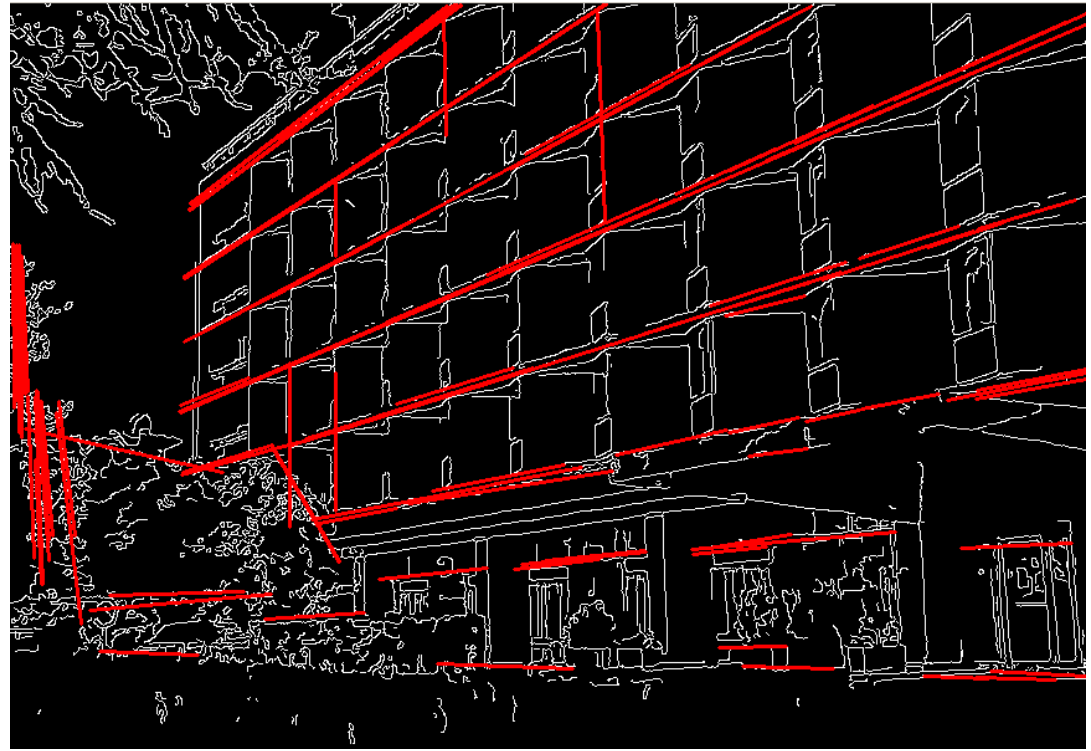
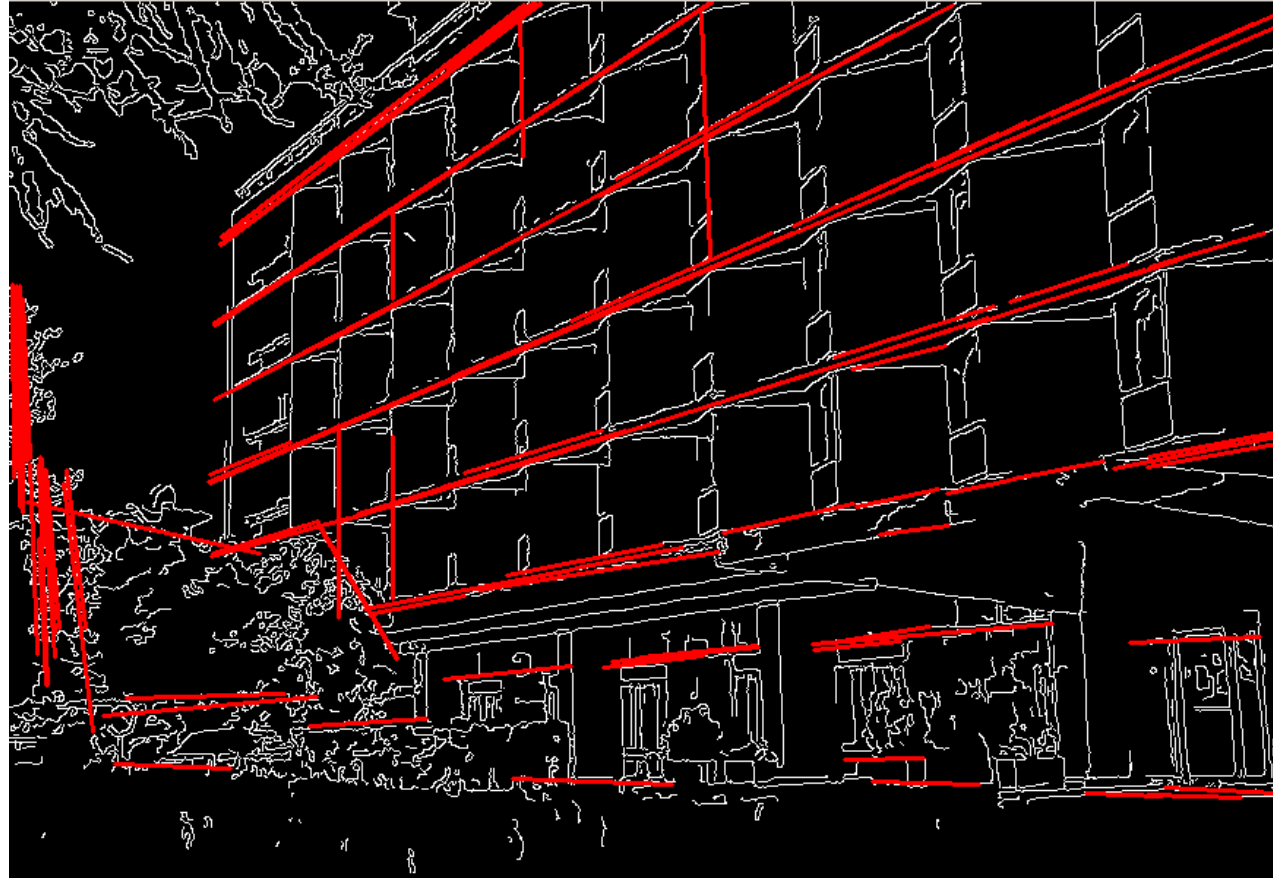


