

# Principles of Robot Autonomy I

Information extraction



**Stanford**  
University



# Logistics

- Thank you for bearing with us on section logistics
  - Please email the course staff at [aa274a-aut2122-staff@lists.stanford.edu](mailto:aa274a-aut2122-staff@lists.stanford.edu) if you need to switch sections
- Final project group signup sheet is out; relevant also to group problems for HW3 and HW4
  - Concurrent SSH access limit per genbu instance has been raised to 2

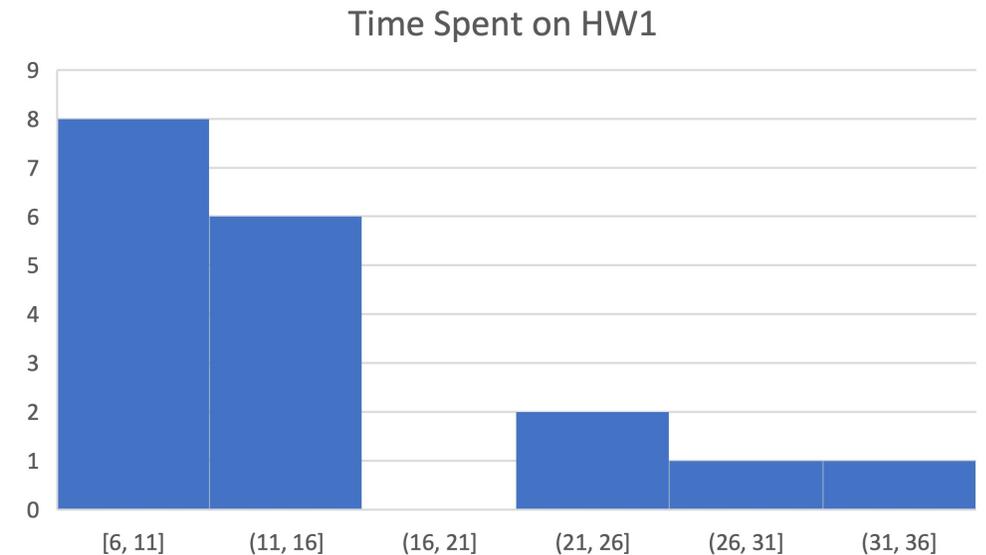
# Mid-Quarter Feedback – Statistics

Thus far, how would you rate this course overall?

<b>Poor</b>		<b>0%</b>	✓
Fair	1 respondent	4%	
Good	10 respondents	38%	
Very Good	6 respondents	23%	
Excellent	2 respondents	8%	

How would you describe the pace of Principles of Robot Autonomy I so far?

<b>Far too slow.</b>		<b>0%</b>	✓
Slightly too slow.	3 respondents	12%	
Just right.	8 respondents	31%	
Slightly too fast.	7 respondents	27%	
Far too fast.	1 respondent	4%	



# Mid-Quarter Feedback – Comments

- “I don't feel like I'm really learning ROS during sections but just following a series of instructions.”
  - “The fact that it takes us 2 hours to get two 20 line scripts to communicate is worrying.”
  - **Action Item:** We'll adjust future sections to have more of a big picture component
- “Pacing of lectures is uneven at best, with some parts being covered painfully slowly and others being blitzed over in an attempt to make up time.”
  - “The lecture needs pacing. Ed talks quickly in lecture and answers lots of questions, yet we are still falling behind.”
  - “The pace has felt rushed due to too many questions being taken by the instructor.”
  - “It would be better that if we have a fast pace through out the entire class instead of a slow pace at the beginning and a super fast pace near the end”
  - **Action Item:** I'll work on this, but I won't stop taking questions.
- “More comments in the starter code, even if it's in a section we don't need to modify! As someone new to python I try to understand how everything fits together”
  - **Action Item:** We will try to do this; also feel free to ask such clarification questions on EdStem.

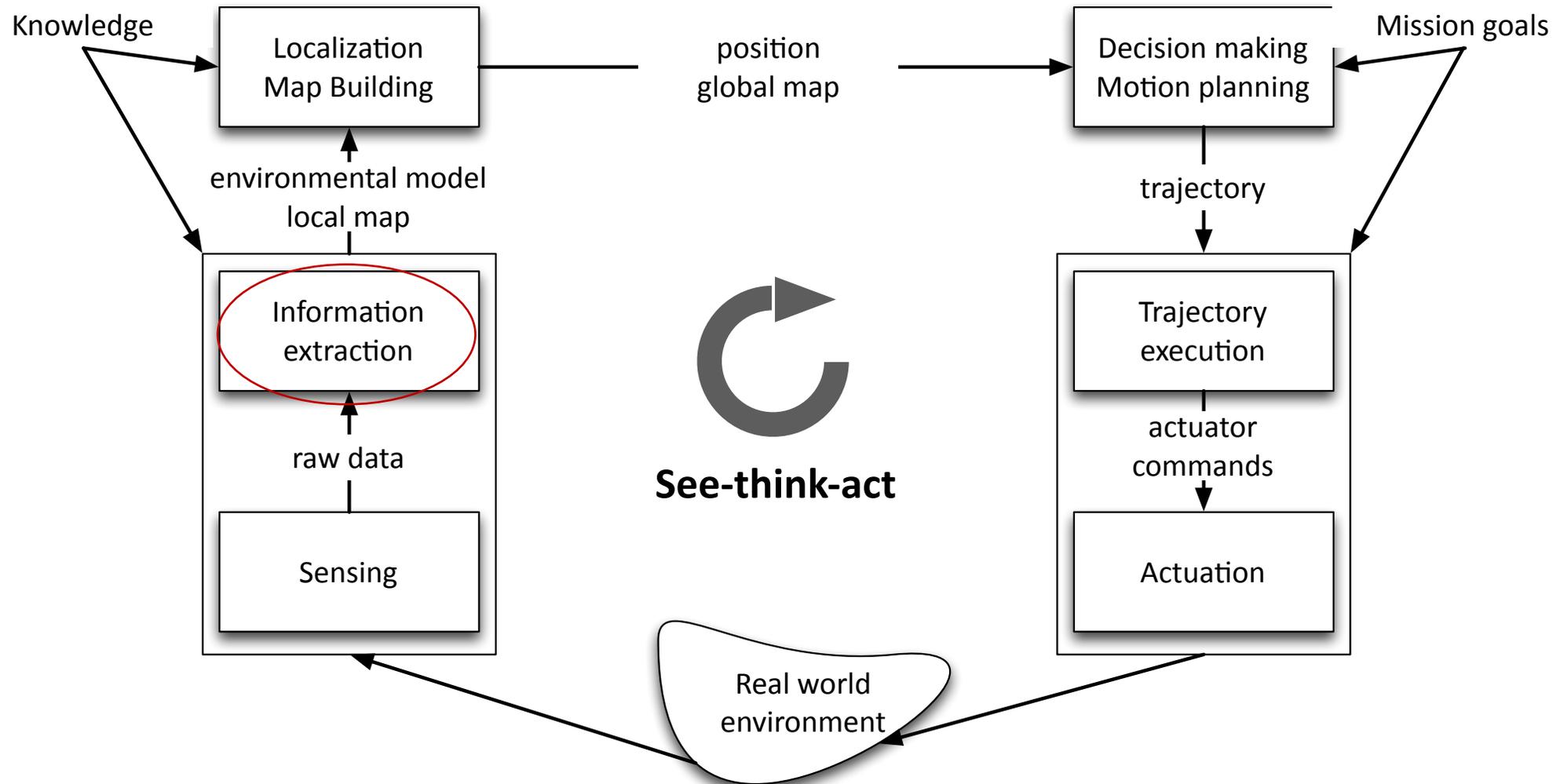
# Mid-Quarter Feedback – Comments

- “Check the mapping between homework and lecture, sometimes the approach or naming conventions are different. For example on HW1, the inclusion or exclusion of  $v$  as a state variable.”
  - “It would be better if HW assignments didn't depend on other things like outside packages or code from previous submissions because it introduces a lot of issues dealing with these other things that it distracts from time spent actually solving the problem for the current homework”
  - **Action Item:** Unfortunately this is life as a roboticist, but we'll try to note/smooth such cases
- “If [Ed] could have virtual office hours that would greatly expand his accessibility, particularly for remote students.”
  - **Action Item:** OH by appointment, but I'll also make a Zoom event for my Thursday OH. Open to doing another virtual hour specifically for remote students.
- Please follow up with us at [aa274a-aut2122-staff@lists.stanford.edu](mailto:aa274a-aut2122-staff@lists.stanford.edu) if your comment wasn't addressed!

# Techniques for information extraction

- Aim
  - Learn how to extract information from sensor measurements
- Readings
  - Siegwart, Nourbakhsh, Scaramuzza. Introduction to Autonomous Mobile Robots. Sections: 4.1.3, 4.6.1 - 4.6.5, 4.7.1 - 4.7.4

# The see-think-act cycle



# Information extraction

- Next step is to extract *information* from images, such as
  - Geometric primitives (e.g., lines and circles): useful, for example, for robot localization and mapping
  - Object recognition and scene understanding: useful, for example, for localization within a topological map and for high-level reasoning

# Geometric feature extraction

- **Geometric feature extraction**: extract geometric primitives from sensor data (e.g., range data)
- Examples: line, circles, corners, planes, etc.
- We focus on *line extraction* from range data (a quite common task); other geometric feature extraction tasks are conceptually analogous
- The two main problems of line extraction from range data
  1. Which points belong to which line? → *segmentation*
  2. Given an association of points to a line, how to estimate line parameters? → *fitting*

# Step #2: line fitting

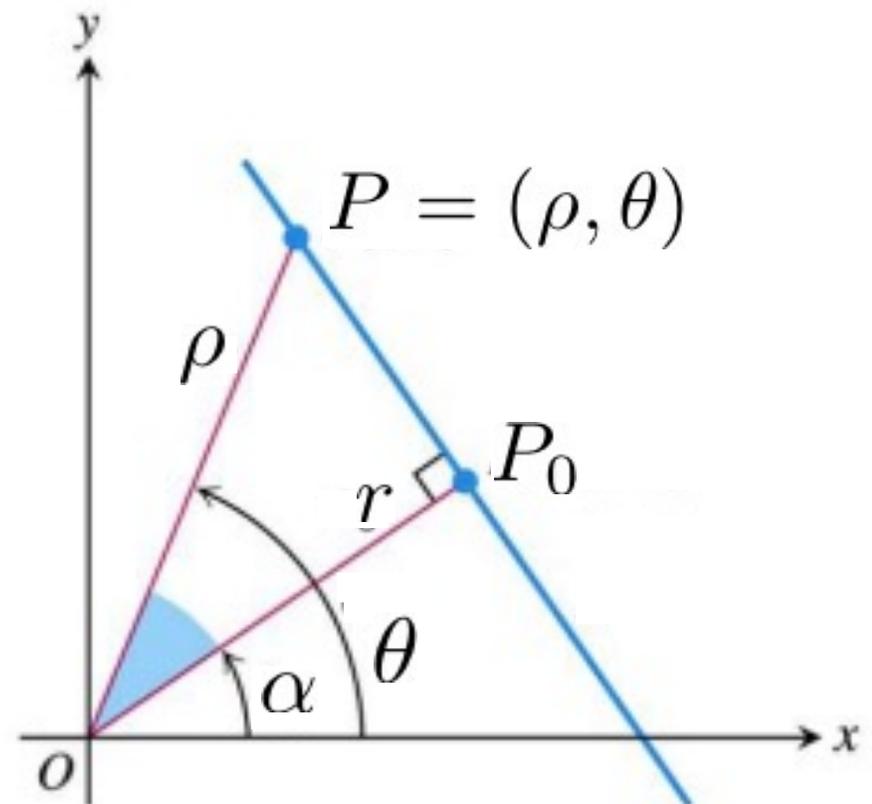
- **Goal:** fit a line to a set of sensor measurements
- It is useful to work in polar coordinates:

$$x = \rho \cos \theta, \quad y = \rho \sin \theta$$

- Equation of a line in polar coordinates
  - Let  $P = (\rho, \theta)$  be an arbitrary point on the line
  - Since  $P, P_0, O$  determine a right triangle

$$\rho \cos(\theta - \alpha) = r \quad \text{or} \quad x \cos \alpha + y \sin \alpha = r$$

