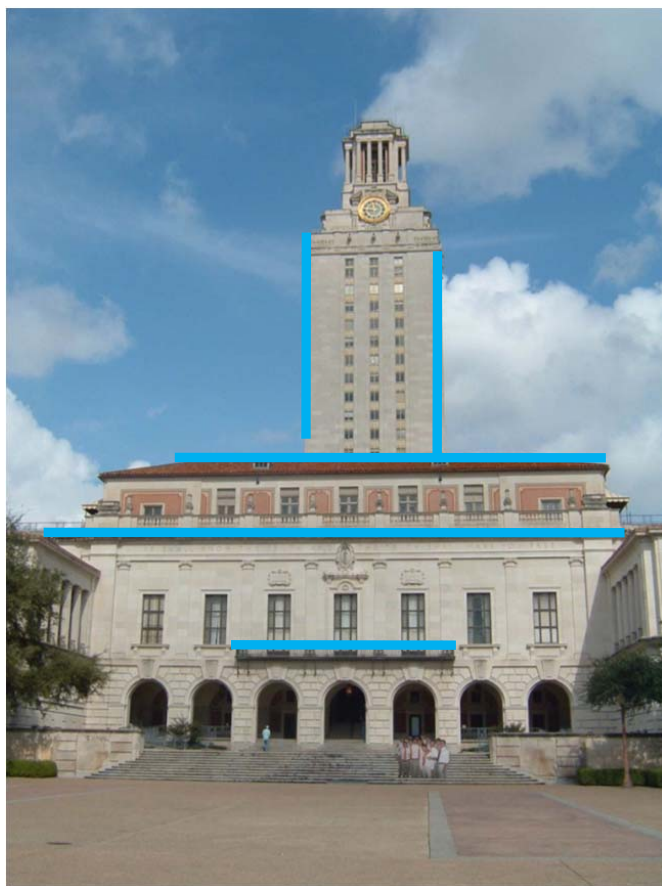
[Fig from Marszalek & Schmid, 2007]

For example, the model could be a line, a circle, or an arbitrary shape. We use a model to produce compact representations that capture the relevant image structures we seek.

# Example: Line fitting

*Why fit lines?*

Many objects characterized by presence of straight lines

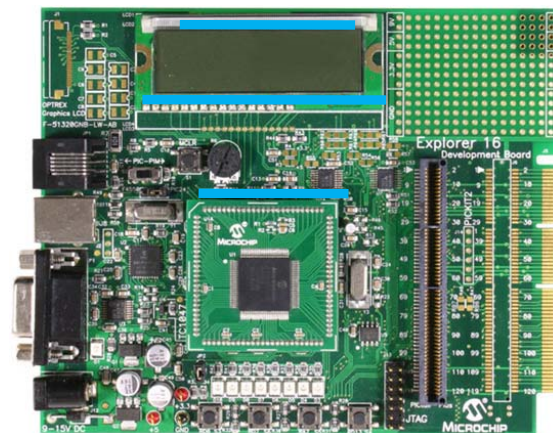




# Example: Line fitting

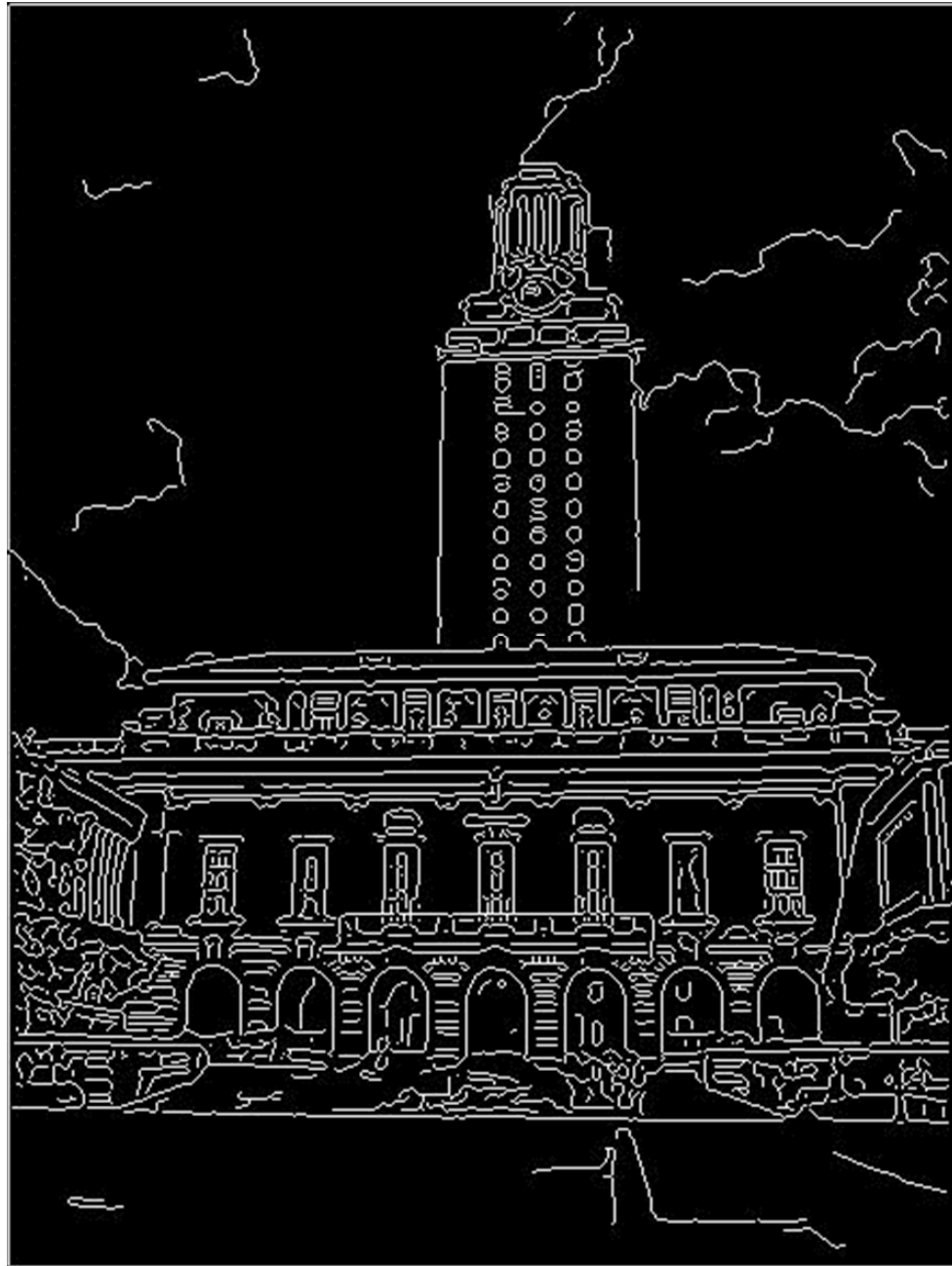
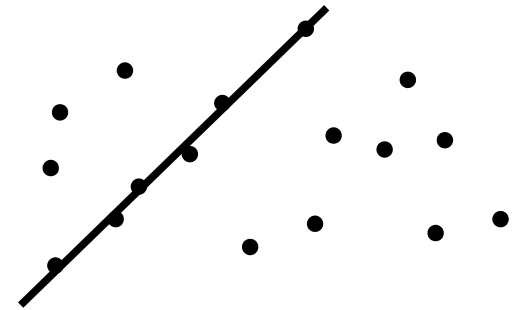
*Why fit lines?*

Many objects characterized by presence of straight lines



Why aren't we done just by running edge detection?

# Difficulty of line fitting



- **Extra** edge points (clutter), multiple models:
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing**:
  - how to find a line that bridges missing evidence?
- **Noise** in measured edge points, orientations:
  - how to detect true underlying parameters?

# Line Fitting as Study Problem

- Fitting involves determining what possible structures could have given rise to a set of tokens in an image. For example, we might have a set of edge points (the tokens) and wish to determine which lines fit them best. There are three increasingly more general problems that occur in fitting:
- 1. Parameter estimation: Assume we know which tokens came from particular structure, and we want to know what the parameters of the structure are.
  - For example, we might have a set of edge points, all of which are known to have come from a line, and we wish to know what line they came from.
  - Most interesting case is when criterion is not local – cannot tell whether a set of points lies on a line by looking only at each point and the next.