# Fitting



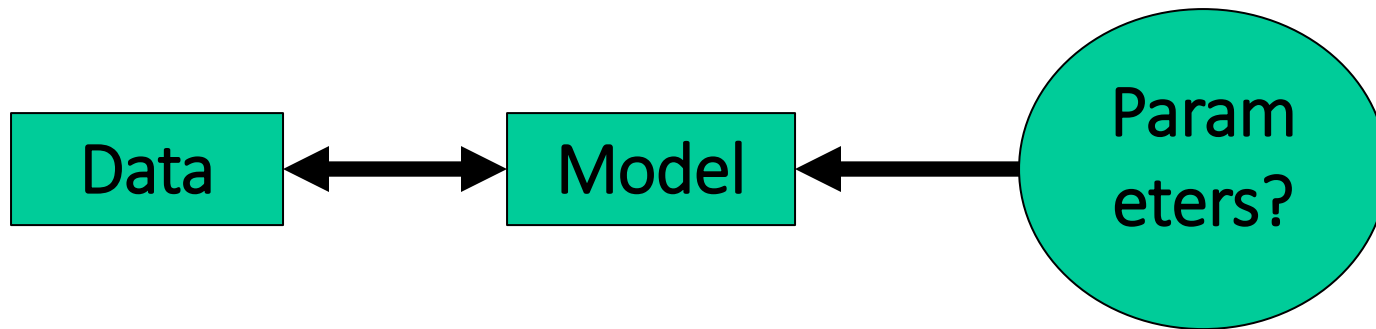
Slides are from Svetlana Lazebnik

# Fitting



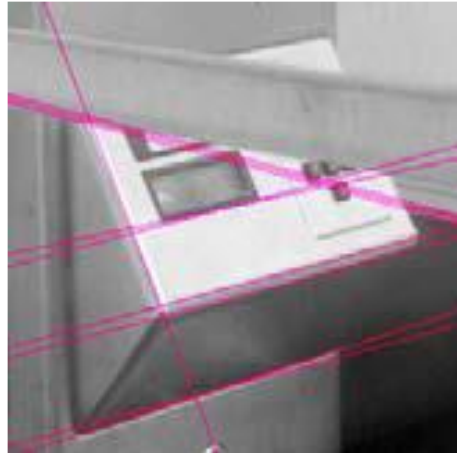
# Fitting

- We've learned how to detect edges, corners, blobs. Now what?
- We would like to form a higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model



# Fitting

- Choose a *parametric model* to represent a set of features



simple model: lines



simple model: circles



complicated model: car

# Fitting: Challenges

Case study: Line detection



- **Noise** in the measured feature locations
- **Extraneous data:** clutter (outliers), multiple lines
- **Missing data:** occlusions
- **Intra-class variability**

# Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC
- Hough transform
- Generalized Hough transform



# Least squares line fitting: First attempt

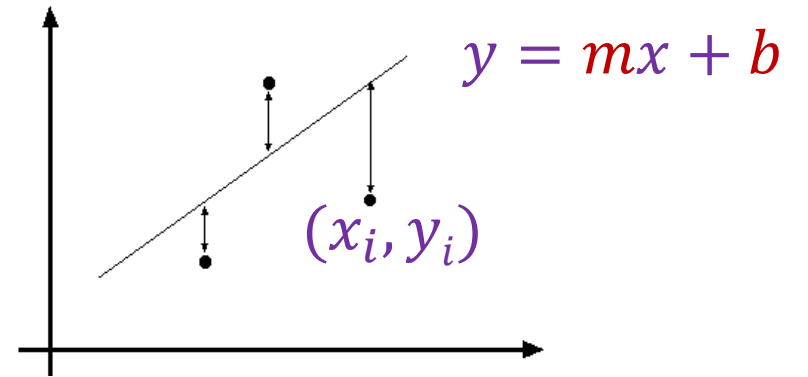
- Data:  $(x_1, y_1), \dots, (x_n, y_n)$
- Line equation:  $y_i = mx_i + b$
- Find  $(m, b)$  to minimize

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

- Equivalent to finding least-squares solution to:

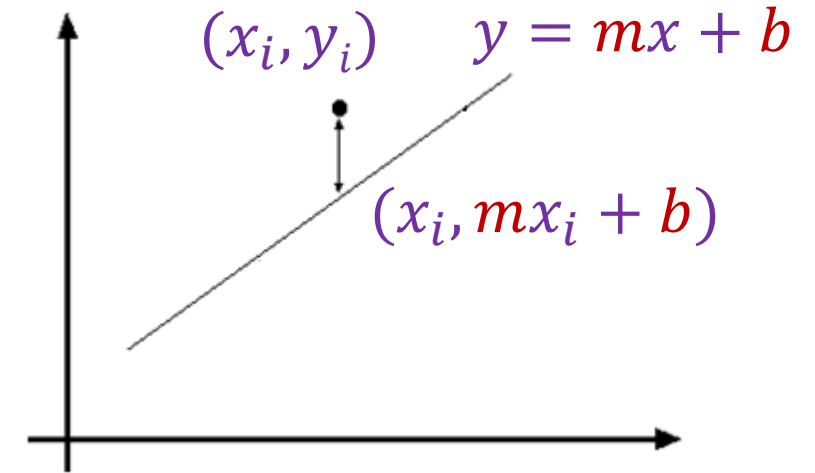
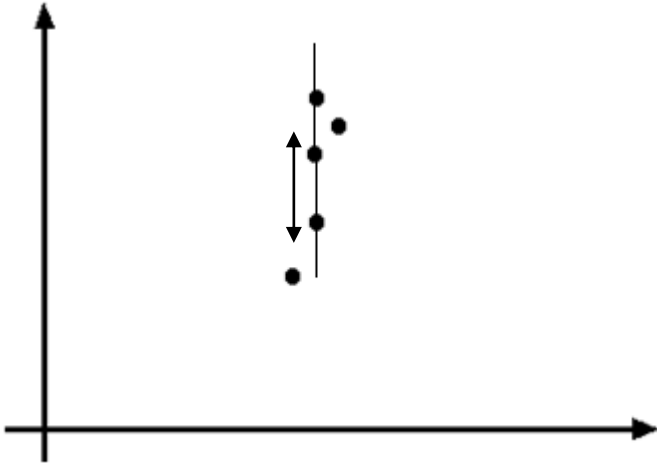
$$\begin{array}{ccc} \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} & \begin{pmatrix} m \\ b \end{pmatrix} & = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \\ X & a & y \end{array}$$