- $(r, \alpha)$  are the parameters of the line



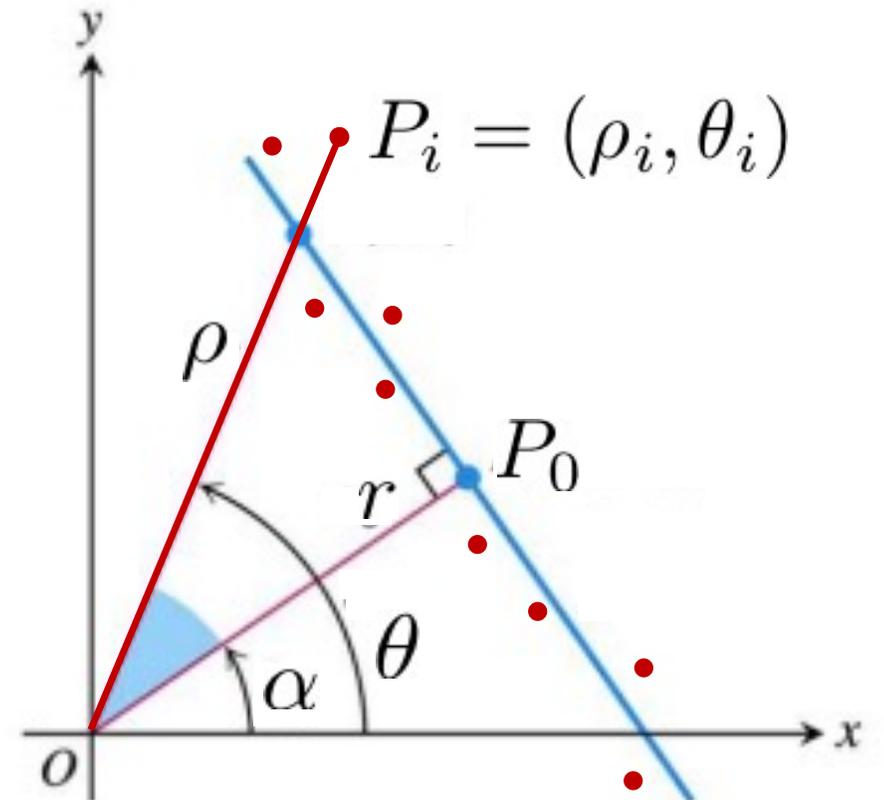
# Step #2: line fitting

- Since there is measurement error, the equation of the line is only *approximately* satisfied

$$\rho_i \cos(\theta_i - \alpha) = r + d_i$$

Error

- Assume  $n$  ranging measurement points represented in polar coordinates as  $(\rho_i, \theta_i)$
- We want to find a line that best “fits” all the measurement points



## Step #2: line fitting

- Consider, first, that all measurements are equally uncertain
- Find line parameters  $(r, \alpha)$  that minimize squared error

$$S(r, \alpha) := \sum_{i=1}^n d_i^2 = \sum_{i=1}^n (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Unweighted least squares

## Step #2: line fitting

- Consider, now, the case where each measurement has its own, unique uncertainty
- For example, assume that the variance for each range measurement  $\rho_i$  is  $\sigma_i$
- Associate with each measurement a weight, e.g.,  $w_i = 1/\sigma_i^2$
- Then, one minimizes

$$S(r, \alpha) := \sum_{i=1}^n w_i d_i^2 = \sum_{i=1}^n w_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Weighted least squares

## Step #2: line fitting solution

- Assume that the  $n$  ranging measurements are **independent**
- Solution:

$$\alpha = \frac{1}{2} \operatorname{atan2} \left( \frac{\sum_i w_i \rho_i^2 \sin 2\theta_i - \frac{2}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos \theta_i \sin \theta_j}{\sum_i w_i \rho_i^2 \cos 2\theta_i - \frac{1}{\sum_i w_i} \sum_i \sum_j w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)} \right) + \frac{\pi}{2}$$

$$r = \frac{\sum_i w_i \rho_i \cos(\theta_i - \alpha)}{\sum_i w_i}$$

# Step #1: line segmentation

- Several algorithms are available
- We will consider three popular algorithms
  1. Split-and-merge
  2. RANSAC
  3. Hough-Transform

# Split-and-merge algorithm

- Most popular line extraction algorithm

---

**Data:** Set  $S$  consisting of all  $N$  points, a distance threshold  $d > 0$

**Result:**  $L$ , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

**while**  $i \leq \text{len}(L)$  **do**

    fit a line  $(r, \alpha)$  to the set  $L_i$ ;

    detect the point  $P \in L_i$  with the maximum distance  $D$  to the line  $(r, \alpha)$ ;

**if**  $D < d$  **then**

        |  $i \leftarrow i + 1$

**else**

        | split  $L_i$  at  $P$  into  $S_1$  and  $S_2$ ;

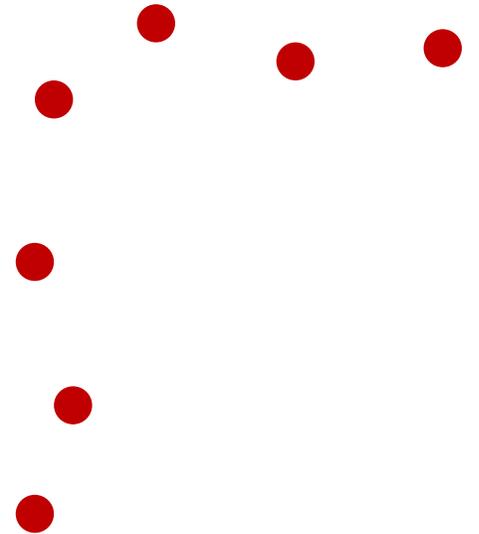
        |  $L_i \leftarrow S_1; L_{i+1} \leftarrow S_2$ ;

**end**

**end**

Merge collinear sets in  $L$ ;

---



# Split-and-merge algorithm

- Most popular line extraction algorithm

---

**Data:** Set  $S$  consisting of all  $N$  points, a distance threshold  $d > 0$

**Result:**  $L$ , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

**while**  $i \leq \text{len}(L)$  **do**

    fit a line  $(r, \alpha)$  to the set  $L_i$ ;

    detect the point  $P \in L_i$  with the maximum distance  $D$  to the line  $(r, \alpha)$ ;

**if**  $D < d$  **then**

        |  $i \leftarrow i + 1$

**else**

        | split  $L_i$  at  $P$  into  $S_1$  and  $S_2$ ;

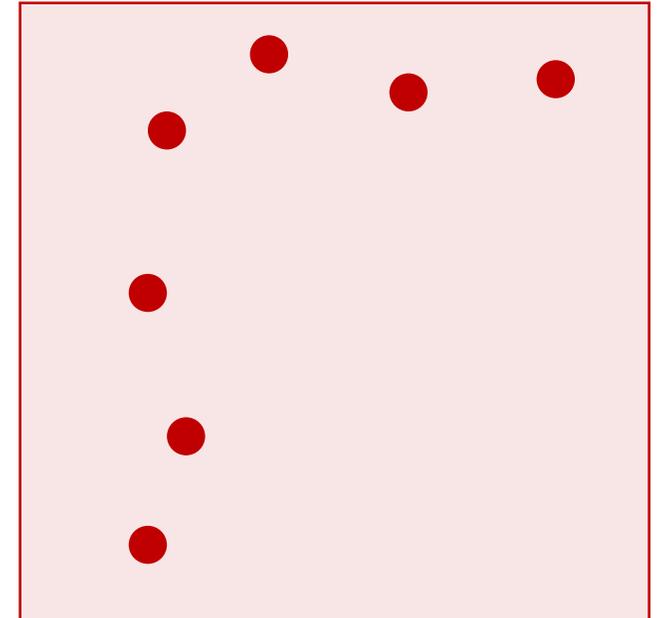
        |  $L_i \leftarrow S_1; L_{i+1} \leftarrow S_2$ ;

**end**

**end**

Merge collinear sets in  $L$ ;

---



# Split-and-merge algorithm

- Most popular line extraction algorithm

---

**Data:** Set  $S$  consisting of all  $N$  points, a distance threshold  $d > 0$

**Result:**  $L$ , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

**while**  $i \leq \text{len}(L)$  **do**

    fit a line  $(r, \alpha)$  to the set  $L_i$ ;

    detect the point  $P \in L_i$  with the maximum distance  $D$  to the line  $(r, \alpha)$ ;

**if**  $D < d$  **then**

        |  $i \leftarrow i + 1$

**else**

        | split  $L_i$  at  $P$  into  $S_1$  and  $S_2$ ;

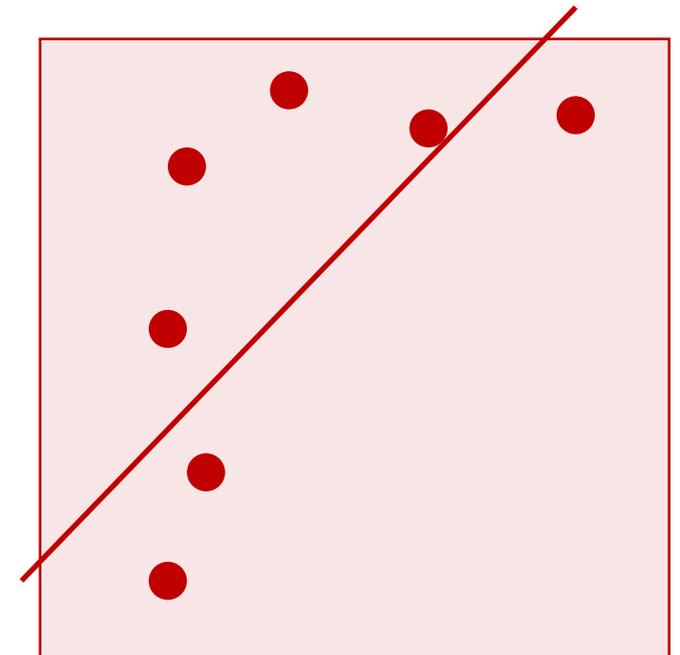
        |  $L_i \leftarrow S_1; L_{i+1} \leftarrow S_2$ ;

**end**

**end**

Merge collinear sets in  $L$ ;

---



# Split-and-merge algorithm

- Most popular line extraction algorithm

---

**Data:** Set  $S$  consisting of all  $N$  points, a distance threshold  $d > 0$

**Result:**  $L$ , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

**while**  $i \leq \text{len}(L)$  **do**

    fit a line  $(r, \alpha)$  to the set  $L_i$ ;

    detect the point  $P \in L_i$  with the maximum distance  $D$  to the line  $(r, \alpha)$ ;

**if**  $D < d$  **then**

        |  $i \leftarrow i + 1$

**else**

        | split  $L_i$  at  $P$  into  $S_1$  and  $S_2$ ;