# Line Fitting as Study Problem

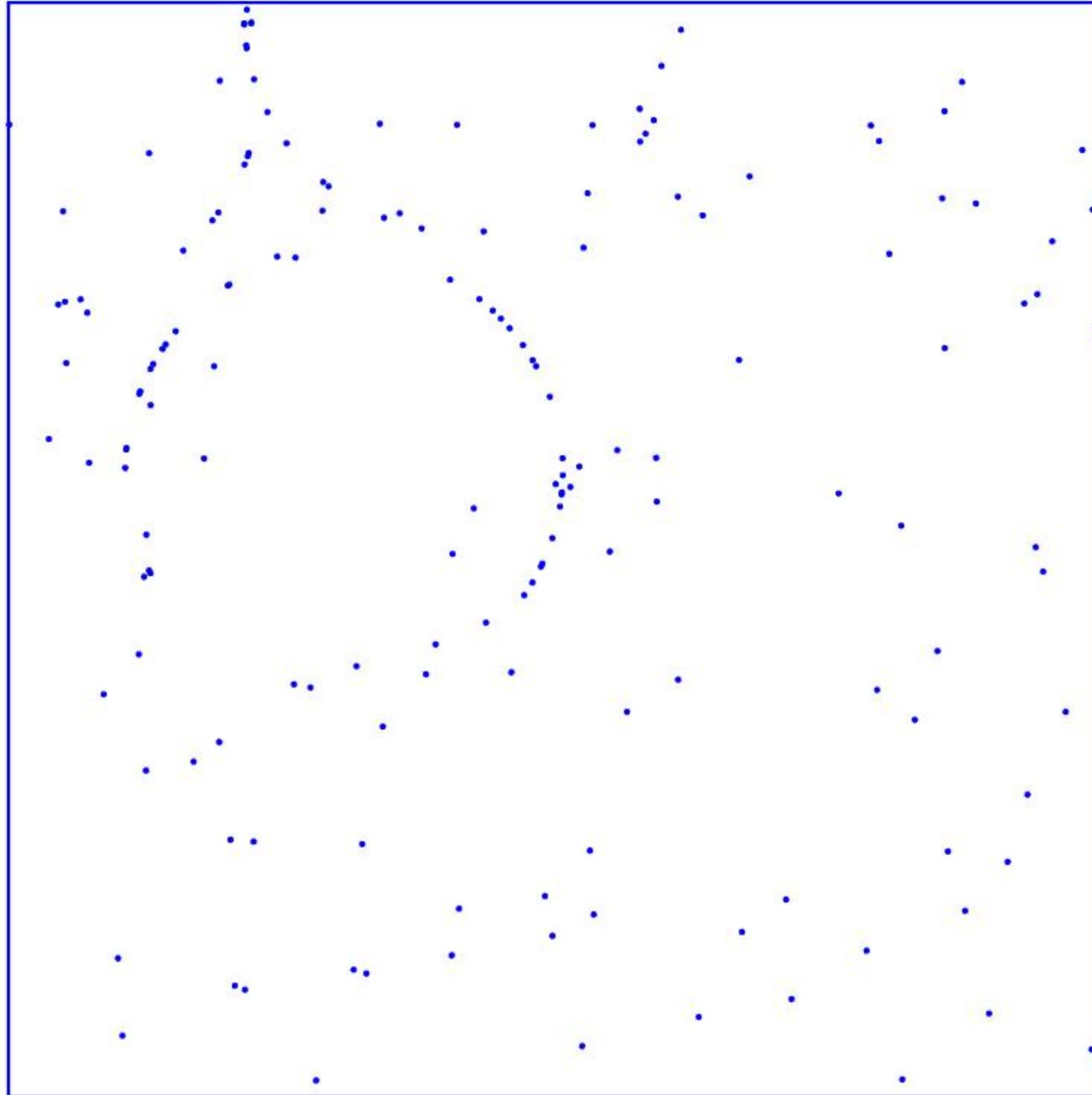
- 2. Data Association: Assume we know how many structures are present, and we wish to determine which tokens came from which structure.
  - For example, we might have a set of edge points, and we need to know the best set of lines fitting these points; this involves (1) determining which points belong together on a line and (2) figuring out what each line is.
  - Generally, these problems are not independent (because one good way of knowing whether points belong together on a line is checking how well the best fitting line approximates them).

# Line Fitting as Study Problem

- 3. Model Selection: We would like to know (1) how many structures are present (2) which points are associated with which structure and (3) what the structures are.
- For example, given a set of edge points, we might want to return a set of lines that fits them well.
- This is, in general, a substantially difficult problem the answer to which depends strongly on the type of model adopted (for example, we could simply pass a line through every pair of edge points – this gives a set of lines that fit extremely well, but will likely be a poor representation).



# What do you see?



# Overview – Parameter Estimation, fit

1. **Voting, The Hough Transform**
2. Fitting one line, least squares, total least squares, svd
3. Curve Fitting
4. Robust Fitting
  1. M-estimator
  2. RANSAC
5. Fitting Multiple Objects
  1. K-means
  2. Remove and fit again



# Voting

- It is not easy to check all combinations of features by fitting a model to each possible subset (but see later RANSAC)
- **Voting** is a general technique where we let the features vote for all models that are compatible with them
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive a lot of votes.
- Noise and clutter features will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features. Hence they will have less support.
- Ok if some features not observed, as model can span multiple fragments. The voting process is distributed (each individual feature casts a vote for a possible model, so they can work independently)

# Hough Transform

- Goal: Finding linear structures in images
- Used on edge data

$$l_{a,b} : \quad ax + by = 1 \quad (\text{Assume } 0 \notin l)$$

$$(x_k, y_k) \in l_{a,b} \quad \Leftrightarrow \quad ax_k + by_k = 1$$

- Study the set

$$\Lambda_k = \{ (a, b) \mid (x_k, y_k) \in l_{a,b} \}$$

- This forms a line in the  $ab$ -plane.



# Other line representations

1) Represent lines as

$$y = kx + l.$$

Each line is a point in the  $kl$ -plane.  $k$  is the slope of the line,  $l$  is the  $y$ -intercept.

Vertical lines (undefined slope/gradient) cannot be represented.

# Other line representations

**2)** Represent the lines as

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

Each line is a point in the  $\rho\theta$ -plane. Represent all lines.  
Each point gives a sine-formed curve in the  $\rho\theta$ -plane.

# Hough transform algorithm

---

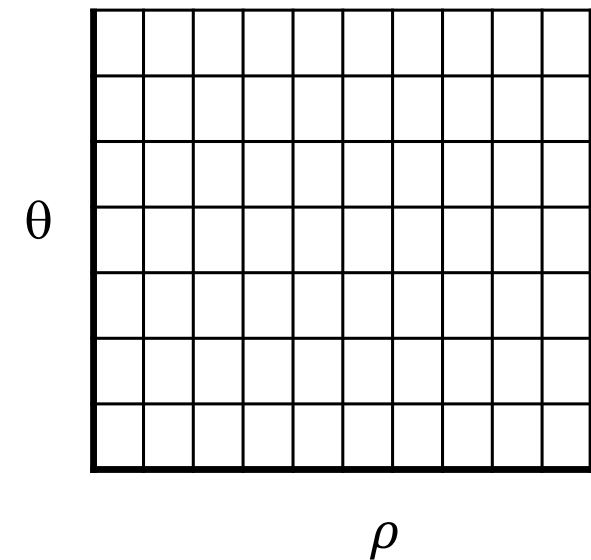
Using the polar parameterization:

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

## Basic Hough transform algorithm

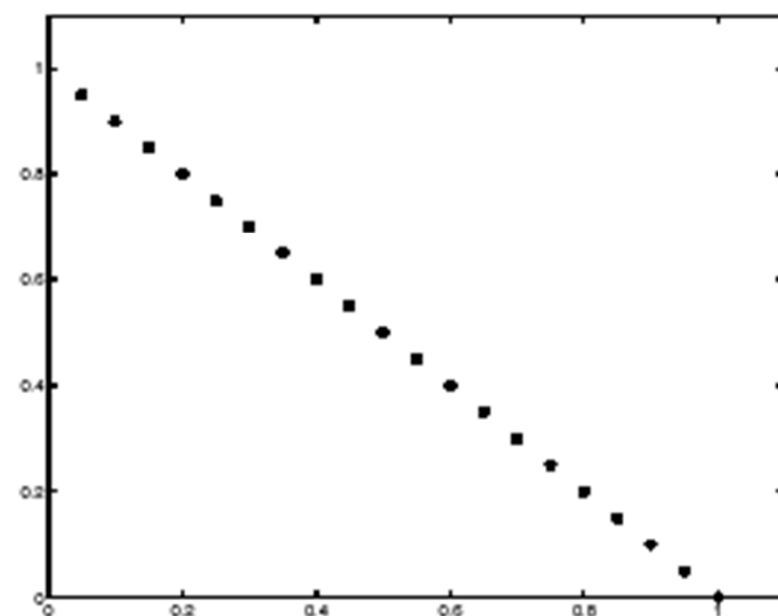
1. Initialize  $H[\rho, \theta] = 0$
2. for each edge point  $I[x, y]$  in the image  
for  $\theta = 0$  to  $180$  // some quantization  
 $\rho = x \cos \theta + y \sin \theta$   
 $H[\rho, \theta] += 1$
3. Find the value(s) of  $(\rho, \theta)$  where  $H[\rho, \theta]$  is maximum
4. The detected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