- Solution is given by  $\min_a \|Xa - y\|^2 \Rightarrow X^T Xa = X^T y$
- $a = (X^T X)^{-1} X^T y$



# Is this a good solution?

- Slope-intercept parametrization fails for vertical lines



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

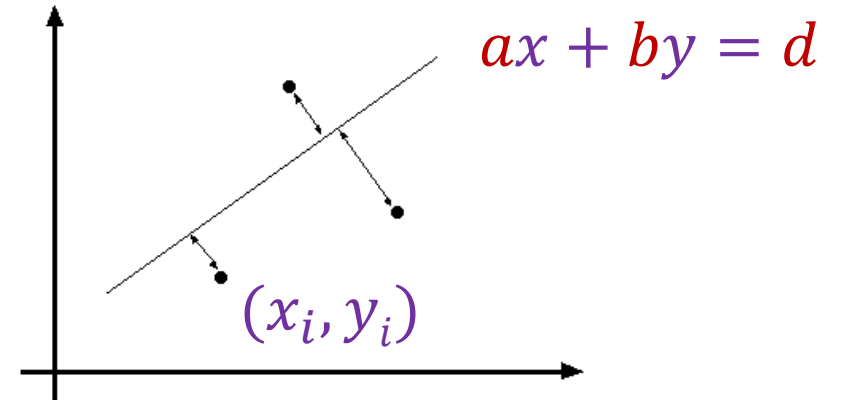
- Minimizes vertical distances, We can not fit vertical/near vertical lines
- Solution is not equivariant w.r.t. rotation
- How to improve? Any ideas?



# Total least squares

- Line parametrization:  $ax + by = d$ 
  - $(a, b)$  is the *unit normal* to the line (i.e.,  $a^2 + b^2 = 1$ )
  - $d$  is the distance between the line and the origin
- **Perpendicular distance** between point  $(x_i, y_i)$  and line  $ax + by = d$  (assuming  $a^2 + b^2 = 1$ )\*:  
 $|ax_i + by_i - d|$
- Objective function:

$$E(a, b, d) = \sum_{i=1}^n (ax_i + by_i - d)^2$$



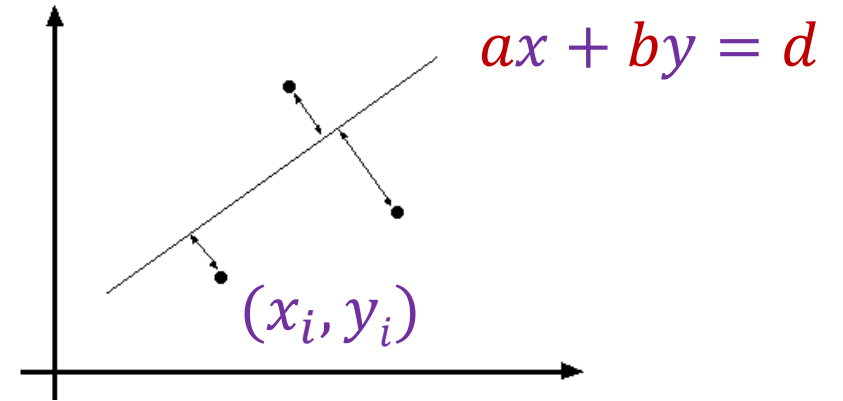
# Total least squares

$$E(a, b, d) = \sum_{i=1}^n (ax_i + by_i - d)^2$$

- Solve for  $d$  first:

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

$$d = \frac{a}{n} \sum_{i=1}^n x_i + \frac{b}{n} \sum_{i=1}^n y_i = a\bar{x} + b\bar{y}, \text{ where } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$$