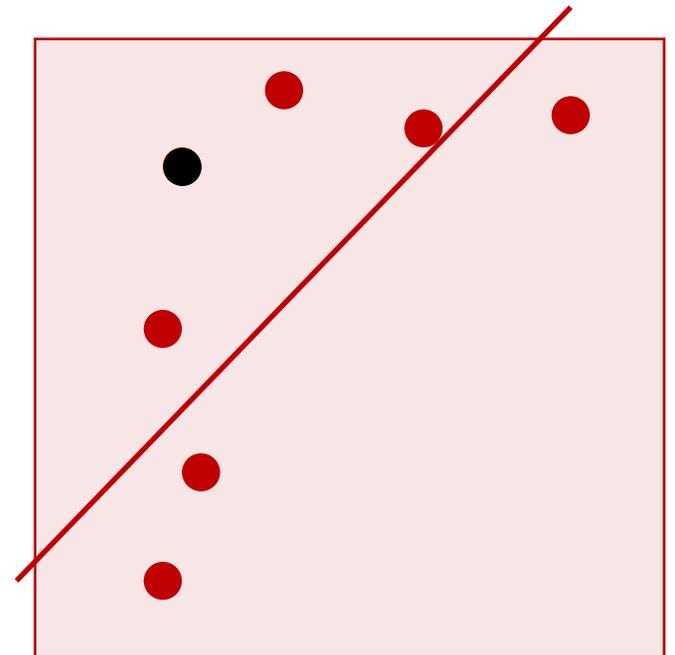
        |  $L_i \leftarrow S_1; L_{i+1} \leftarrow S_2$ ;

**end**

**end**

Merge collinear sets in  $L$ ;

---



# Split-and-merge algorithm

- Most popular line extraction algorithm

---

**Data:** Set  $S$  consisting of all  $N$  points, a distance threshold  $d > 0$

**Result:**  $L$ , a list of sets of points each resembling a line

$L \leftarrow (S), i \leftarrow 1;$

**while**  $i \leq \text{len}(L)$  **do**

    fit a line  $(r, \alpha)$  to the set  $L_i$ ;

    detect the point  $P \in L_i$  with the maximum distance  $D$  to the line  $(r, \alpha)$ ;

**if**  $D < d$  **then**

        |  $i \leftarrow i + 1$

**else**

        | split  $L_i$  at  $P$  into  $S_1$  and  $S_2$ ;

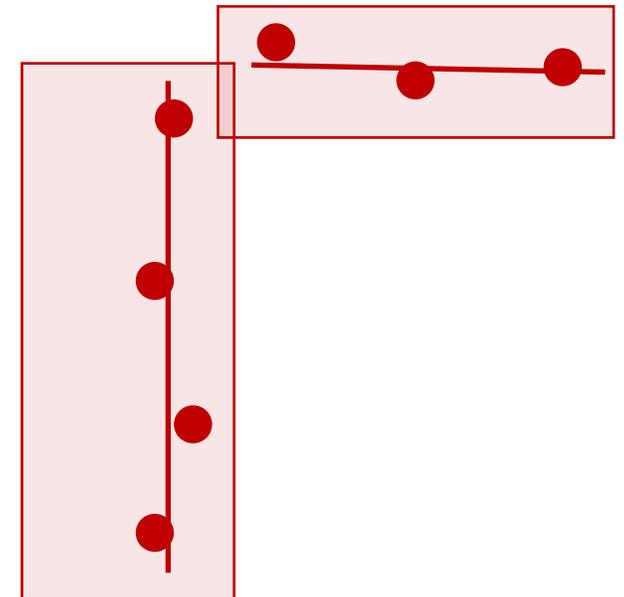
        |  $L_i \leftarrow S_1; L_{i+1} \leftarrow S_2$ ;

**end**

**end**

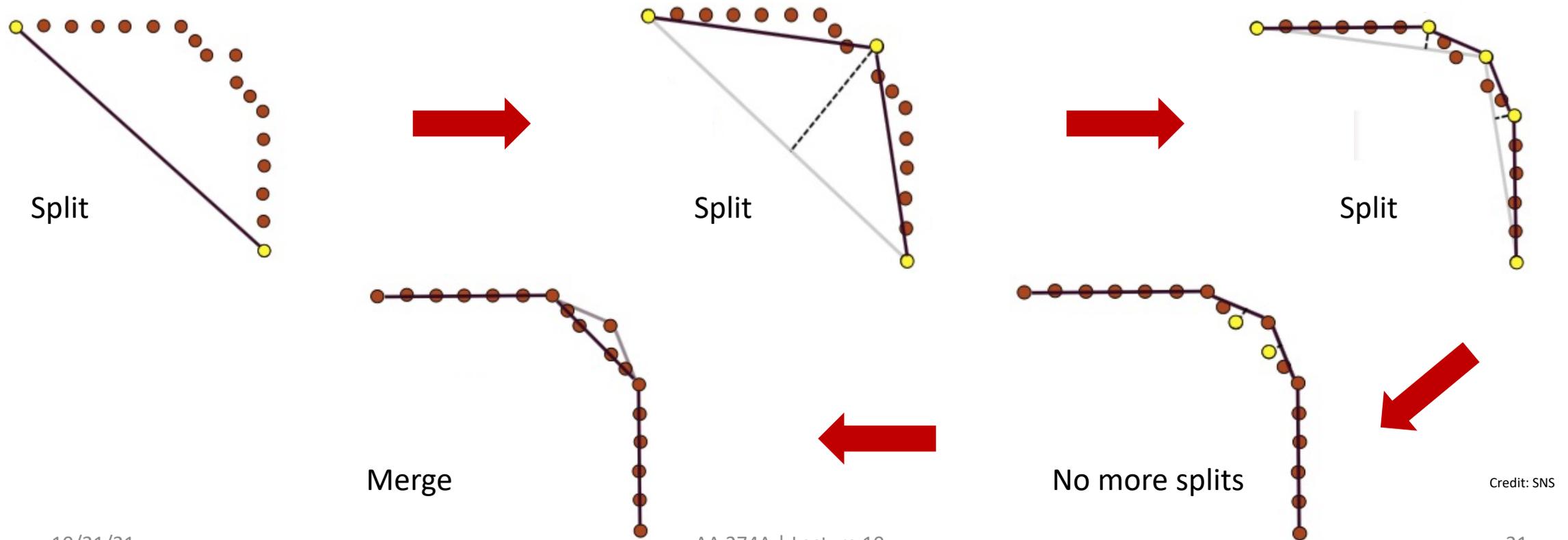
Merge collinear sets in  $L$ ;

---



# Split-and-merge: iterative-end-point-fit variant

- Iterative-end-point-fit: split-and-merge where the line is constructed by simply connecting the first and last points (as opposed to least squares fit)



# RANSAC

- RANSAC: **R**andom **S**ample **C**onsensus
- General method to estimate parameters of a model from a set of observed data in the presence of outliers, where outliers should have no influence on the estimates of the values
- Typical applications in robotics: line extraction from 2D range data, plane extraction from 3D point clouds, feature matching for structure from motion, etc.
- RANSAC is *iterative* and *non-deterministic*: the probability of finding a set free of outliers increases as more iterations are used

# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

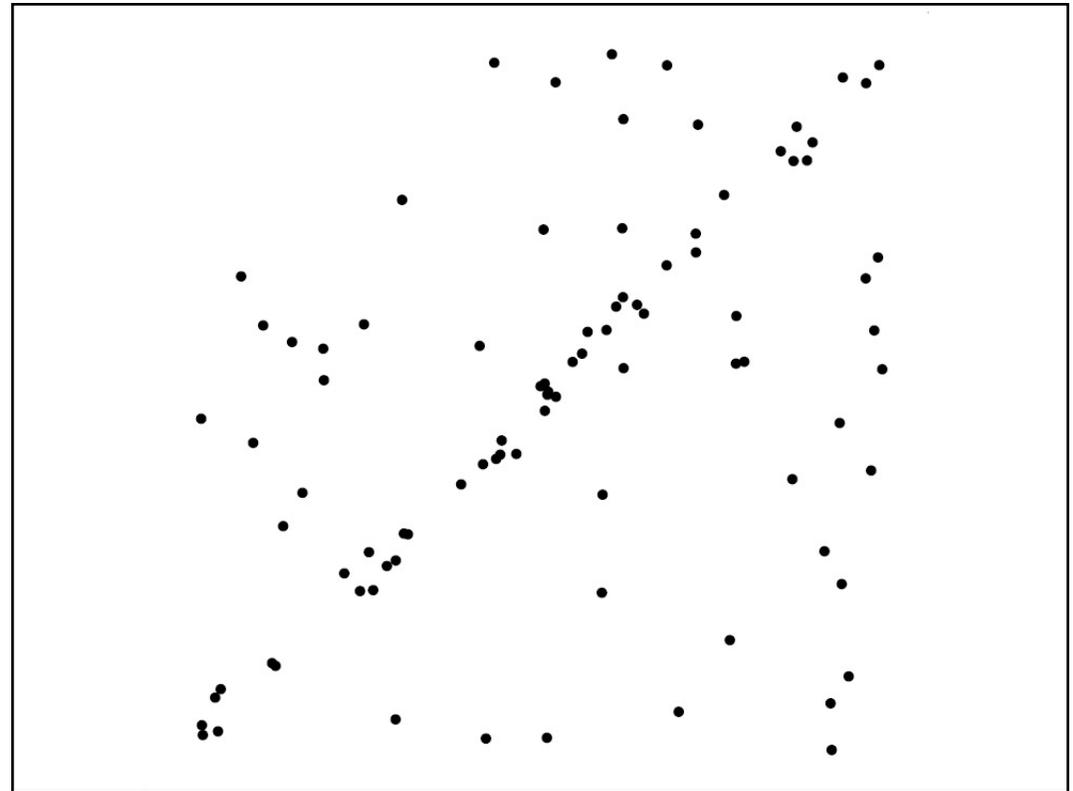
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

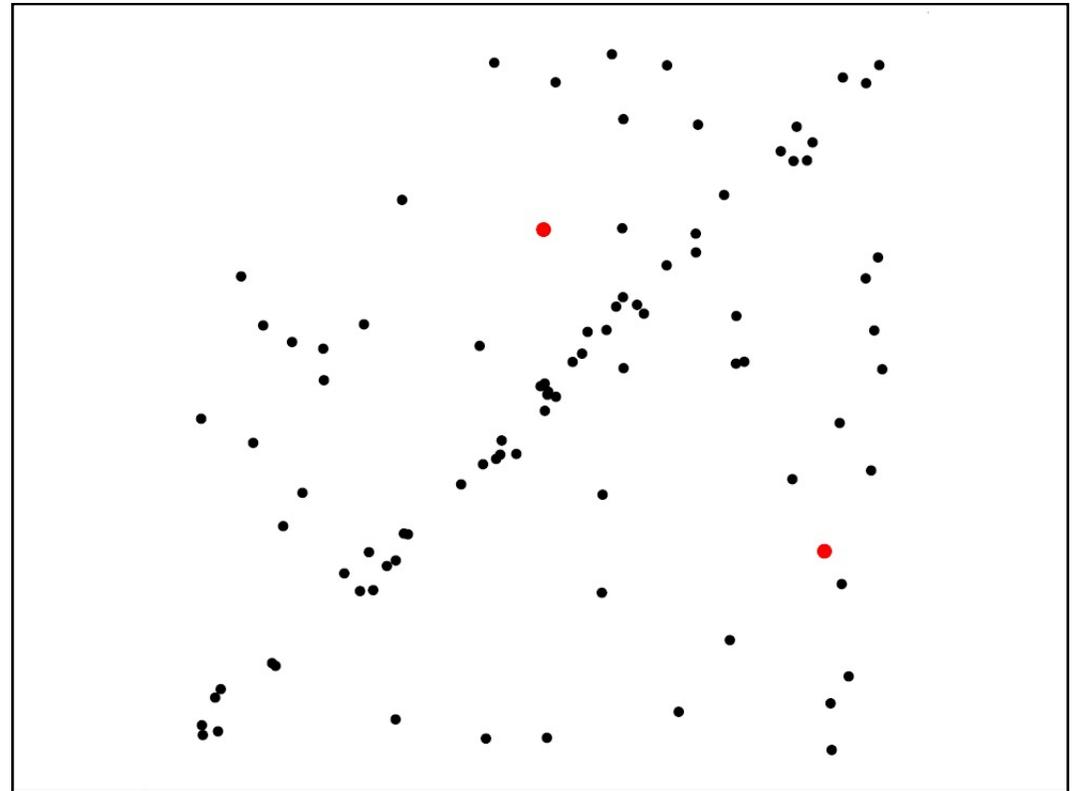
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