H: accumulator array (votes)

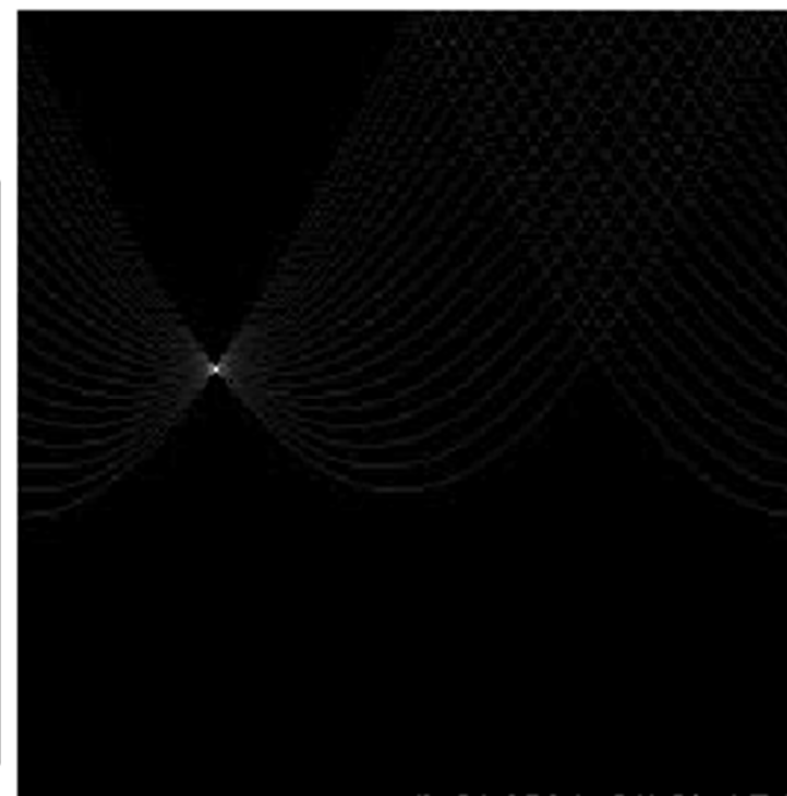


Complexity (in terms of number of votes)?





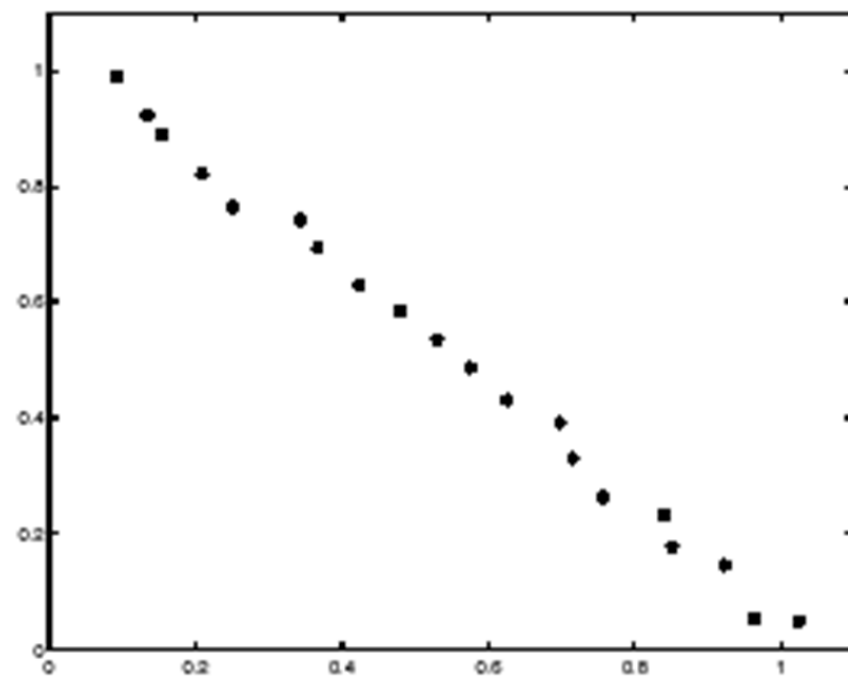
tokens



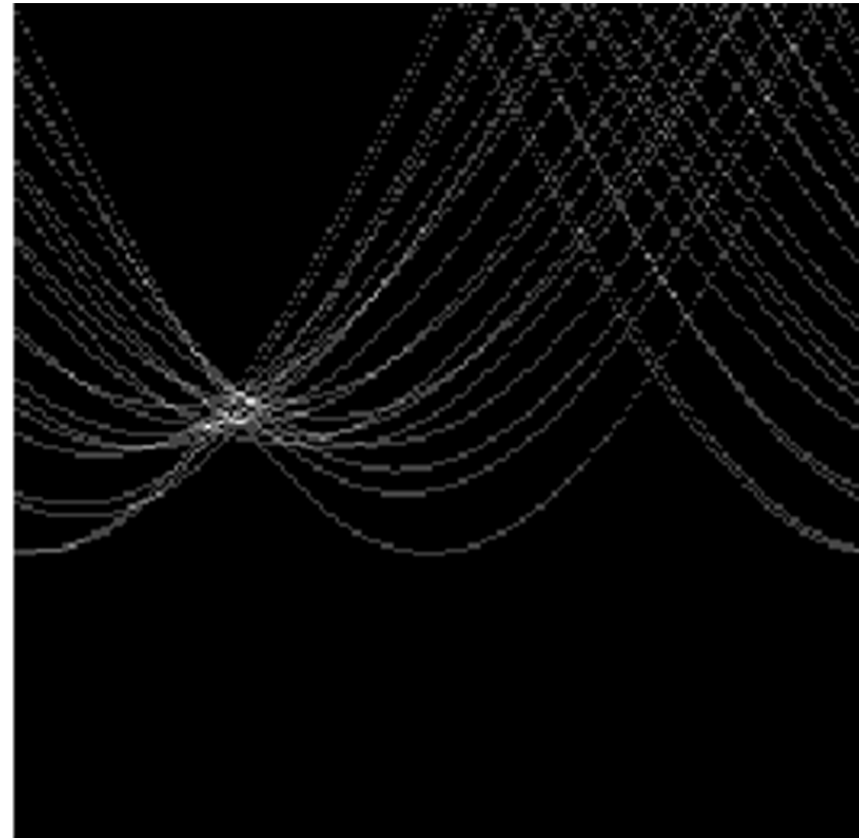
$\theta$

$\rho$

Note that most points in the vote array are very dark, because they get only one vote.