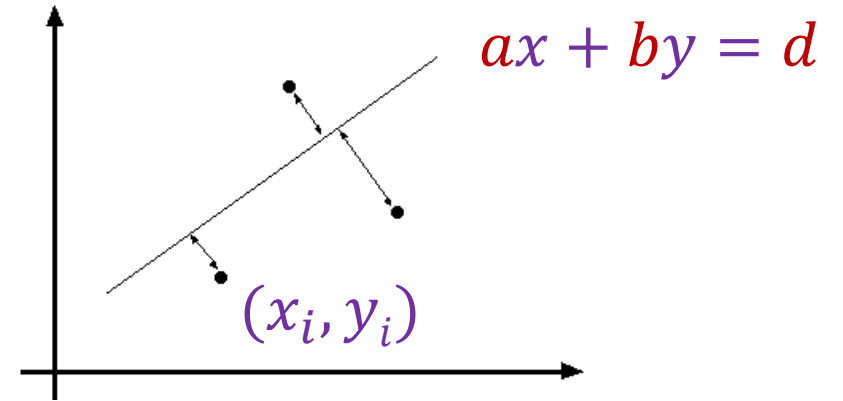


# Total least squares

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

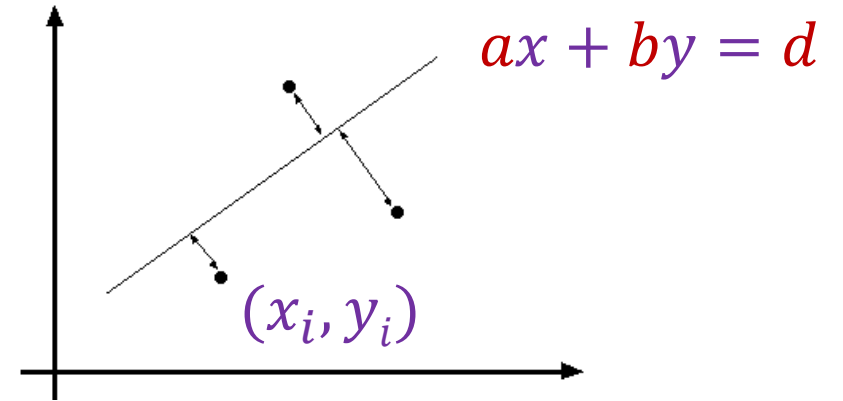
- Solve for  $d$  first:  $d = a\bar{x} + b\bar{y}$
- Plugging back in:

$$E = \sum_{i=1}^n (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 =$$



# Total least squares

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



- Solve for  $d$  first:  $d = a\bar{x} + b\bar{y}$
- Plugging back in:

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

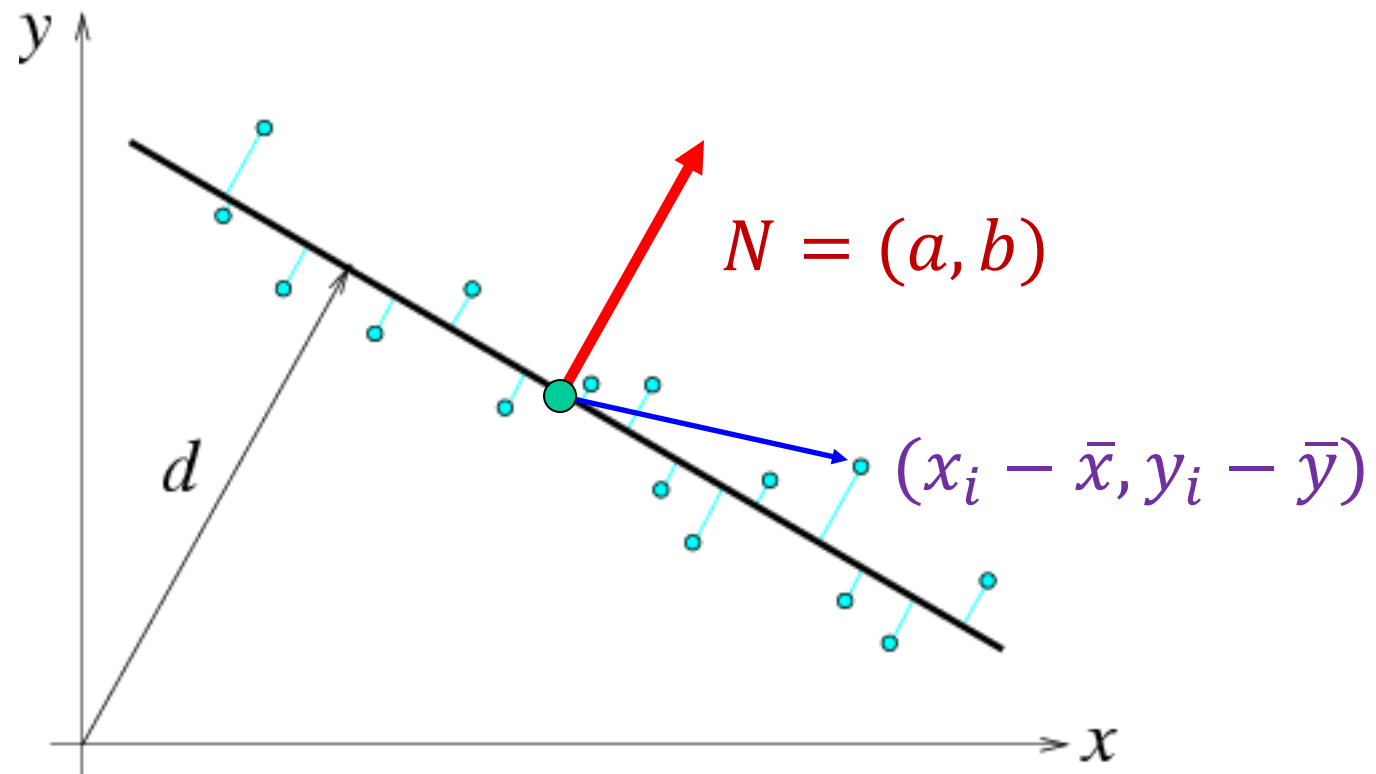
$U$   $N$

- We want to find  $N$  that minimizes  $\|UN\|^2$  subject to  $\|N\|^2 = 1$ 
  - Solution is given by the eigenvector of  $U^T U$  associated with the smallest eigenvalue