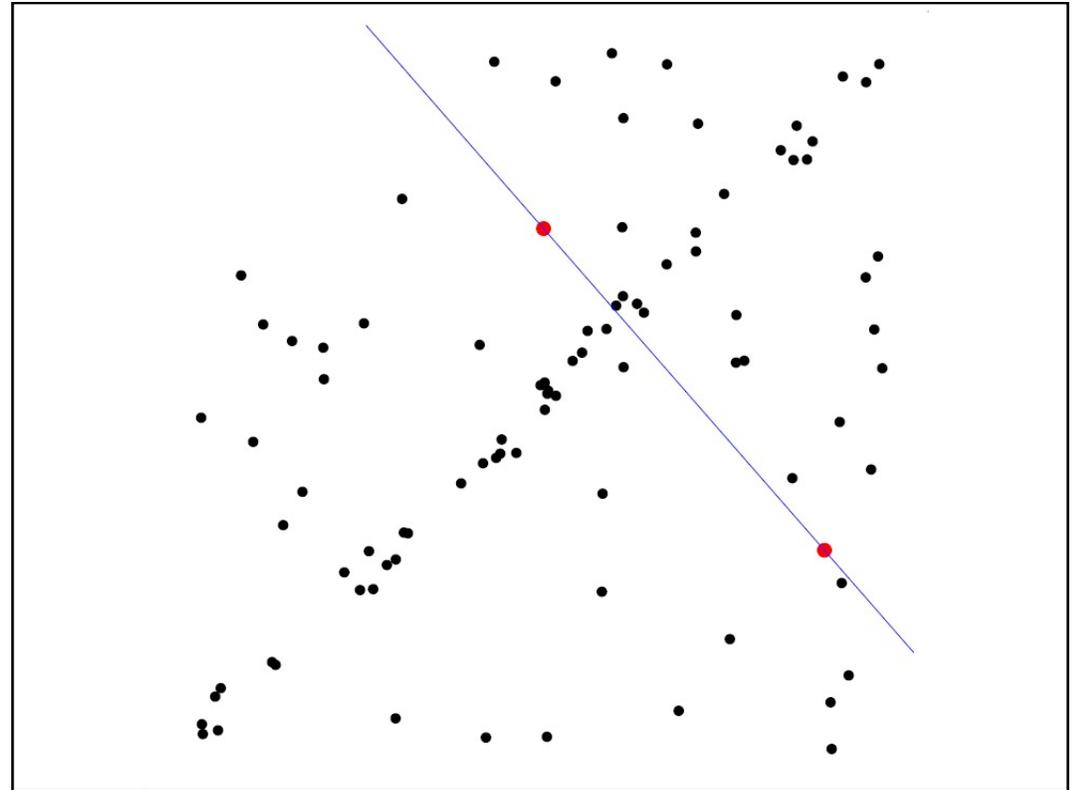
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

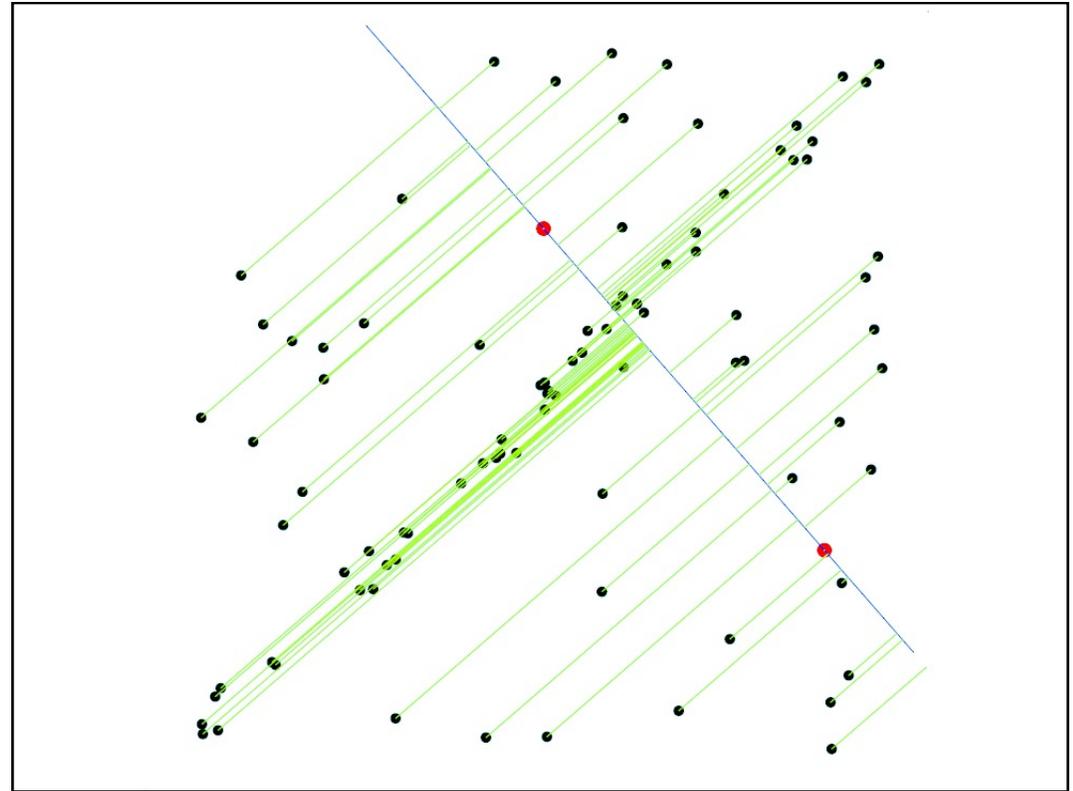
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

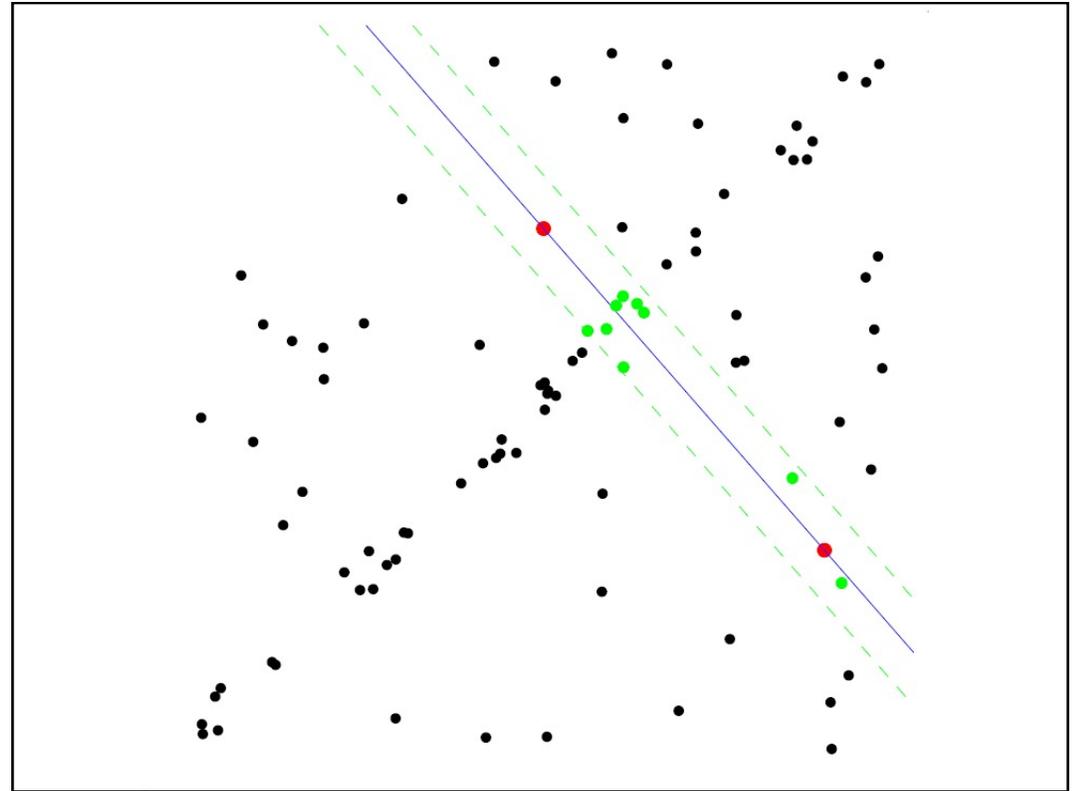
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

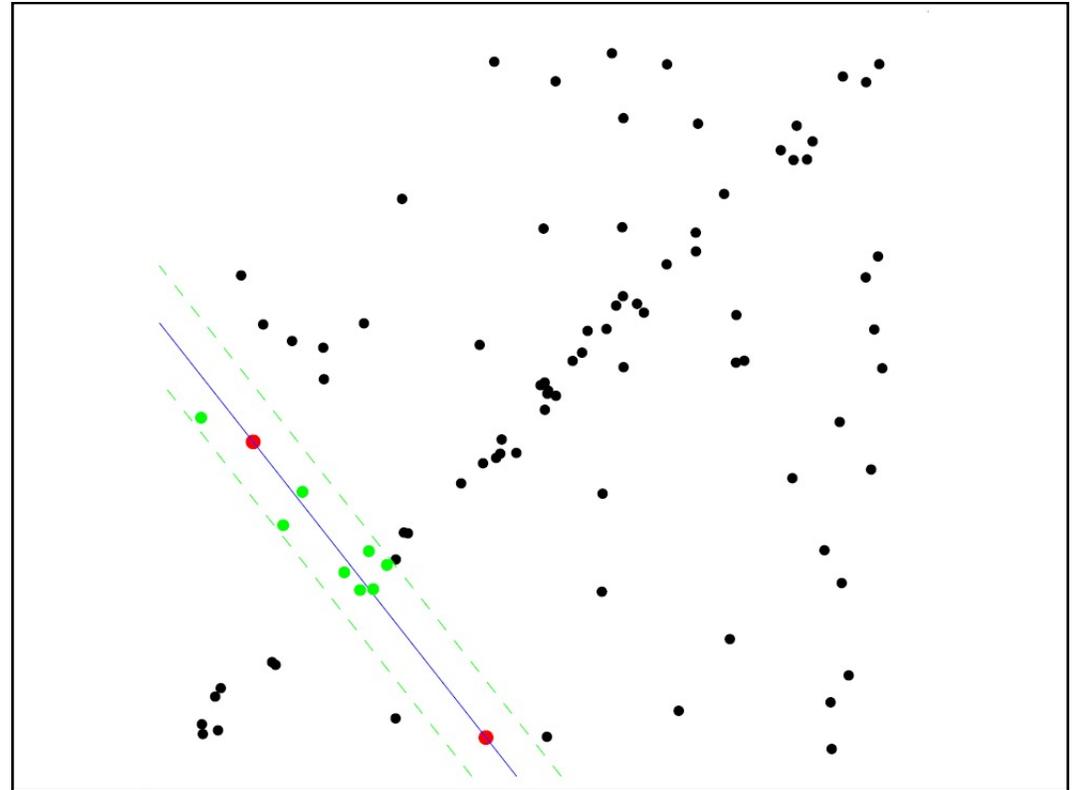
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