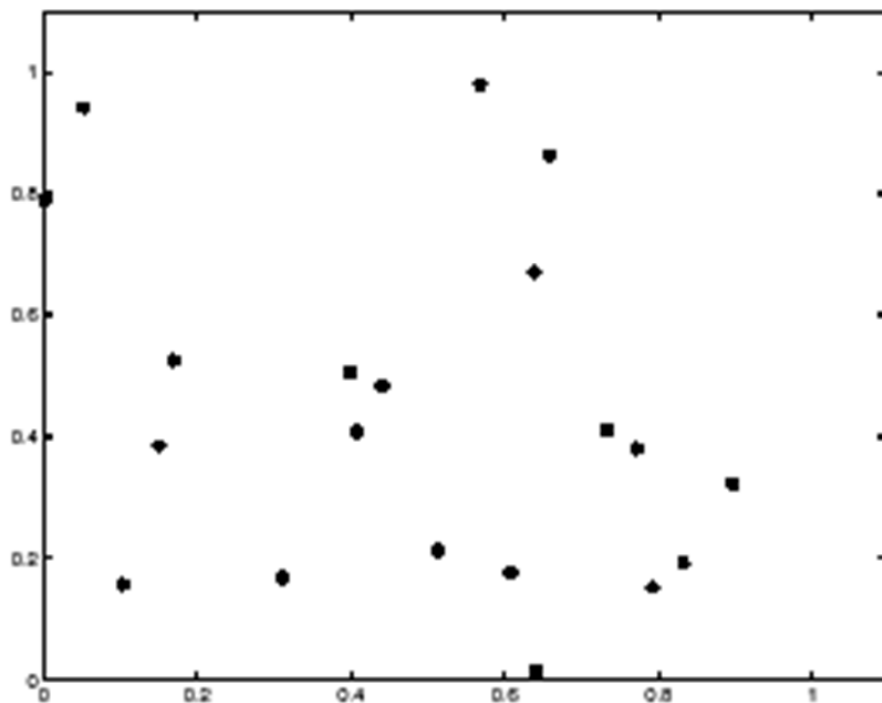


tokens

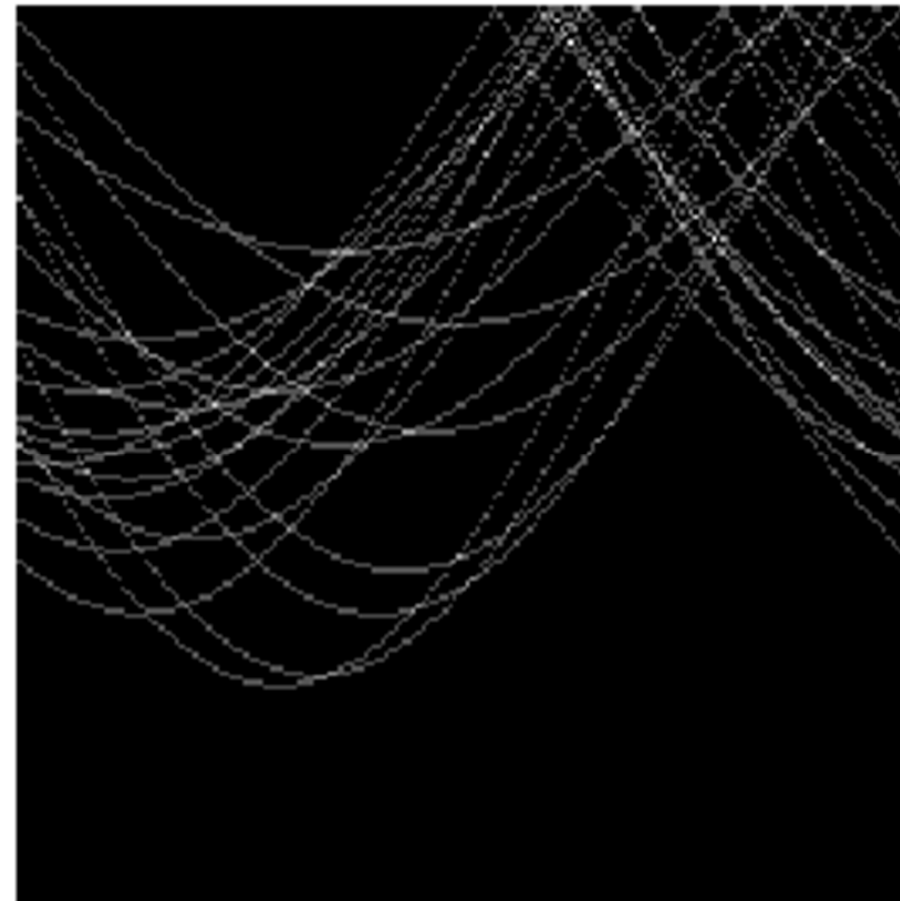


votes

What difficulty does this present for an implementation?



tokens



votes

Here, everything appears to be “noise”, or random edge points, but we still see peaks in the vote space.



# Extensions

---

Extension 1: Use the image gradient

1. same
2. for each edge point  $I[x,y]$  in the image

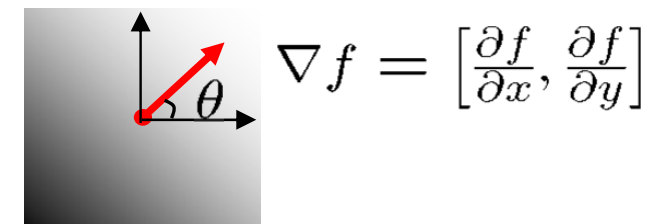
$\theta = \text{gradient at } (x,y)$

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

$$H[\rho, \theta] += 1$$

3. same
4. same

(Reduces degrees of freedom)



$$\theta = \tan^{-1} \left( \frac{\partial f / \partial y}{\partial f / \partial x} \right)$$

# Extensions

---

## Extension 1: Use the image gradient

1. same
2. for each edge point  $I[x,y]$  in the image  
compute unique  $(\rho, \theta)$  based on image gradient at  $(x,y)$   
 $H[\rho, \theta] += 1$
3. same
4. same

(Reduces degrees of freedom)

## Extension 2

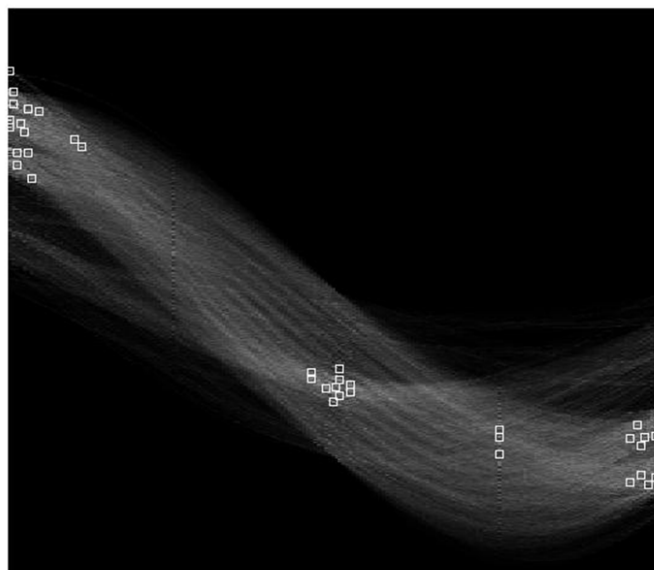
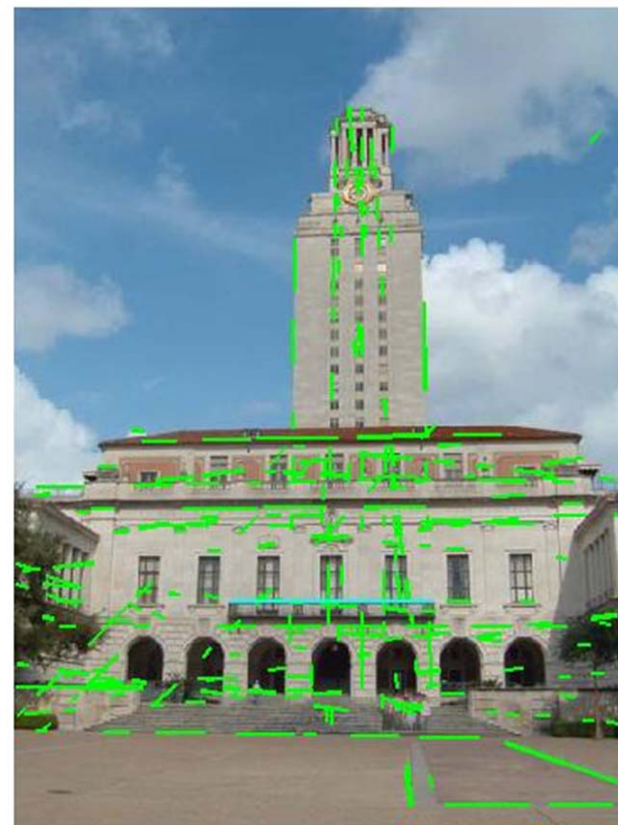
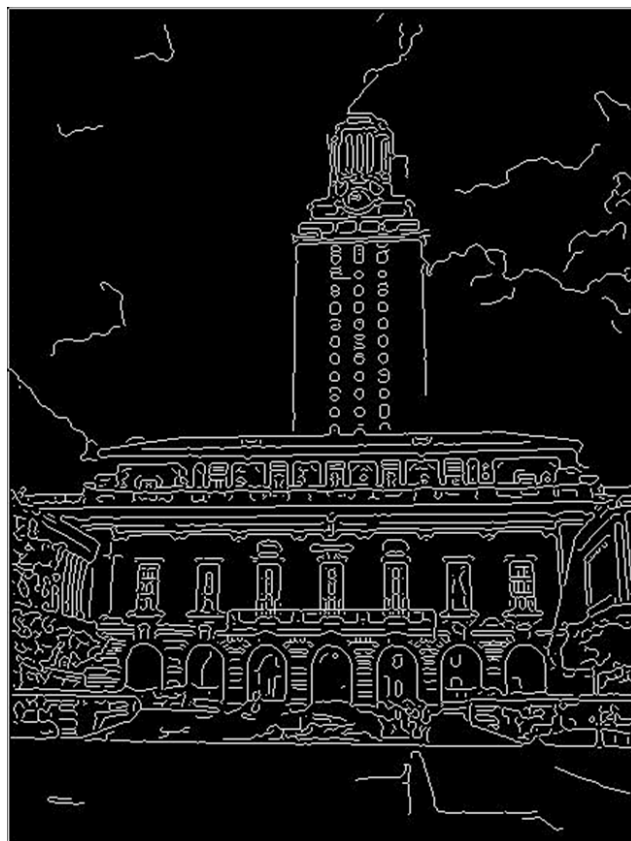
- give more votes for stronger edges (use magnitude of gradient)