# Total least squares

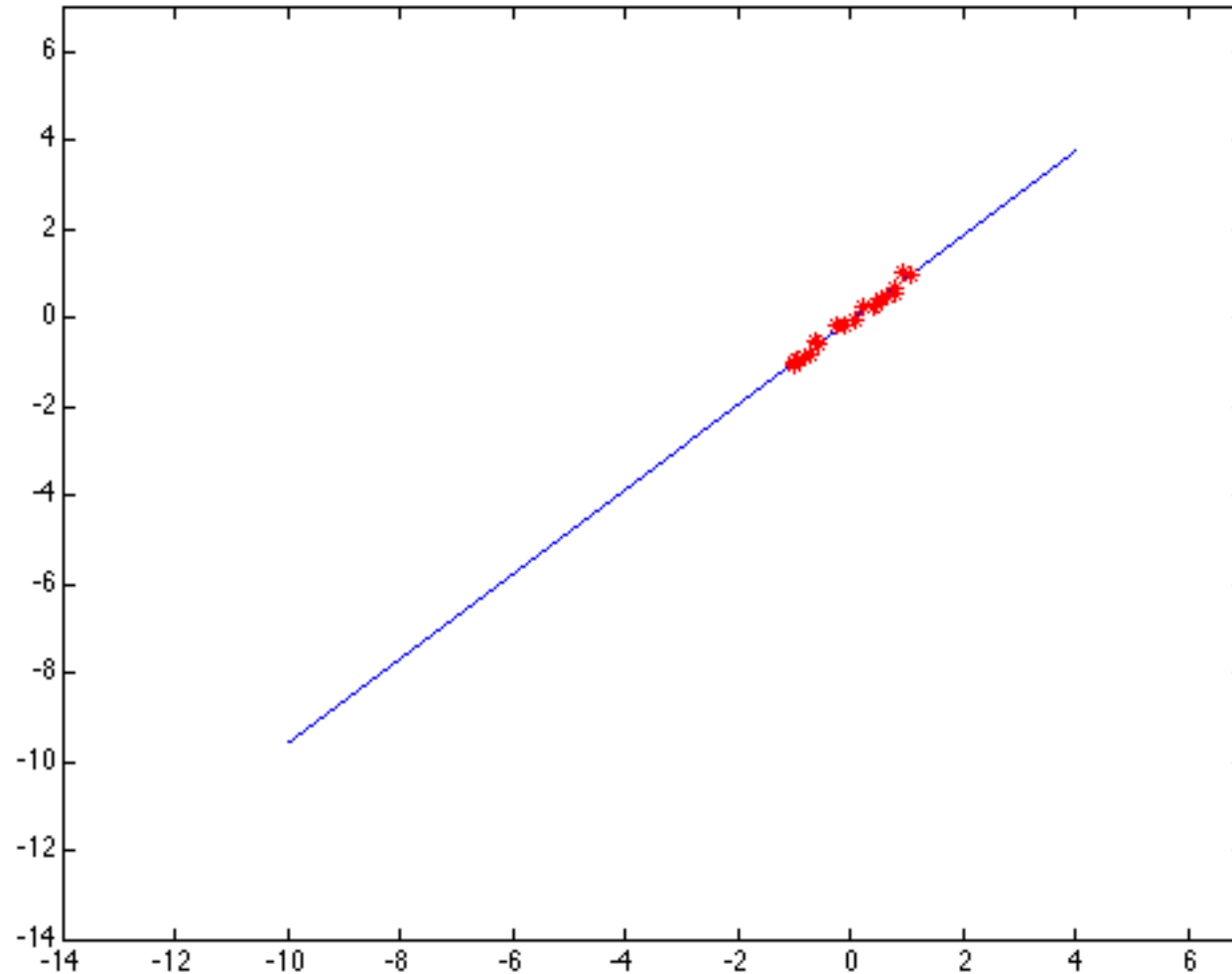
$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \quad U^T U = \begin{bmatrix} \sum_{i=1}^n (x_i - \bar{x})^2 & \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^n (y_i - \bar{y})^2 \end{bmatrix}$$