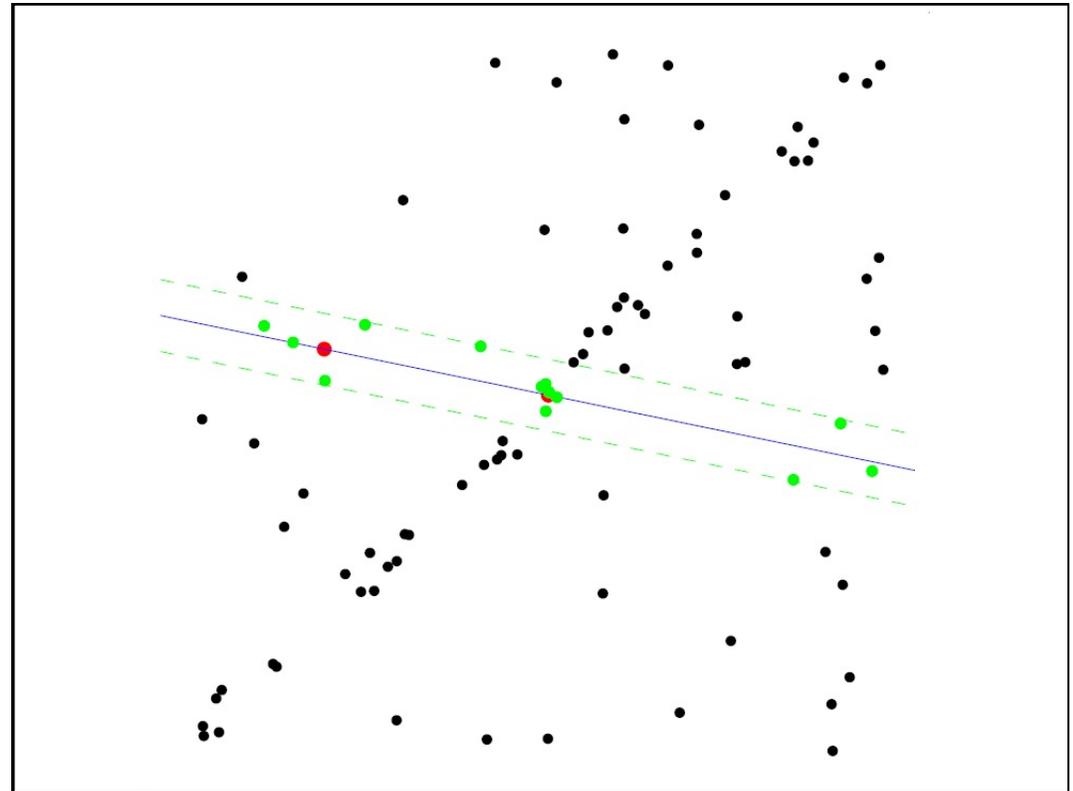
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC

**Data:** Set  $S$  consisting of all  $N$  points

**Result:** Set with maximum number of inliers  
(and corresponding fitting line)

**while**  $i \leq k$  **do**

    randomly select 2 points from  $S$ ;

    fit line  $l_i$  through the 2 points;

    compute distance of all other points to line  $l_i$  ;

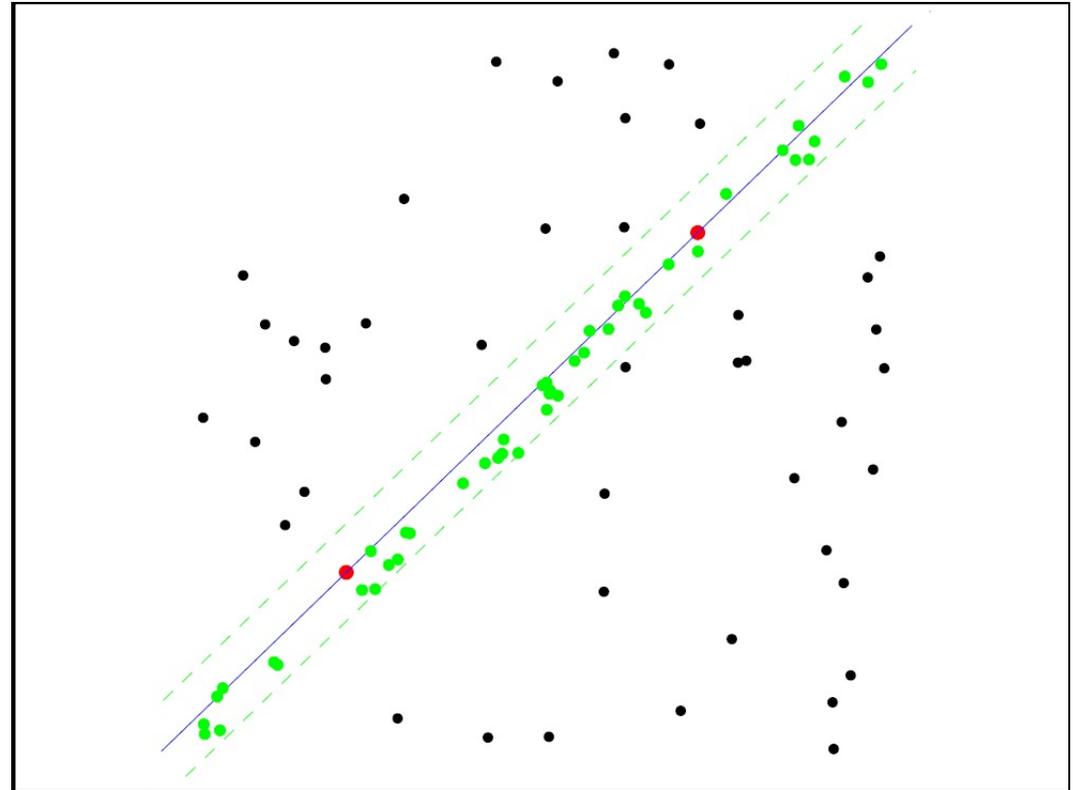
    construct *inlier* set, i.e., count number of  
    points with distance to the line less than  $\gamma$ ;

    store line  $l_i$  and associated set of inliers;

$i \leftarrow i + 1$

**end**

Choose set with maximum number of inliers



# RANSAC iterations

- In principle, one would need to check all possible combinations of 2 points in dataset
- If  $|S| = N$ , number of combinations is  $\frac{N(N-1)}{2} \rightarrow$  too many
- However, if we have a rough estimate of the percentage of inliers, we do not need to check all combinations...

# RANSAC iterations: statistical characterization

- Let  $w$  be the percentage of inliers in the dataset, i.e.,

$$w = \frac{\text{number of inliers}}{N}$$

- Let  $p$  be the desired probability of finding a set of points free of outliers (typically,  $p = 0.99$ )
- Assumption: 2 points chosen for line estimation are selected independently
  - $P(\text{both points selected are inliers}) = w^2$
  - $P(\text{at least one of the selected points is an outlier}) = 1 - w^2$
  - $P(\text{RANSAC never selects two points that are both inliers}) = (1 - w^2)^k$

# RANSAC iterations: statistical characterization

- Then minimum number of iterations  $\bar{k}$  to find an outlier-free set with probability at least  $p$  is:

$$1 - p = (1 - w^2)^{\bar{k}} \Rightarrow \bar{k} = \frac{\log(1 - p)}{\log(1 - w^2)}$$

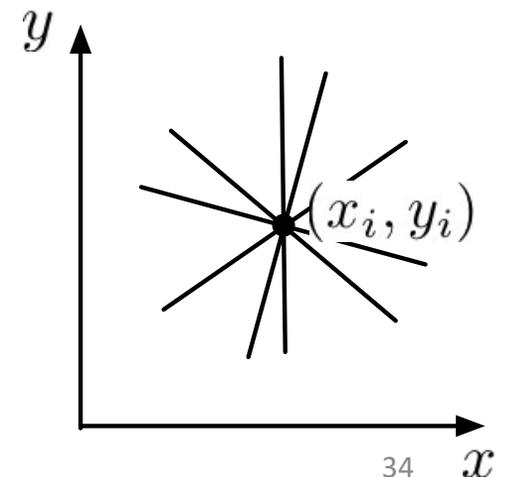
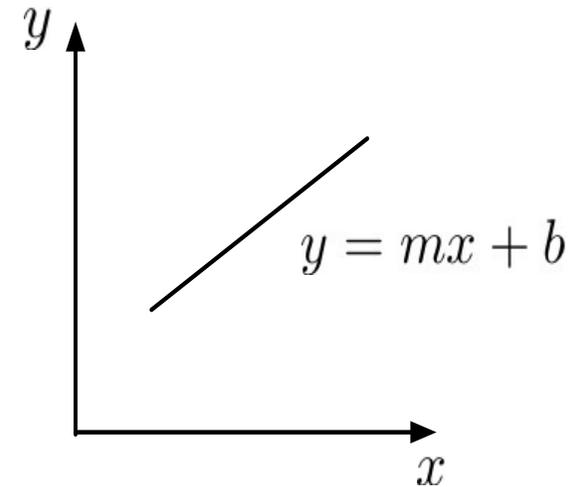
- Thus if we know  $w$  (at least approximately), after  $\bar{k}$  iterations RANSAC will find a set free of outliers with probability  $p$
- Note:
  - $\bar{k}$  depends only on  $w$ , not on  $N$ !
  - More advanced versions of RANSAC estimate  $w$  adaptively

# Hough transform

- **Key idea:** each point votes for a set of plausible line parameters

- A line has two parameters:  $(m, b)$
- Given a point  $(x_i, y_i)$ , the lines that could pass through this point are all  $(m, b)$  satisfying

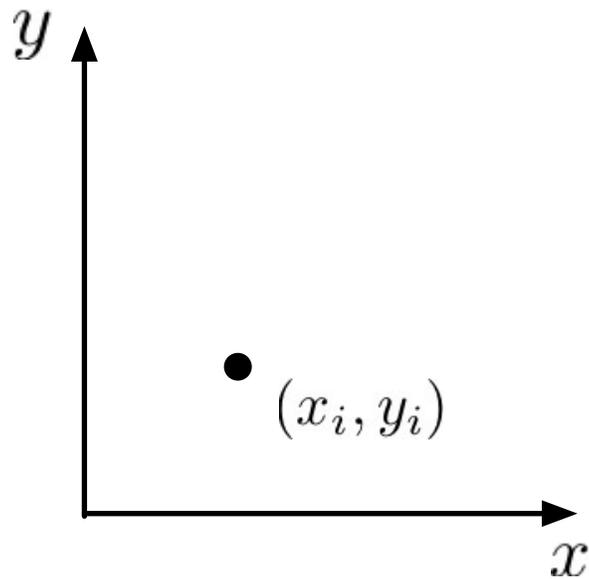
$$y_i = mx_i + b, \quad \text{or } b = -mx_i + y_i$$