## Extension 3

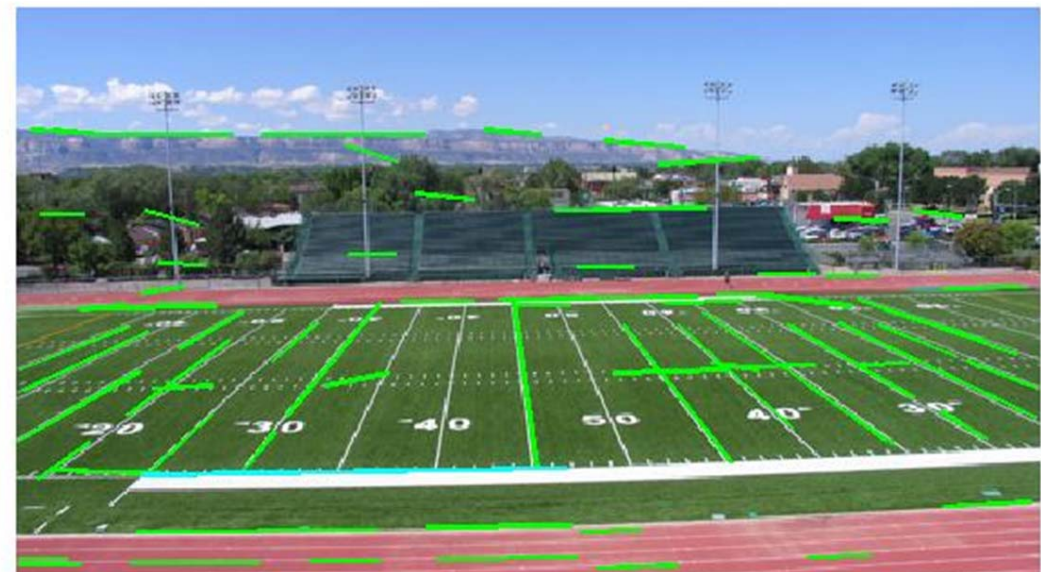
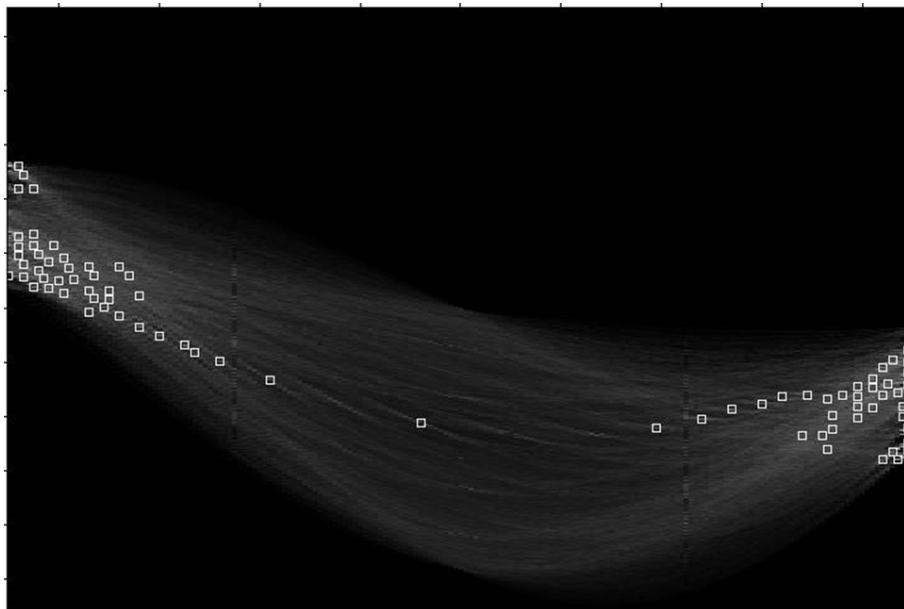
- change the sampling of  $(\rho, \theta)$  to give more/less resolution

## Extension 4

- The same procedure can be used with circles, squares, or any other shape



2



Showing longest segments found



# Voting: practical tips

- Minimize irrelevant tokens first (take edge points with significant gradient magnitude)
- Choose a good grid / discretization
  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
- Vote for neighbors, also (smoothing in accumulator array)
- To read back which points voted for “winning” peaks, keep tags on the votes.

# 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

# Hough Transform

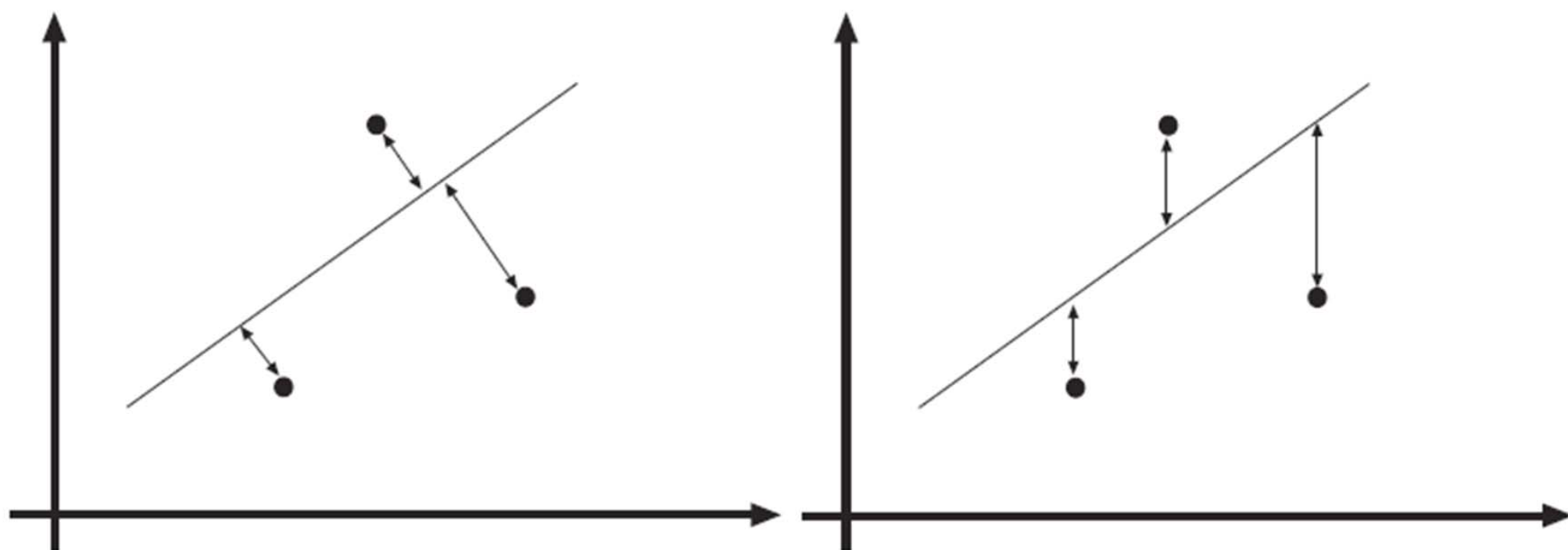
- ▶ Construct an array representing  $[\rho, \theta]$ . For each point, render the curve  $\rho\theta$  into this array, adding one at each cell.
- ▶ *Difficulties:* **How big should the cells be?** Too big, and we cannot distinguish between quite different lines; too small, and noise causes lines to be missed.
- ▶ **How many lines?** Count the peaks in the Hough array.
- ▶ **Who belongs to which line?** Tag the votes.

Problems with noise and cell size can defeat it

# Overview – Parameter Estimation, fit

1. Voting, The Hough Transform
2. **Fitting one line, least squares, total least squares, svd**
3. Curve Fitting
4. Robust Fitting
  1. M-estimator
  2. RANSAC
5. Fitting Multiple Objects
  1. K-means
  2. Remove and fit again





**Figure 16.6. Left:** Perpendicular least squares models data points as being generated by an abstract point along the line to which is added a vector perpendicular to the line, with a length given by a zero mean, Gaussian random variable. This means that the distance from data points to the line has a normal distribution. By setting this up as a maximum likelihood problem, we obtain a fitting criterion that chooses a line that minimizes the sum of distances between data points and the line. **Right:** Least squares follows the same general outline, but assumes that the error appears only in the  $y$ -coordinate. This yields a (very slightly) simpler mathematical problem, at the cost of a poor fit.

# The least squares method

## Line fitting

Assume that the points  $(x_i, y_i)$  are measured. Then

$$y_i = kx_i + l$$

for line parameters  $(k, l)$ .

Assume that:

- ▶ that the errors are only in the  $y$ -direction.
- ▶ the line is not vertical (since we are counting only vertical offsets from the line as errors, near vertical lines lead to quite large values of the error)

# Line Fitting

Then

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} k \\ l \end{pmatrix} + \mathbf{n} = Ap + \mathbf{n}$$

If the errors  $\mathbf{n}$  are independent and Gaussian distributed, then it is reasonable to solve  $y = Ap$  in least squares sense, i.e. minimizing  $|y - Ap|$ .

# Line fitting

This least squares problem was studied in Lecture 2 (and in other courses).