second moment matrix



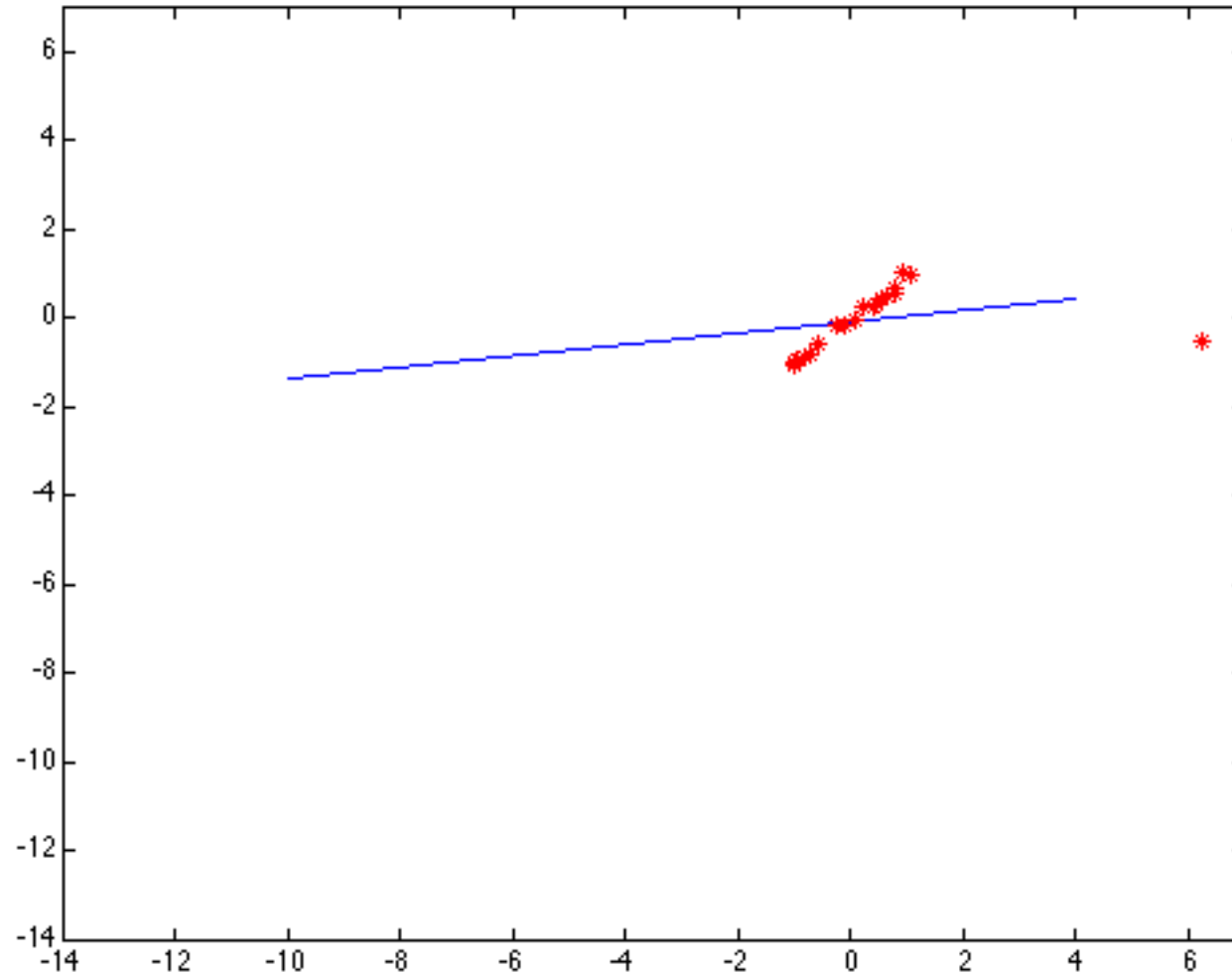
## Least squares: Robustness to noise

- Least squares fit to the red points:



# Least squares: Robustness to noise

- Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers



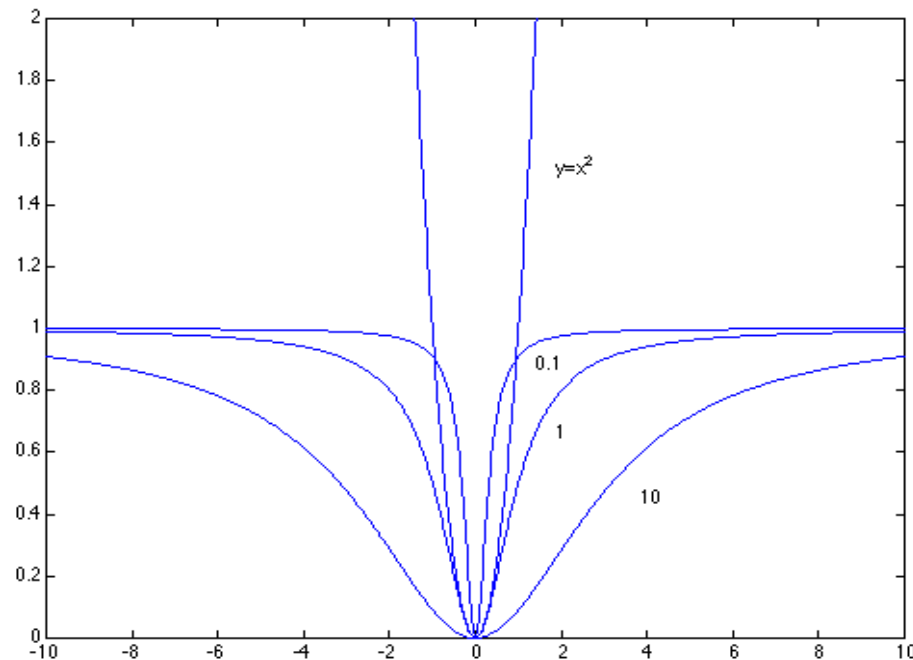
# Robust estimators

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

$$\sum_i \rho_\sigma(r(x_i; \theta))$$

$r(x_i; \theta)$ : residual of  $x_i$  w.r.t. model parameters  $\theta$

$\rho_\sigma$ : robust function with scale parameter  $\sigma$ , e.g.,  $\rho_\sigma(u) = \frac{u^2}{\sigma^2 + u^2}$



# Robust estimators

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

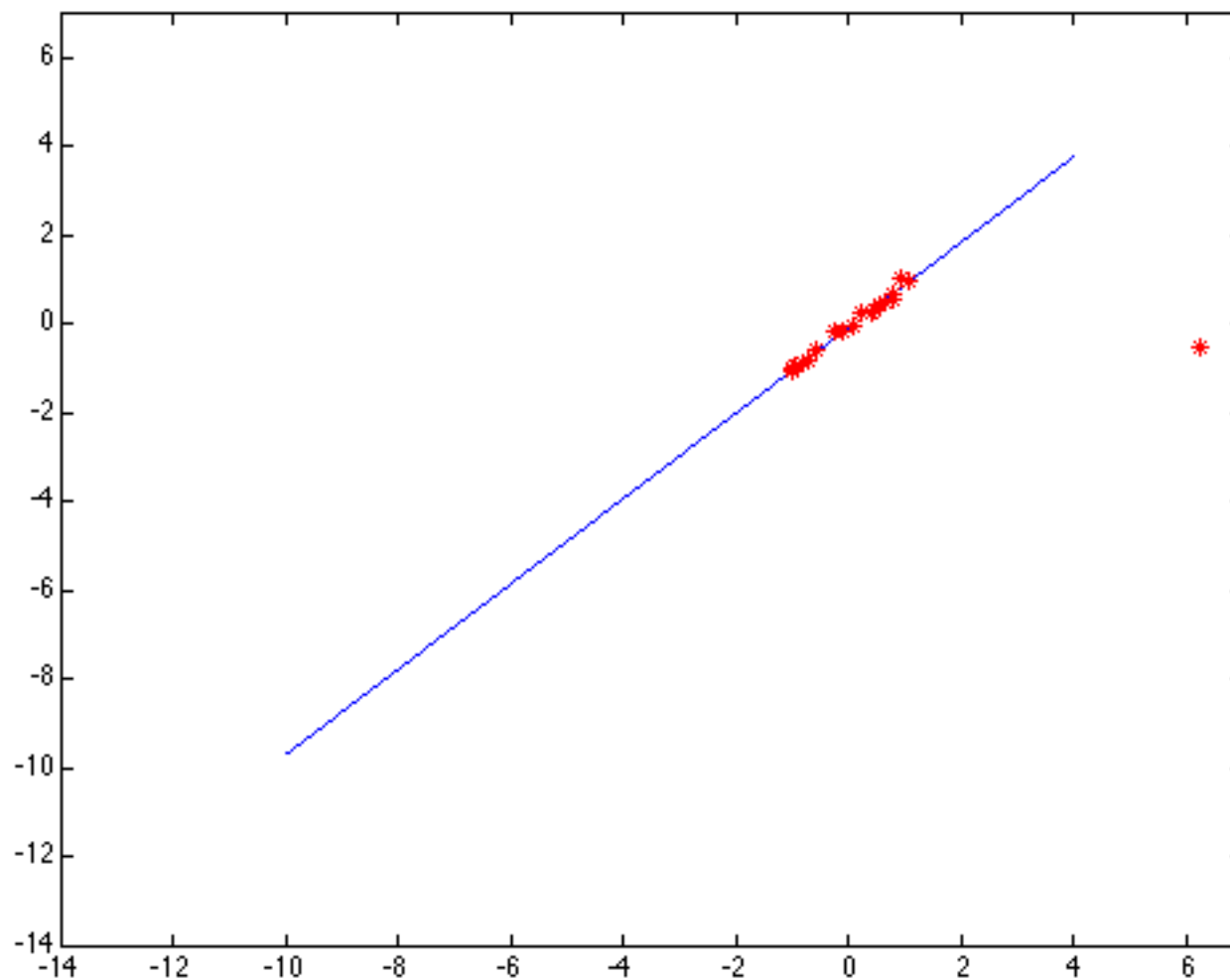
$$\sum_i \rho_{\sigma}(r(x_i; \theta))$$