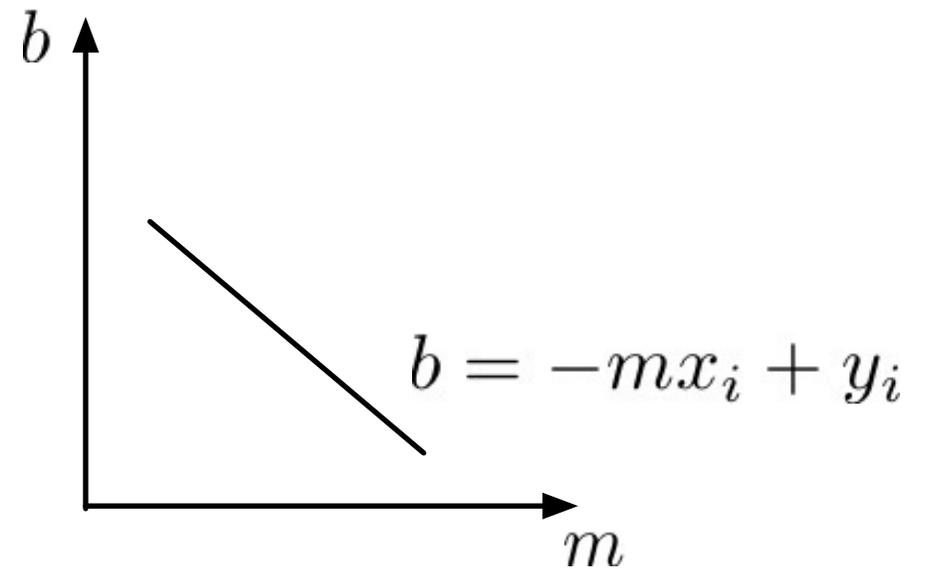
# Hough transform

- A point in image space maps into a line in *Hough space*

Image space

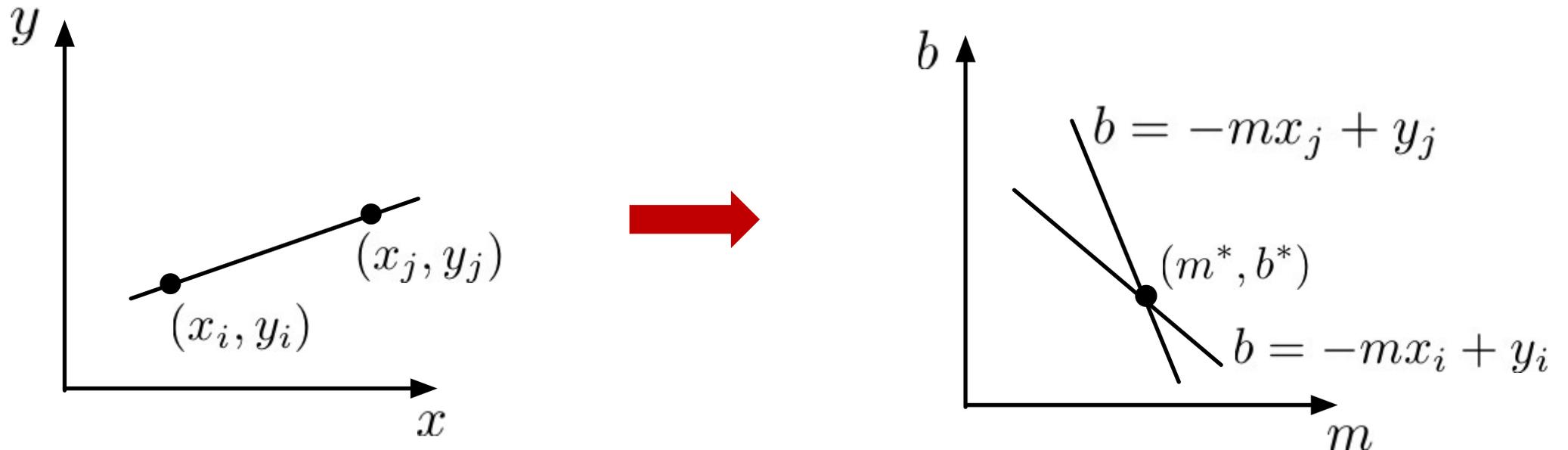


Hough parameter space



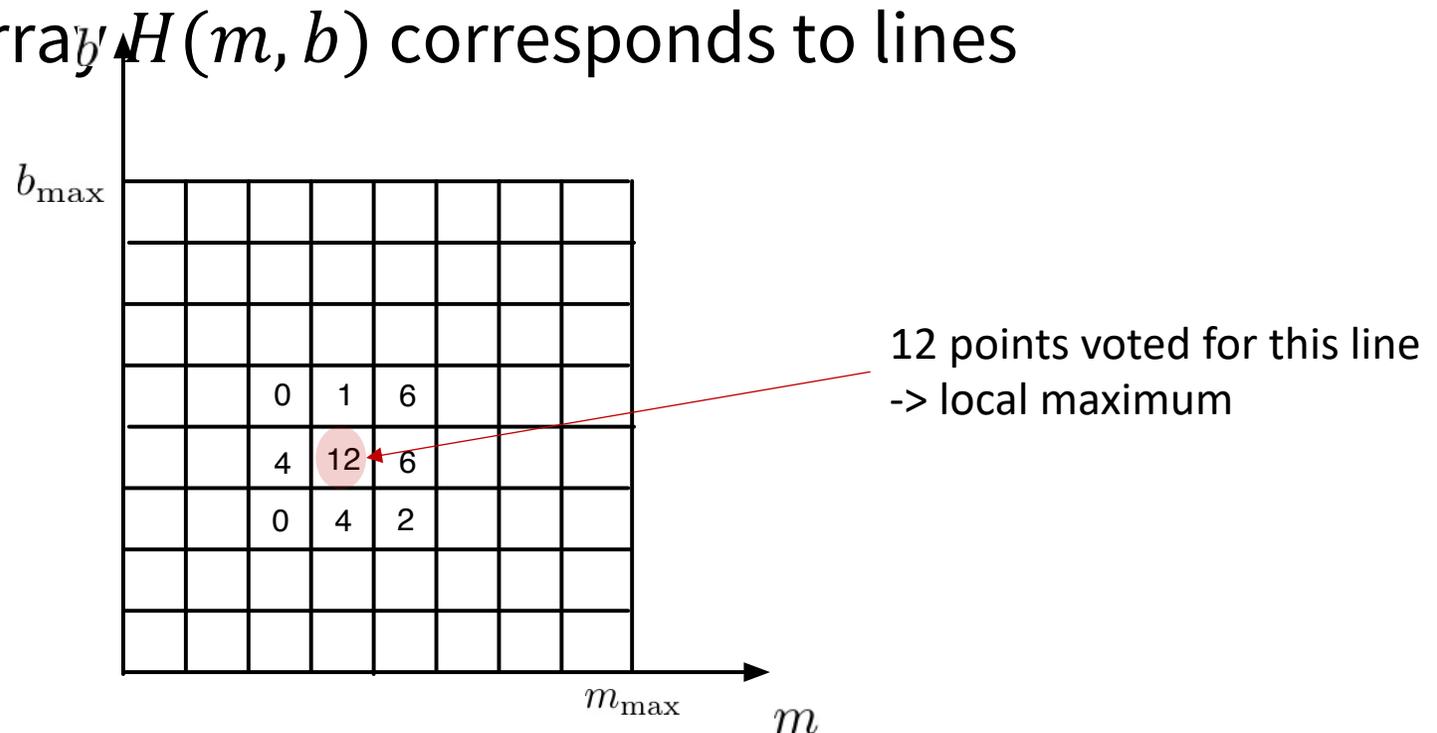
# Hough transform

- **Key fact:** all points on a line in image space yield lines in parameter space which intersect at a *common point*,  $(m^*, b^*)$



# Hough transform algorithm

1. Initialize an accumulator array  $H(m, b)$  to zero
2. For each point  $(x_i, y_i)$ , increment all cells that satisfy  $b = -x_i m + y_i$
3. Local maxima in array  $H(m, b)$  corresponds to lines

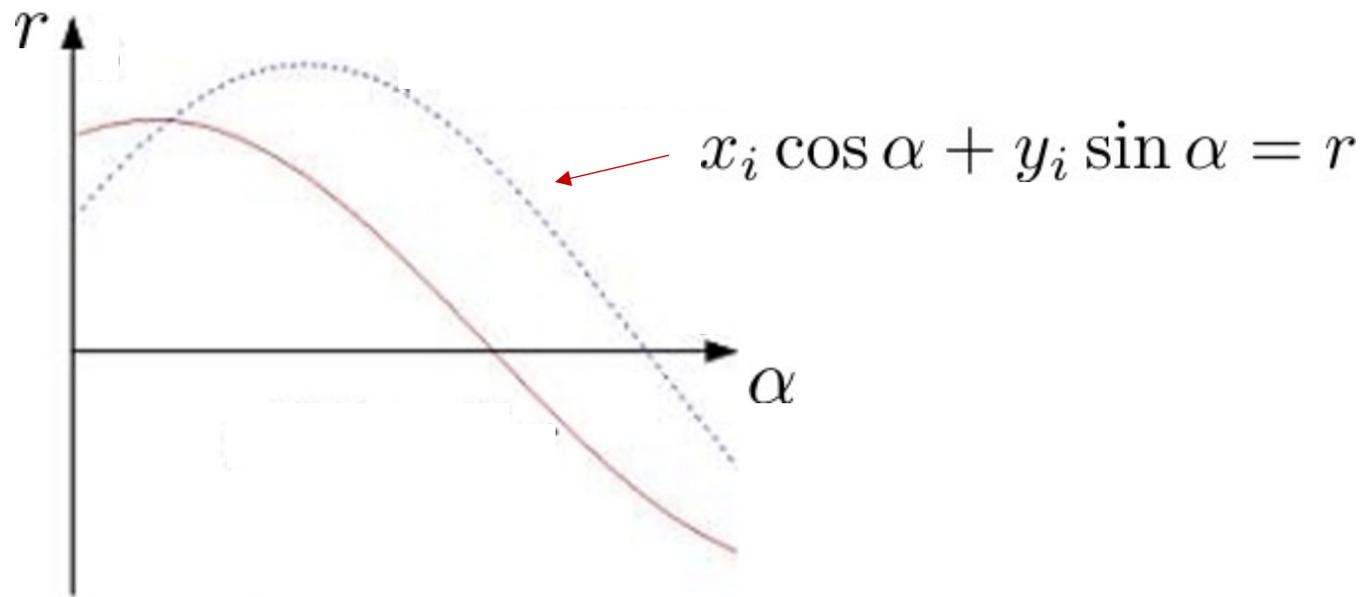


# Hough transform algorithm: polar coordinate representation

- Equation of a line in polar coordinates

$$x \cos \alpha + y \sin \alpha = r$$

- The parameter space transform of a point is a sinusoidal curve



- Avoids infinite slope
- Constant resolution

# Hough transform algorithm, revised

**Data:** Set  $S$  containing  $N$  points

**Result:** Line fitting the points in  $S$

Initialize  $n_\alpha \times n_r$  accumulator  $H$  with zeros;

**foreach**  $(x_i, y_i) \in S$  **do**

**foreach**  $\alpha \in \{\alpha_1, \dots, \alpha_{n_\alpha}\}$  **do**

        compute  $r = x_i \cos \alpha + y_i \sin \alpha$ ;

$H[\alpha, r] \leftarrow H[\alpha, r] + 1$ ;