The solution is

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

Write this out to obtain

$$\begin{pmatrix} k \\ l \end{pmatrix} = \begin{pmatrix} \bar{x}^2 & \bar{x} \\ \bar{x} & N \end{pmatrix}^{-1} \begin{pmatrix} \bar{xy} \\ \bar{y} \end{pmatrix}$$

# Least squares in Matlab

In matlab the least squares solution can be obtained using the slash function

$$p = A \backslash y$$

Read the help text 'help slash' for more information about how the 'slash'-operator works.



# Total least squares

- ▶ One problem with the idea above lies in the two assumptions.
- ▶ It cannot handle vertical lines.
- ▶ For lines that are close to vertical the assumption that the errors are only in the y-direction gives sub-optimal estimates of the line.
- ▶ It is better to minimize the distance between the point and the line.

# Distance between point and line

From linear algebra we know that the distance between the point

$$(x, y)$$

and the line

$$ax + by + c = 0$$

with line parameters

$$l = (a, b, c)$$

is

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

# Optimization

Assume that

$$a^2 + b^2 = 1$$

then the distance is

$$d = |ax + by + c|.$$

The line  $l = (a, b, c)$  that minimizes the sum of squares of the distance is given by

$$\min_{a,b,c,a^2+b^2=1} f(a, b, c) = \sum_i (ax_i + by_i + c)^2 .$$

# Solving the optimization problem

The Lagrange function is

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

whose stationary points is given by (denote e.g.  $\bar{x}^2 \leftarrow \sum_i x_i^2$ )

$$\begin{pmatrix} \bar{x}^2 & \bar{x}\bar{y} & \bar{x} \\ \bar{x}\bar{y} & \bar{y}^2 & \bar{y} \\ \bar{x} & \bar{y} & N \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \\ 0 \end{pmatrix}$$

# Solution

We can solve for  $c$  using the last equation:

$$c = -\frac{1}{N}(a\bar{x} + b\bar{y})$$

Then we can substitute  $c$  in the equations above to obtain

$$\begin{pmatrix} \bar{x}^2 - \frac{1}{N}\bar{x}\bar{x} & \bar{x}\bar{y} - \frac{1}{N}\bar{x}\bar{y} \\ \bar{x}\bar{y} - \frac{1}{N}\bar{x}\bar{y} & \bar{y}^2 - \frac{1}{N}\bar{y}\bar{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \end{pmatrix}$$



# Solution

$$\begin{pmatrix} \bar{x}^2 - \frac{1}{N}\bar{x}\bar{x} & \bar{x}\bar{y} - \frac{1}{N}\bar{x}\bar{y} \\ \bar{x}\bar{y} - \frac{1}{N}\bar{x}\bar{y} & \bar{y}^2 - \frac{1}{N}\bar{y}\bar{y} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \lambda \begin{pmatrix} a \\ b \end{pmatrix}$$

is an eigenvalue problem.

There are two solutions, up to scale. Can be obtained in closed form.

The two solutions are orthogonal. One maximizes the likelihood, the other minimizes it.

It is straightforward to test which one of the two minimize  $f(a, b, c)$ .

# Overview – Parameter Estimation, fit

1. Voting, The Hough Transform
2. Fitting one line, least squares, total least squares, svd
3. **Curve Fitting**
4. Robust Fitting
  1. M-estimator
  2. RANSAC
5. Fitting Multiple Objects
  1. K-means
  2. Remove and fit again

# Curve fitting

Similar ideas can be used to fit conics to points

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

or even higher order algebraic curves.

# Overview – Parameter Estimation, fit

1. Voting, The Hough Transform
2. Fitting one line, least squares, total least squares, svd
3. Curve Fitting
4. **Robust Fitting**
  1. M-estimator
  2. RANSAC
5. Fitting Multiple Objects
  1. K-means
  2. Remove and fit again

# Inference

Assumes that we measure random points  $(x, y)$  along a line  $(a, b, c)$  with an error in the normal direction  $(a, b)$  that is Gaussian distributed.

Then the logarithm of the likelihood function is

$$\frac{1}{2\sigma^2} \sum_i (ax_i + by_i + c)^2$$

with constraint  $a^2 + b^2 = 1$ , and  $\sigma$  denoting the standard deviation of the noise.

# Problems with inference

Some practical problems:

- ▶ Robustness - often there are points there that do not belong to the object.
- ▶ There might be missing data
- ▶ It is difficult to establish correct correspondence between points and objects.

How can one make the fitting less sensitive to such errors?



# Outliers: M-estimators

A common method is to use an error function which is quadratic for small errors, but large for larger errors.

Then large errors (outliers) will not affect the fitting as much.

Instead of minimizing

$$\sum_i (ax_i + by_i + c)^2$$

we minimize

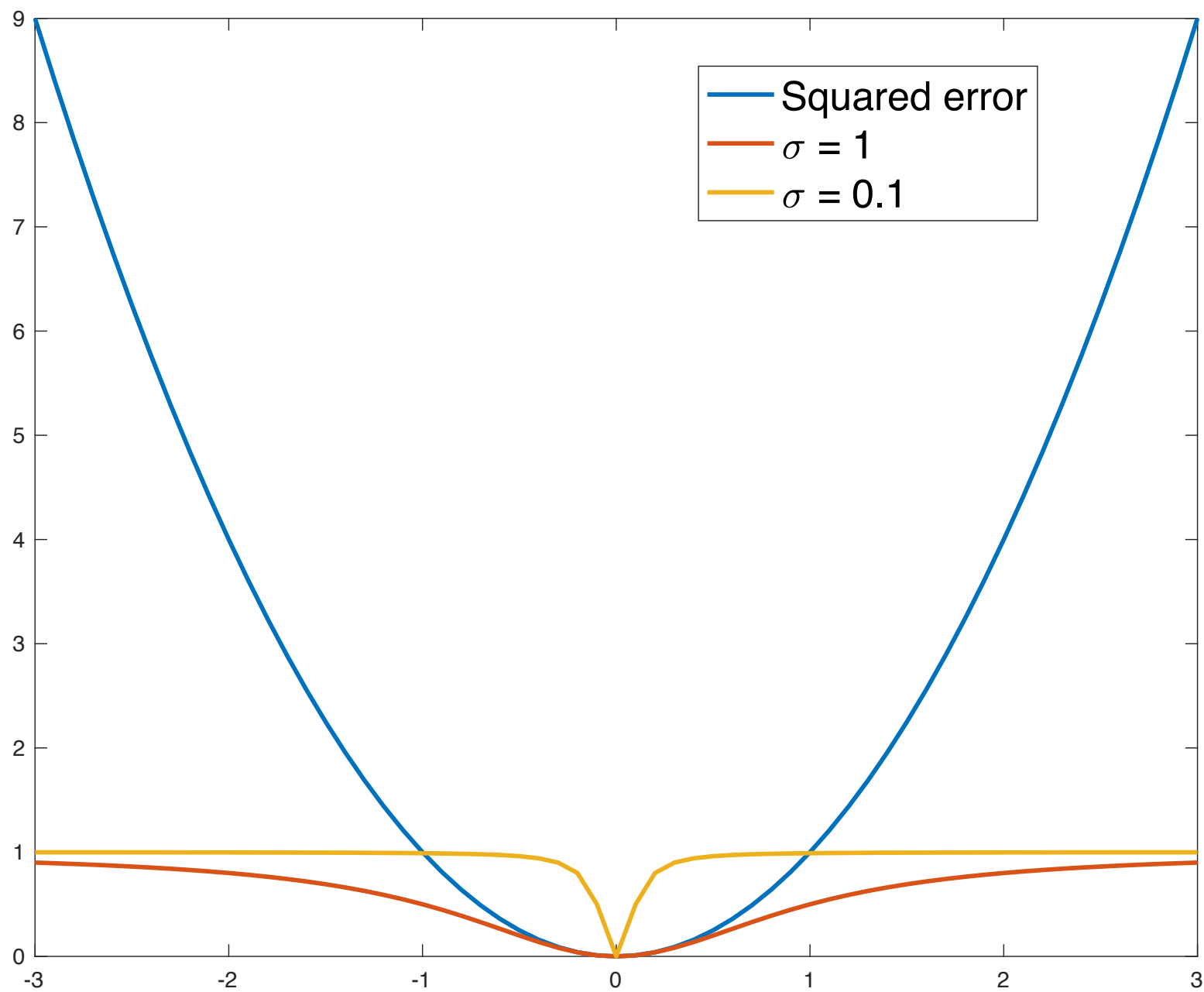
$$\sum_i \rho(ax_i + by_i + c, \sigma)$$