$r(x_i; \theta)$ : residual of  $x_i$  w.r.t. model parameters  $\theta$

$\rho_{\sigma}$ : robust function with scale parameter  $\sigma$ , e.g.,  $\rho_{\sigma}(u) = \frac{u^2}{\sigma^2 + u^2}$

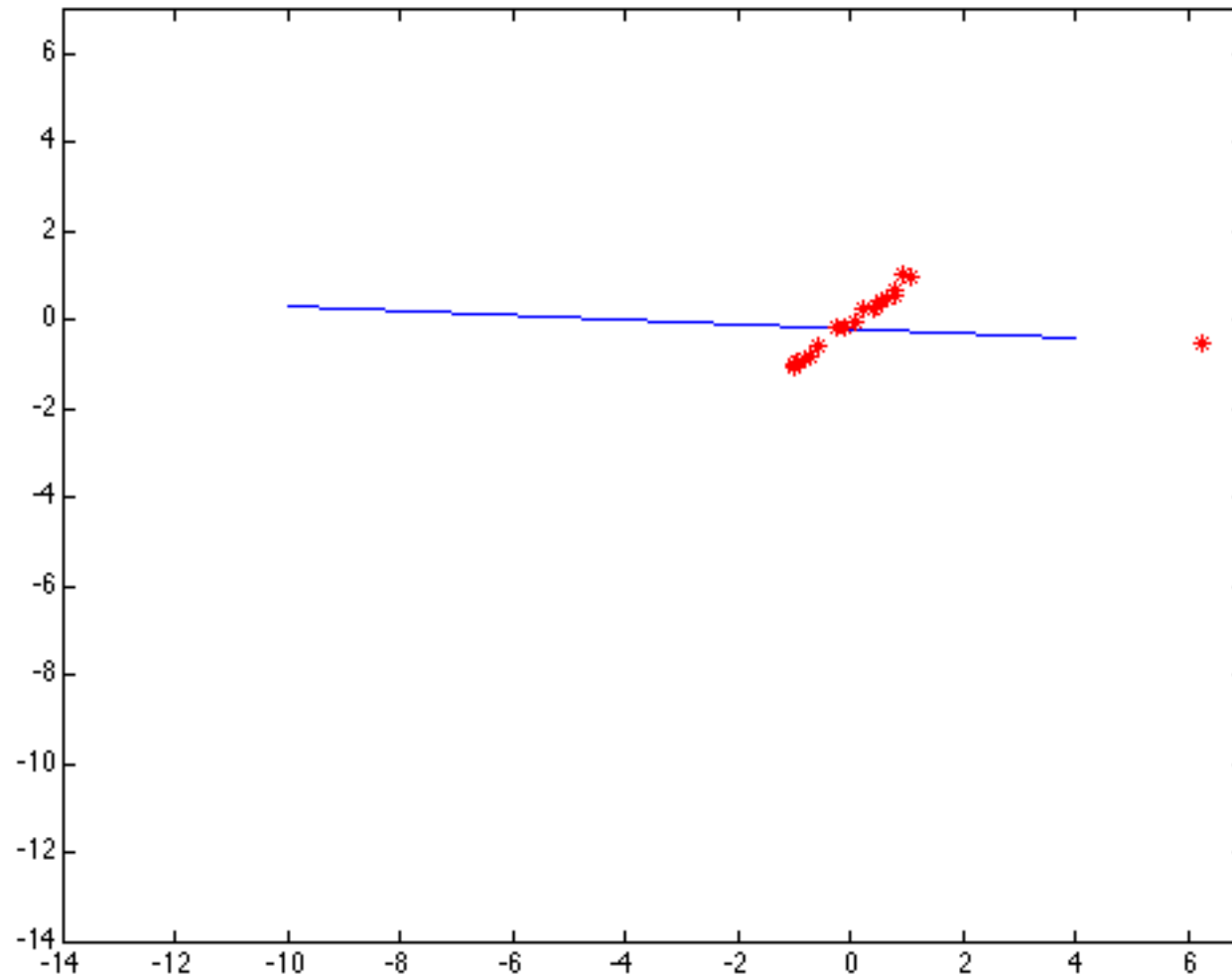
- Nonlinear optimization problem that must be solved iteratively
  - Least squares solution can be used for initialization
  - Scale of robust function should be chosen carefully