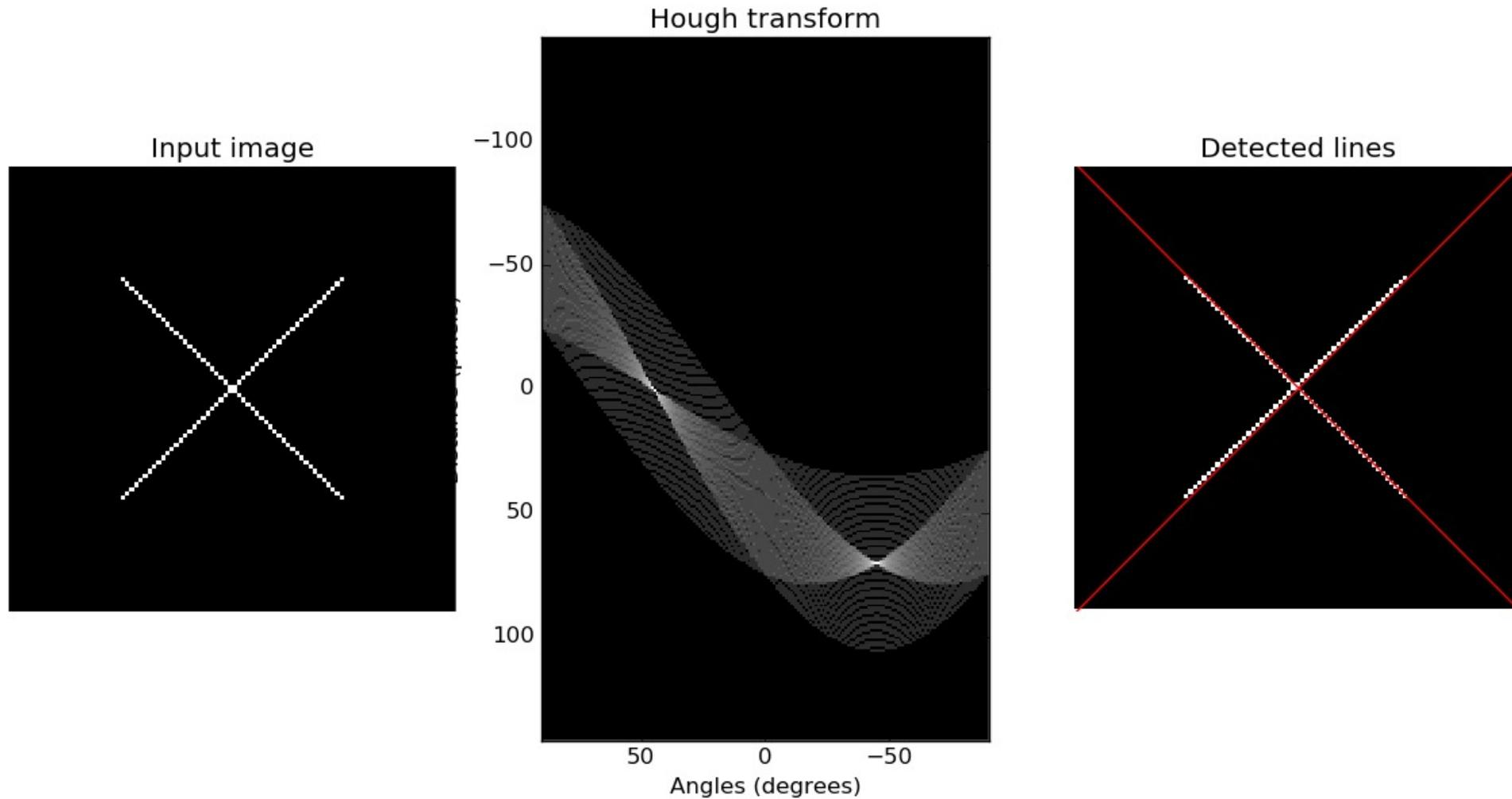
**end**

**end**

Choose  $(\alpha^*, r^*)$  that corresponds to largest count in  $H$ ;

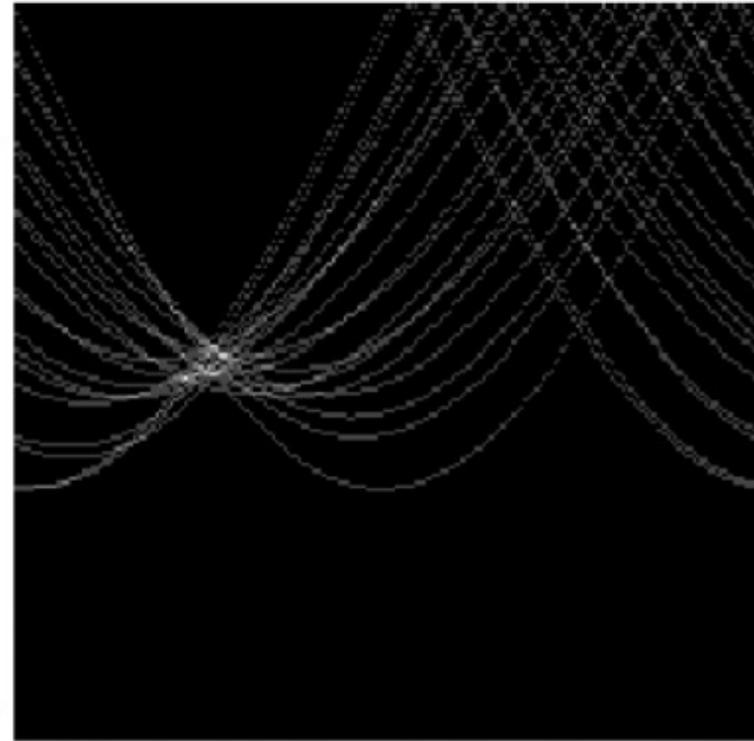
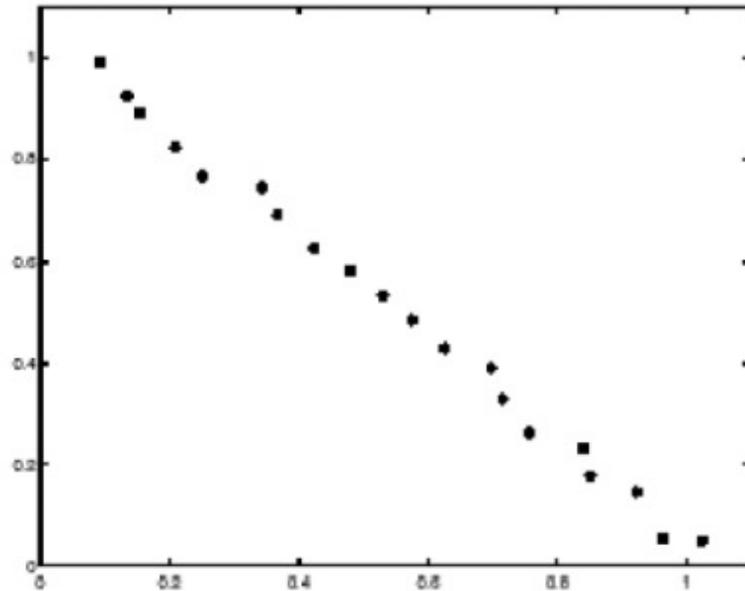
Return line defined by  $(\alpha^*, r^*)$

# Hough transform: example



# Hough transform: example

- With noise, peaks may be hard to detect



# Object recognition

- Object recognition: capability of naming discrete objects in the world
- Why is it hard? Many reasons, including:
  1. Real world is made of a jumble of objects, which all occlude one another and appear in different poses
  2. There is a lot of variability intrinsic within each class (e.g., dogs)
- In this class, we will look at three methods:
  1. Template matching
  2. Bag of visual words
  3. Neural network methods (treated as a black box, take AA274B for details)

# Template matching

- How can we find Waldo?



Source: Sanja Fidler

# Template matching

- Slide and compare!



Image I



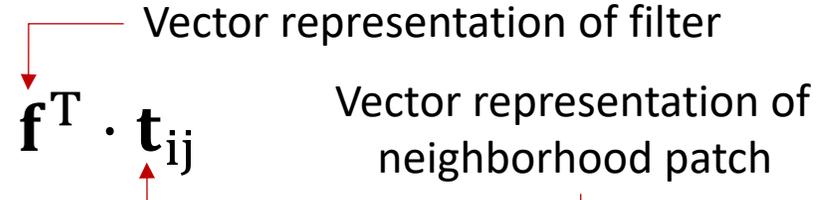
Filter F

Source: Sanja Fidler

# Template matching

- In practice, remember correlation:

$$I'(x, y) = F \circ I = \sum_{i=-N}^N \sum_{j=-M}^M F(i, j) I(x + i, y + j)$$

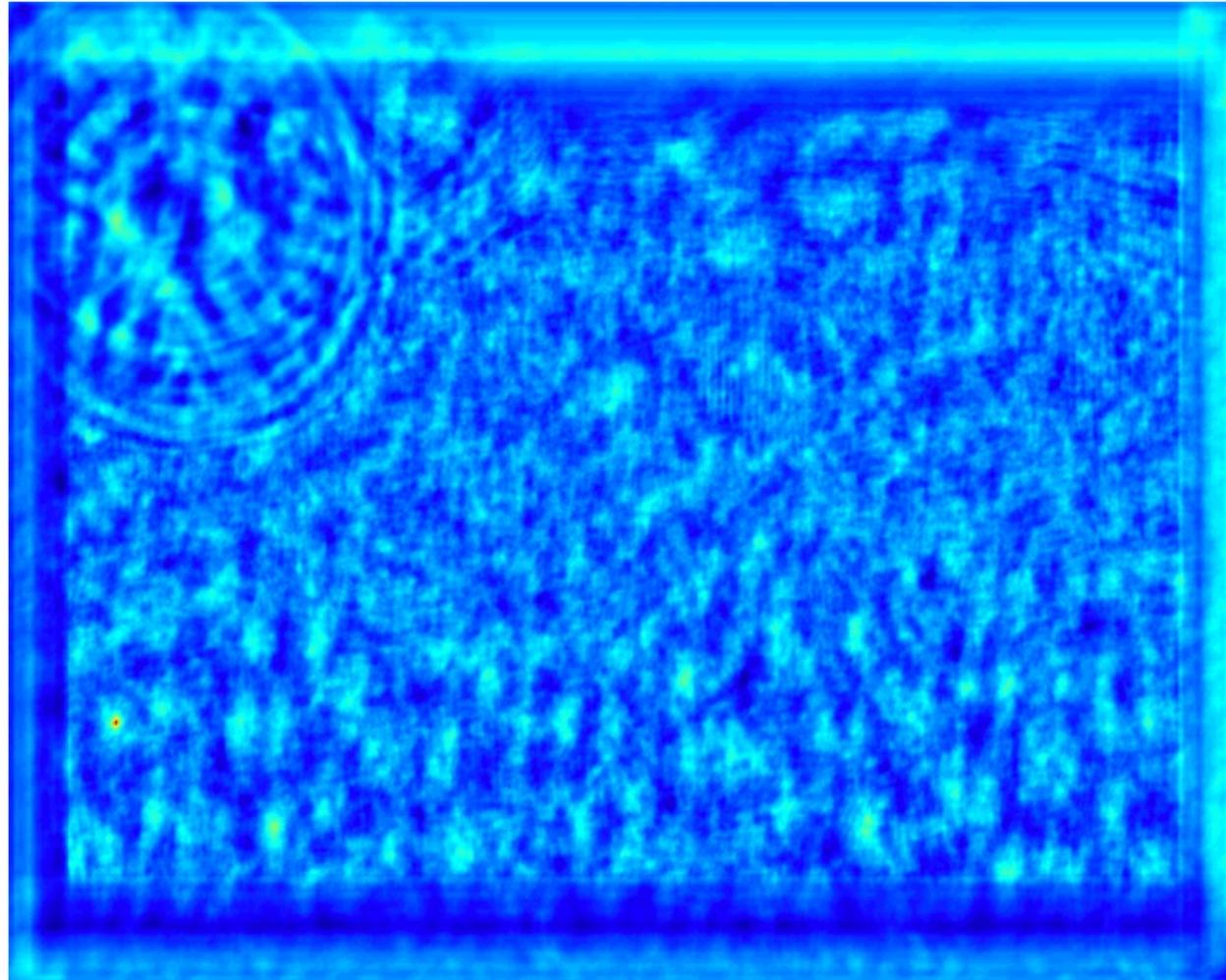
- One can equivalently write:  $I'(x, y) = \mathbf{f}^T \cdot \mathbf{t}_{ij}$   


- To ensure that perfect matching yields one, we consider *normalized* correlation, that is

$$I'(x, y) = \frac{\mathbf{f}^T \cdot \mathbf{t}_{ij}}{\|\mathbf{f}\| \|\mathbf{t}_{ij}\|}$$