where e.g. one could use

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

- - - - -

# Outliers: M-estimators



# M-estimators

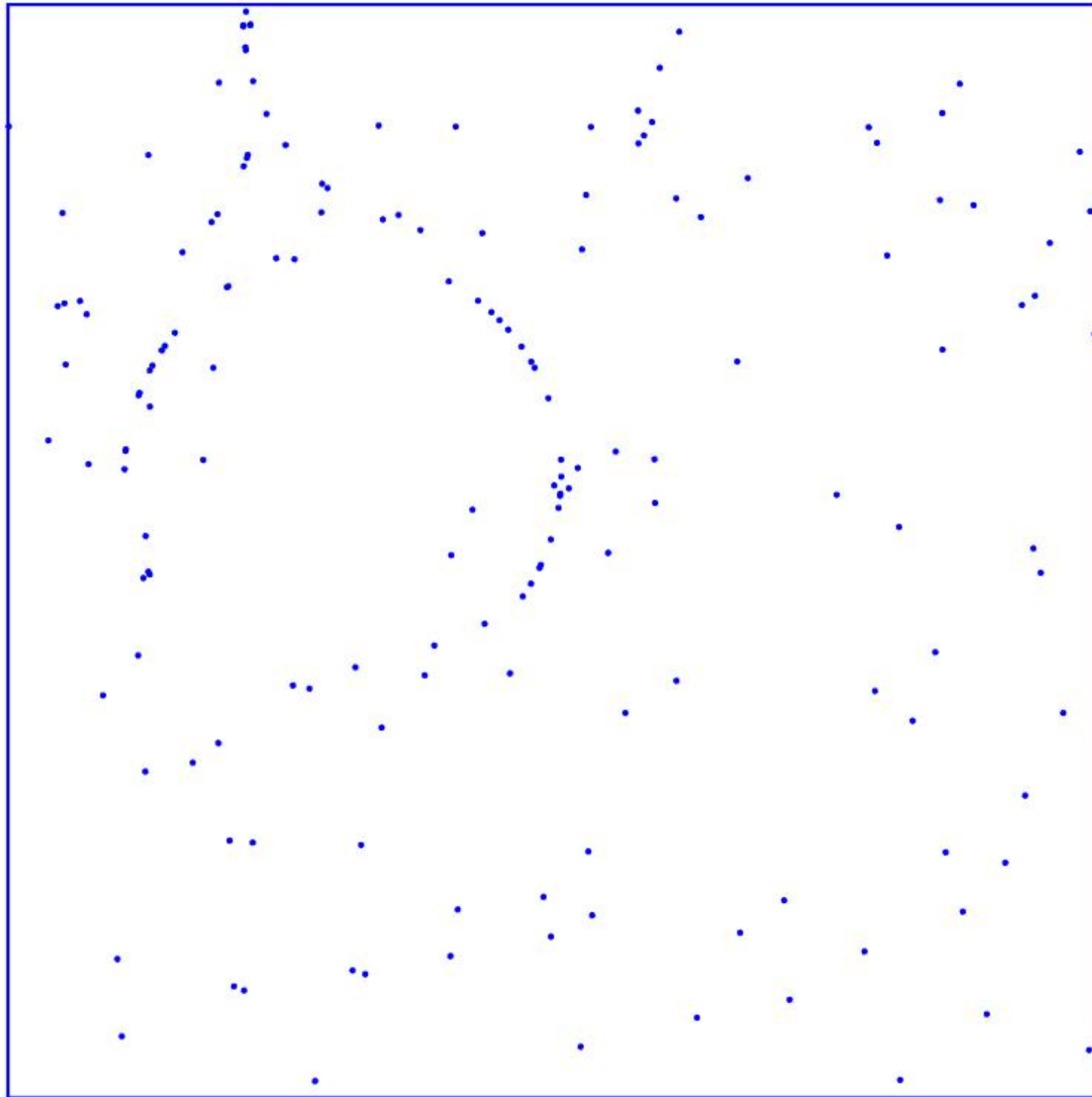
- ▶ How should  $\sigma$  be chosen?
- ▶ Read in the book. Study the examples.
- ▶ Problem with convergence to local optima.
- ▶ How do we obtain an initial guess?

# Outliers: RANSAC

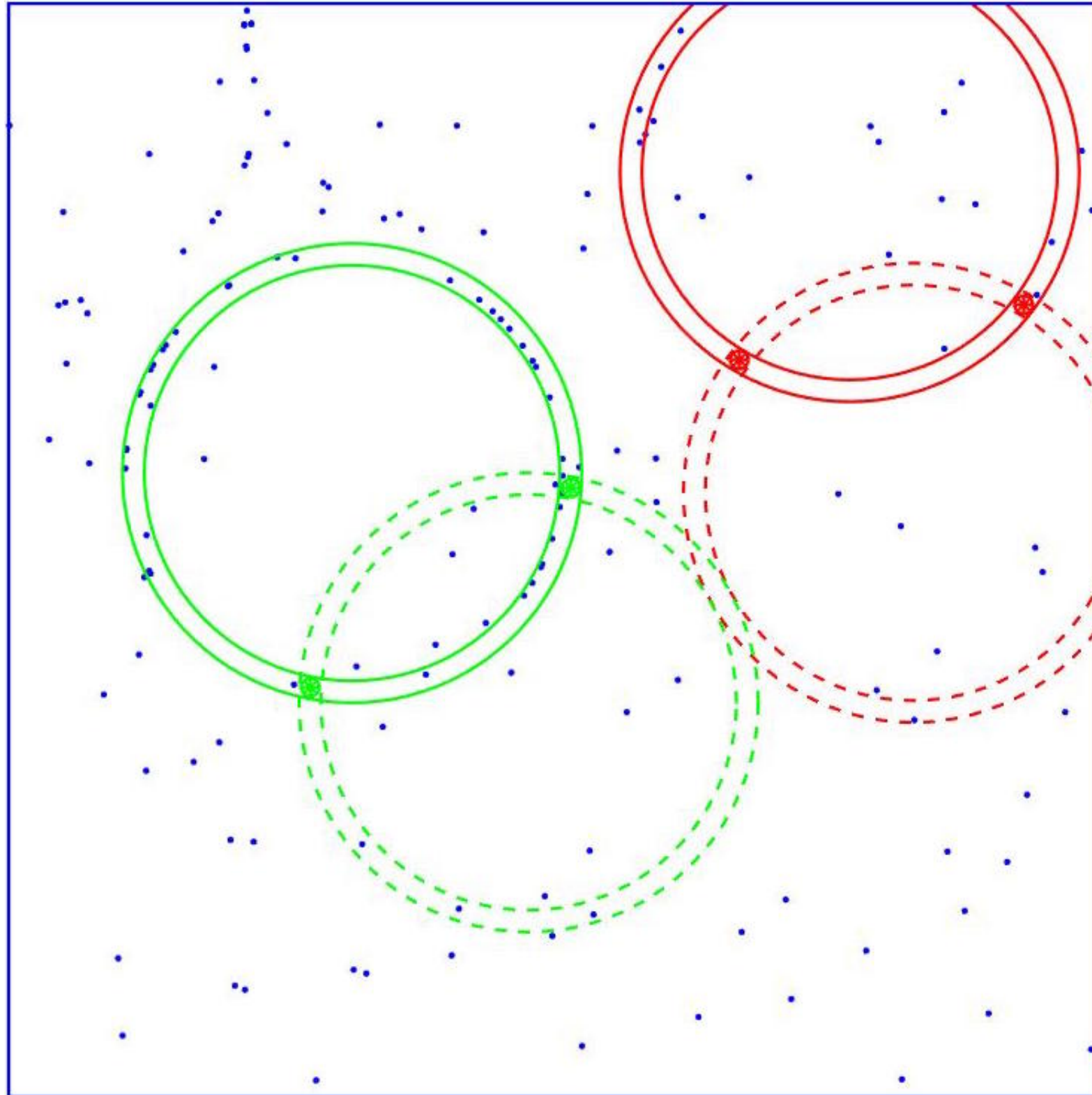
Another popular method to deal with outliers is RANSAC. It is an alternative to M-estimators (where we modified the underlying noise model to have heavier tails):

1. Randomly choose a minimal set of points needed for fitting.
2. Study how many points that now lie close to the line.
3. If there are sufficiently many, stop
4. Iterate 1-3 until stop, but at most  $k$  times.