# Template matching

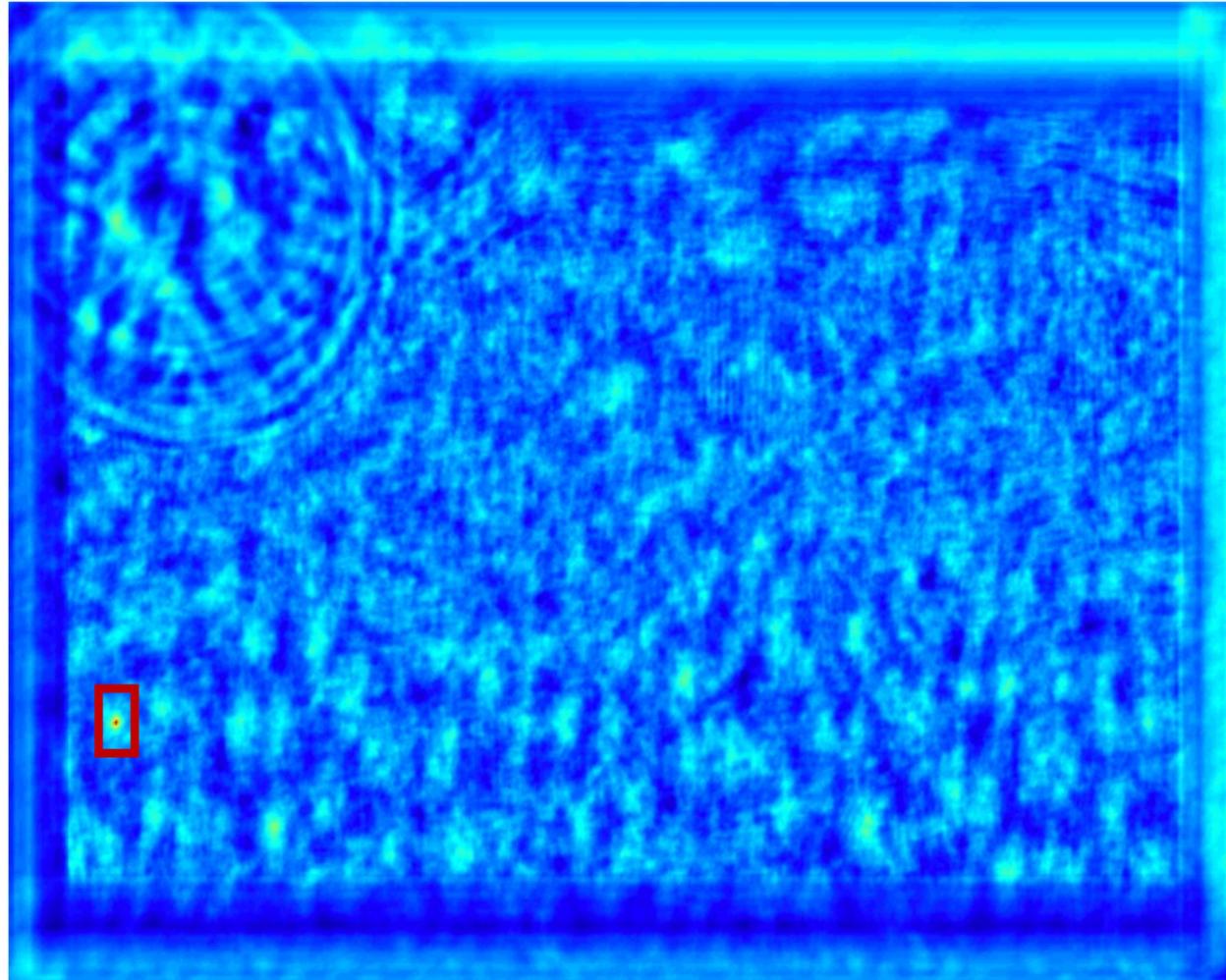
Result:



Source: Sanja Fidler

# Template matching

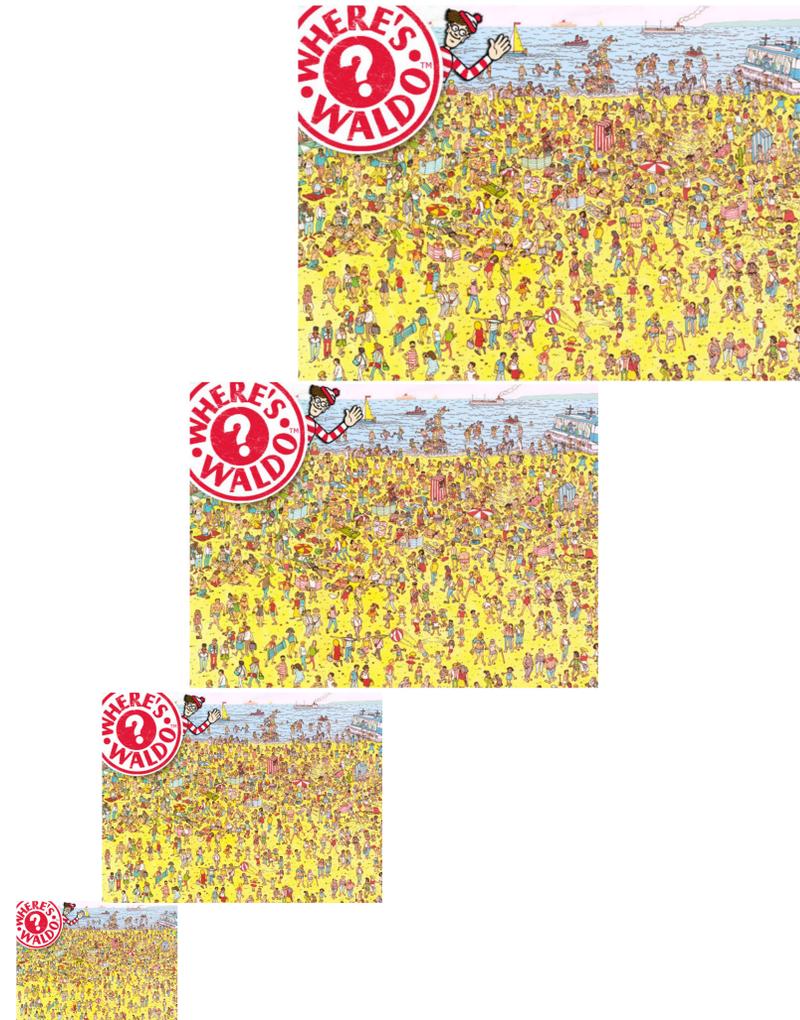
Result:



Source: Sanja Fidler

# Template matching

- Problem: what if the object in the image is much larger or much smaller than our template?
- Solution: re-scale the image multiple times, and do correlation on every size!
- This leads to the idea of *image pyramids*



# Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction



Source:  
Sanja Fidler

# Image pyramids: scaling down

- Naïve solution: keep only some rows and columns
- E.g.: keep every other column to reduce image by 1/2 in width direction



Source:  
Sanja Fidler

# Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



Source:  
Sanja Fidler

# Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



Source:  
Sanja Fidler

# Image pyramids: scaling down

- Solution: blur the image via Gaussian, *then* subsample
- Intuition: remove high frequency content in the image



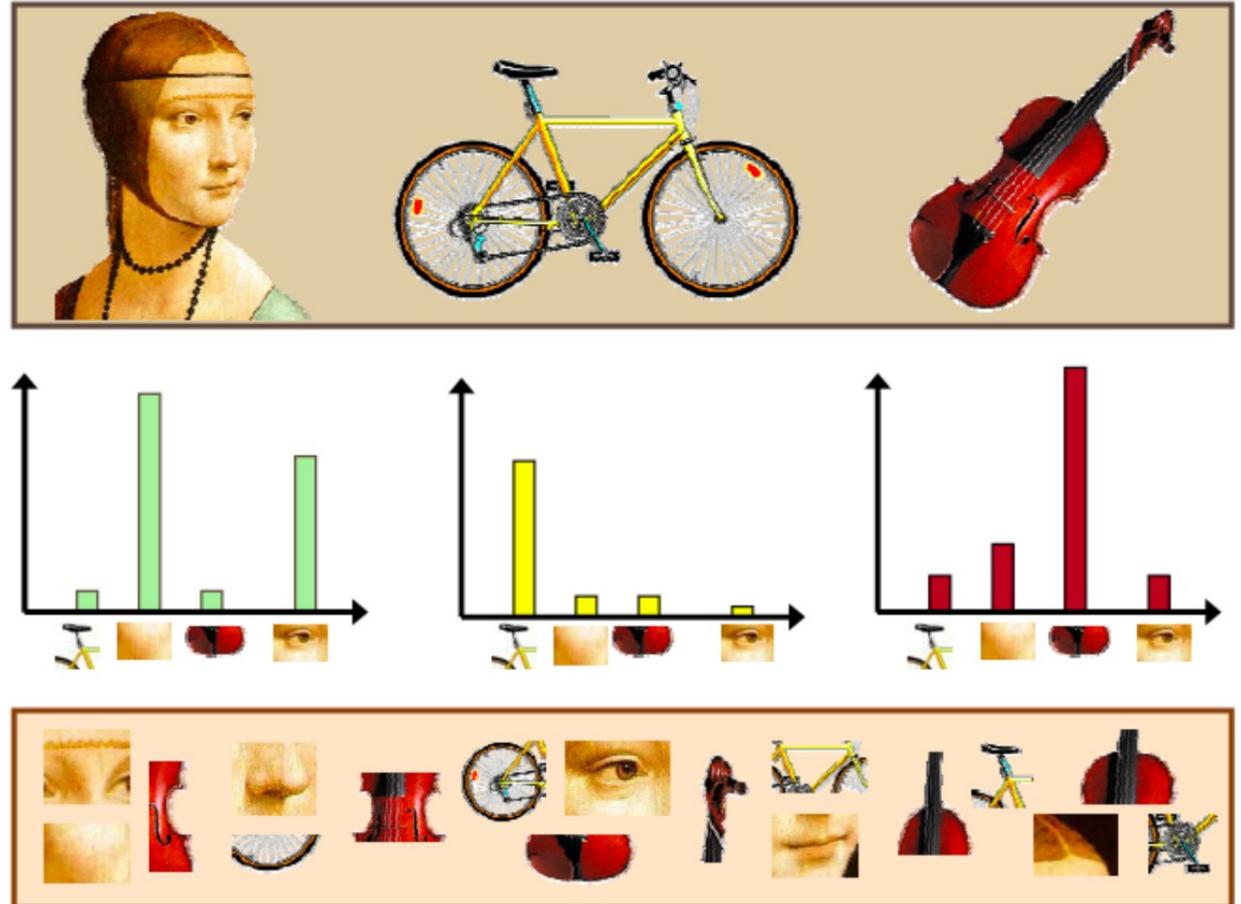
Source:  
Sanja Fidler

# Image pyramids

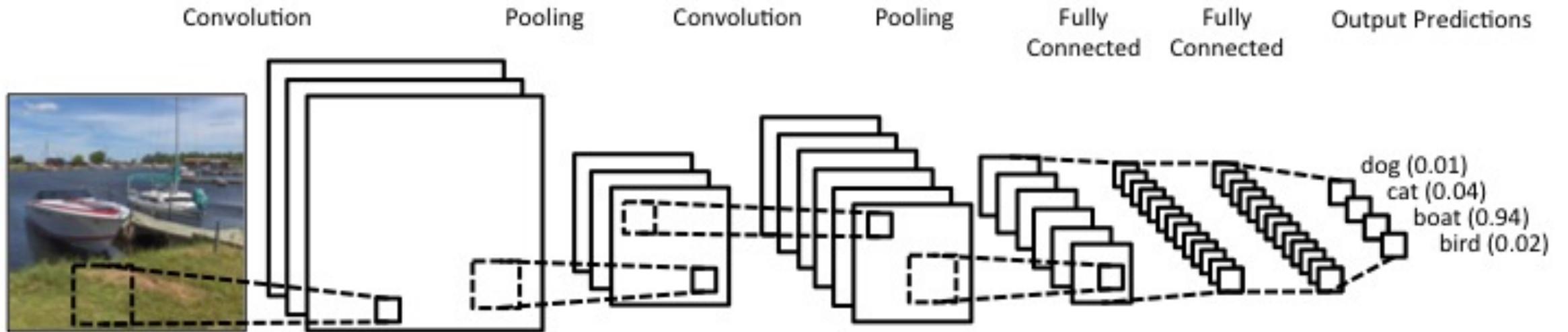
- A sequence of images created with Gaussian blurring and down-sampling is called a Gaussian pyramid
- The other step is to perform up-sampling (nearest neighbor, bilinear, bicubic, etc).

# Bags of Visual Words

- Key idea: compute the distribution (histogram) of *visual words* found in the query image
- Compare this distribution to those found in the training images in order to perform classification



# A different paradigm: using CNNs for recognition



# Next time