# Example: Find circles with known radius

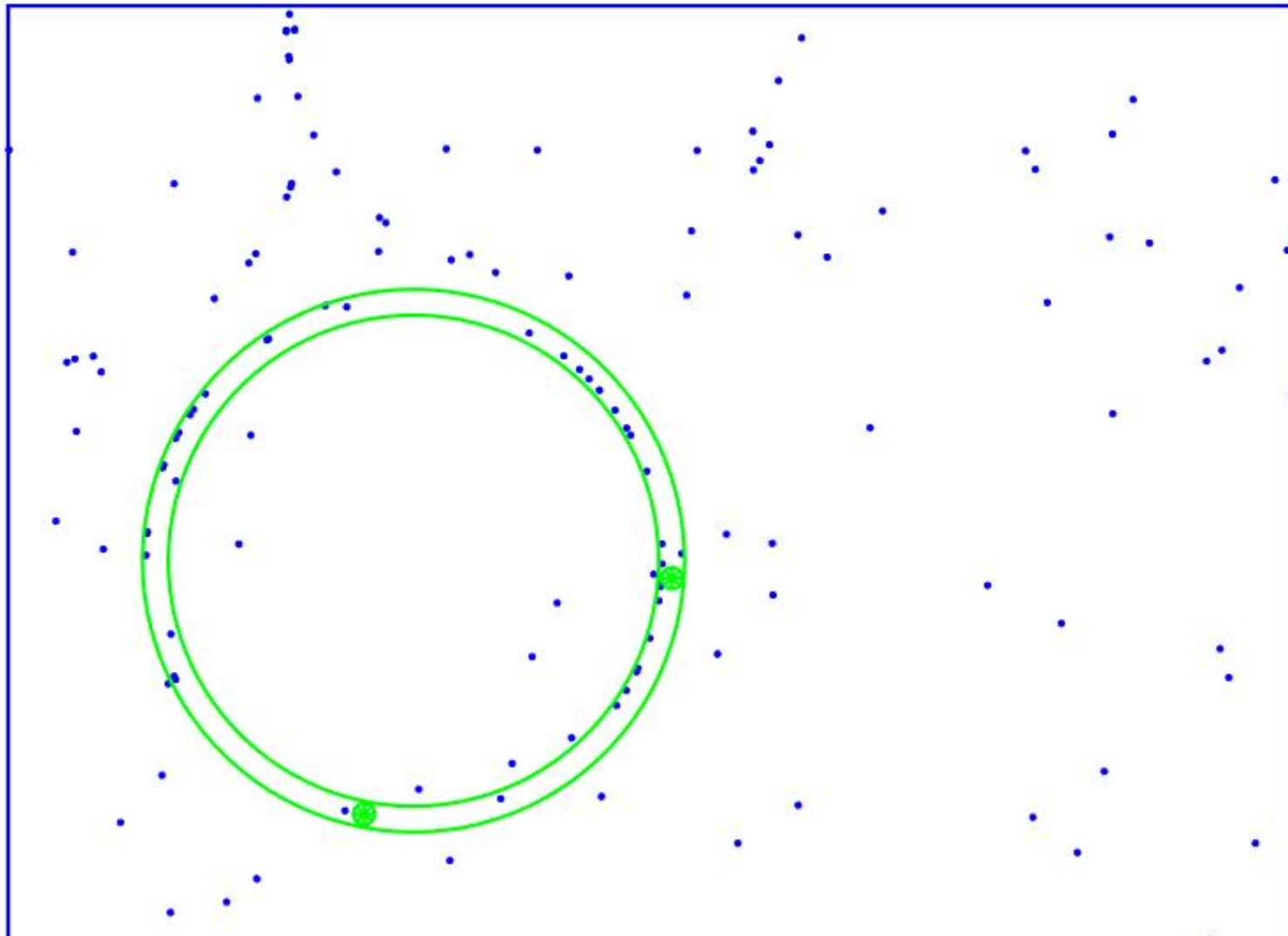


# Example: Find circles with known radius





# Example: Find circles with known radius

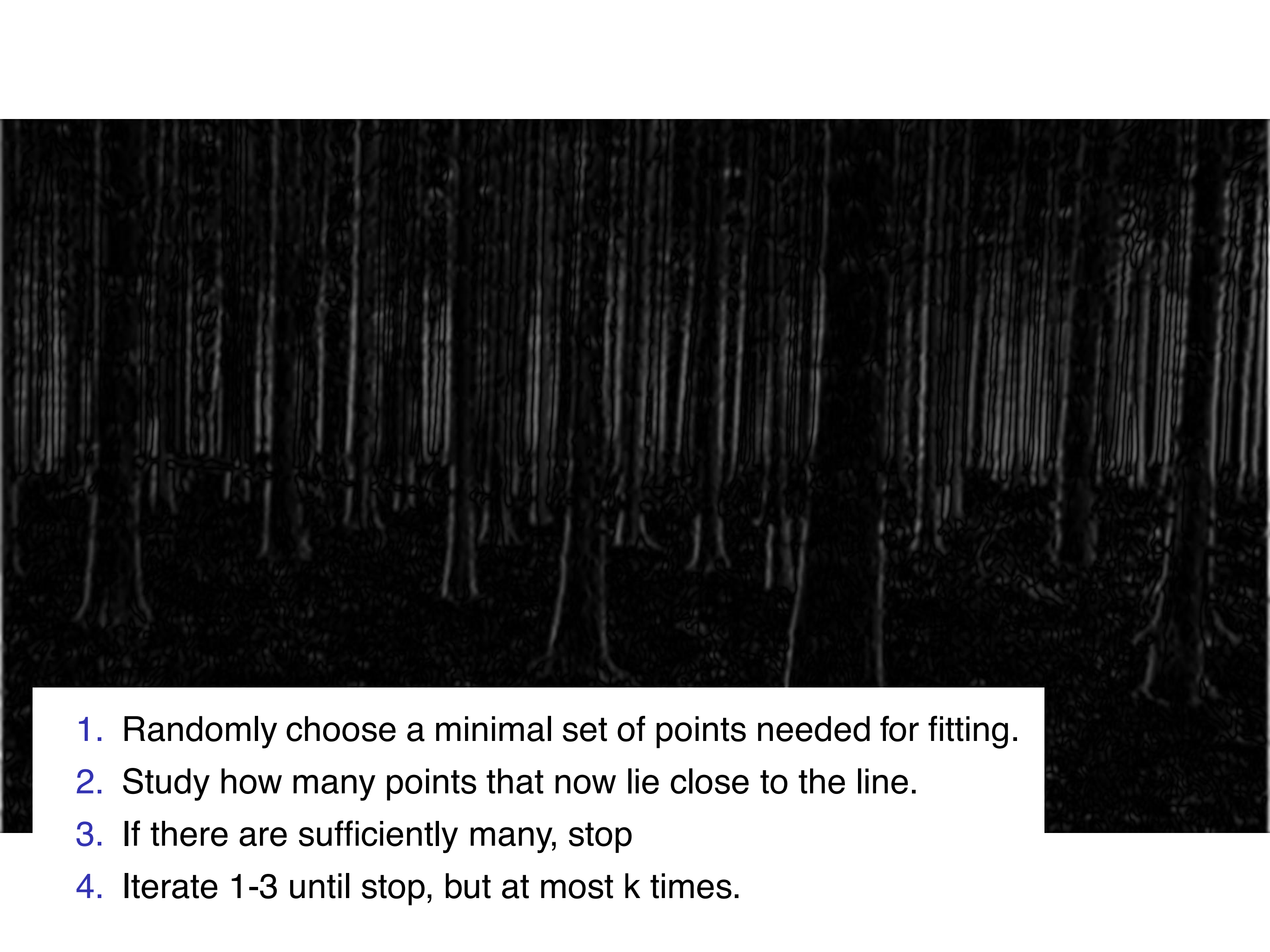


1. Randomly choose a minimal set of points needed for fitting.
2. Study how many points that now lie close to the line.
3. If there are sufficiently many, stop
4. Iterate 1-3 until stop, but at most  $k$  times.



1. Randomly choose a minimal set of points needed for fitting.
2. Study how many points that now lie close to the line.
3. If there are sufficiently many, stop
4. Iterate 1-3 until stop, but at most  $k$  times.



- 
1. Randomly choose a minimal set of points needed for fitting.
  2. Study how many points that now lie close to the line.
  3. If there are sufficiently many, stop
  4. Iterate 1-3 until stop, but at most  $k$  times.



1. Randomly choose a minimal set of points needed for fitting.
2. Study how many points that now lie close to the line.
3. If there are sufficiently many, stop
4. Iterate 1-3 until stop, but at most  $k$  times.

1800

# Overview – Parameter Estimation, fit

1. Voting, The Hough Transform
2. Fitting one line, least squares, total least squares, svd
3. Curve Fitting
4. Robust Fitting
  1. M-estimator
  2. RANSAC
5. **Fitting Multiple Objects**
  1. K-means
  2. Remove and fit again



# K-means and fitting

Assume that there are points from many lines.

Assume also that the number of lines is known.

Then one can use *k – means* for clustering points to lines.

Algorithm 15.2

1. Randomly choose  $k$  lines or a correspondence function  
 $c = \{1 \dots, n\} \rightarrow \{1, \dots, k\}$
2. Update  $c$ , i.e. assign points to the closest line.
3. Update  $l$ , i.e. fit lines to corresponding points.

# RANSAC for multiple objects

Another popular method to deal with outliers is RANSAC. It is an alternative to M-estimators (where we modified the underlying noise model to have heavier tails):

1. Randomly choose a minimal set of points needed for fitting.
2. Study how many points that now lie close to the line.
3. If there are sufficiently many, stop
4. Iterate 1-3 until stop, but at most  $k$  times.





LUND  
UNIVERSITY