## Choosing the scale: Just right



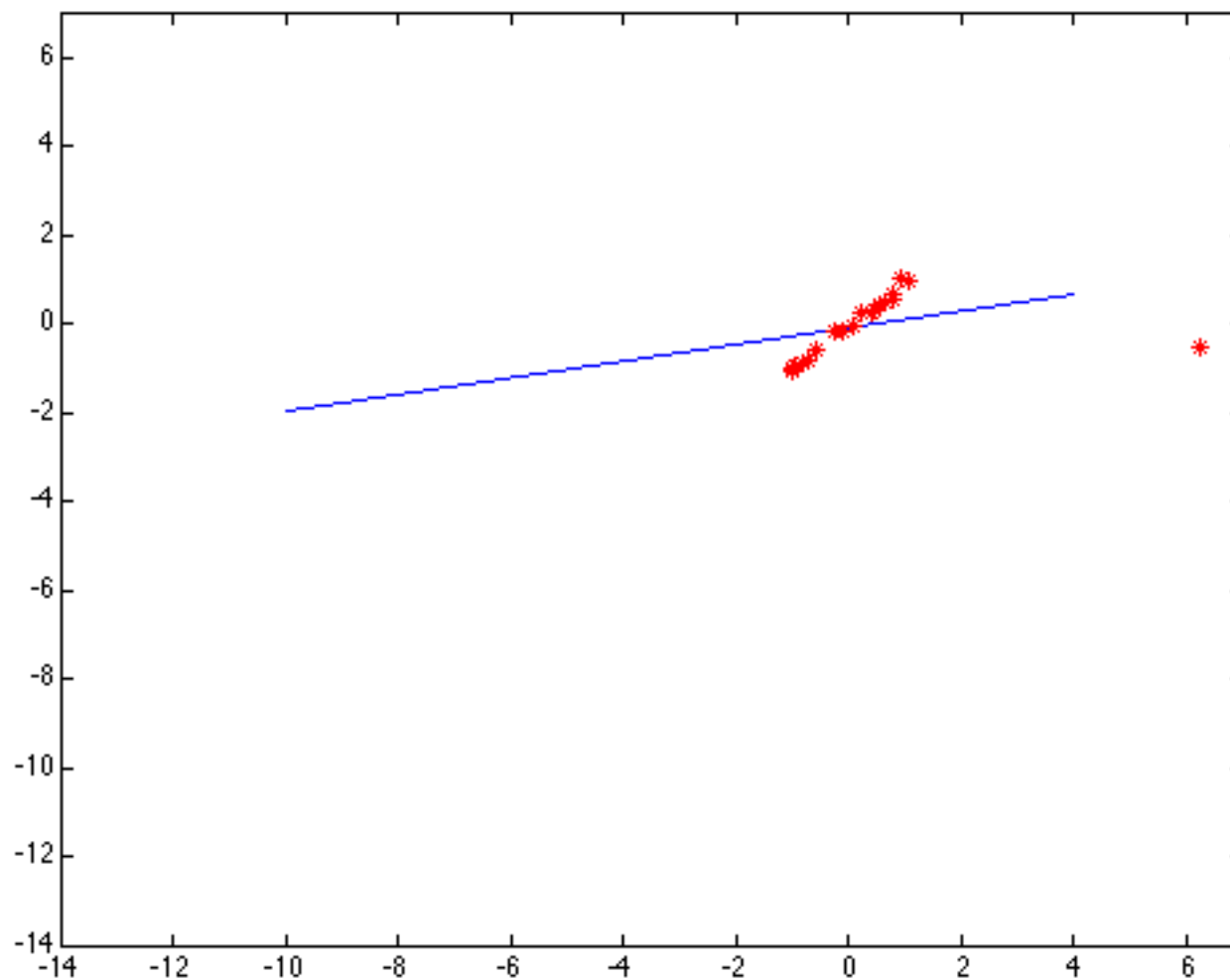
The effect of the outlier is minimized

## Choosing the scale: Too small



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

## Choosing the scale: Too large



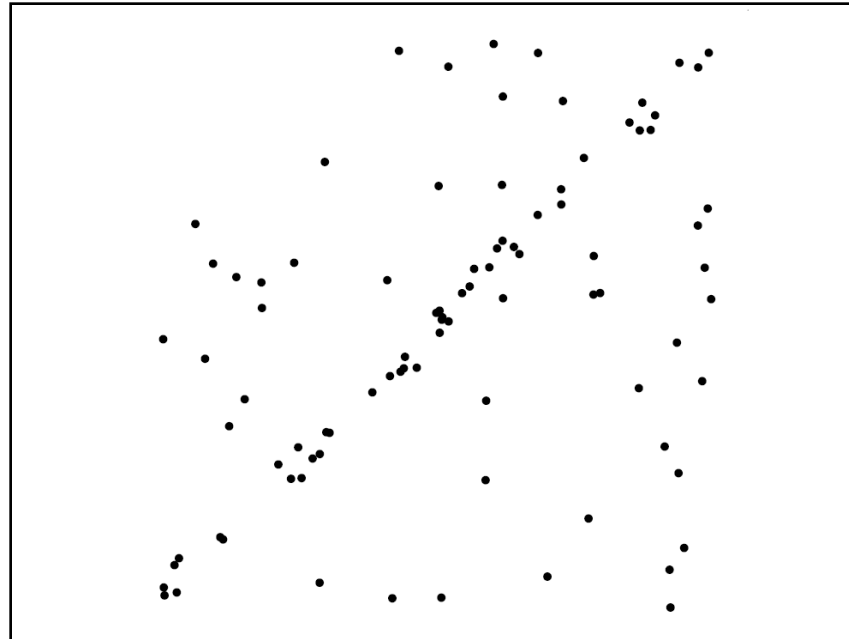
Behaves much the same as least squares

# Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC

# Voting schemes

- Robust fitting can deal with a few outliers – what if we have very many?
- Basic idea: let each point *vote* for all the models that are compatible with it
  - Hopefully the outliers will not vote consistently for any single model
  - The model that receives the most votes is the best fit to the image

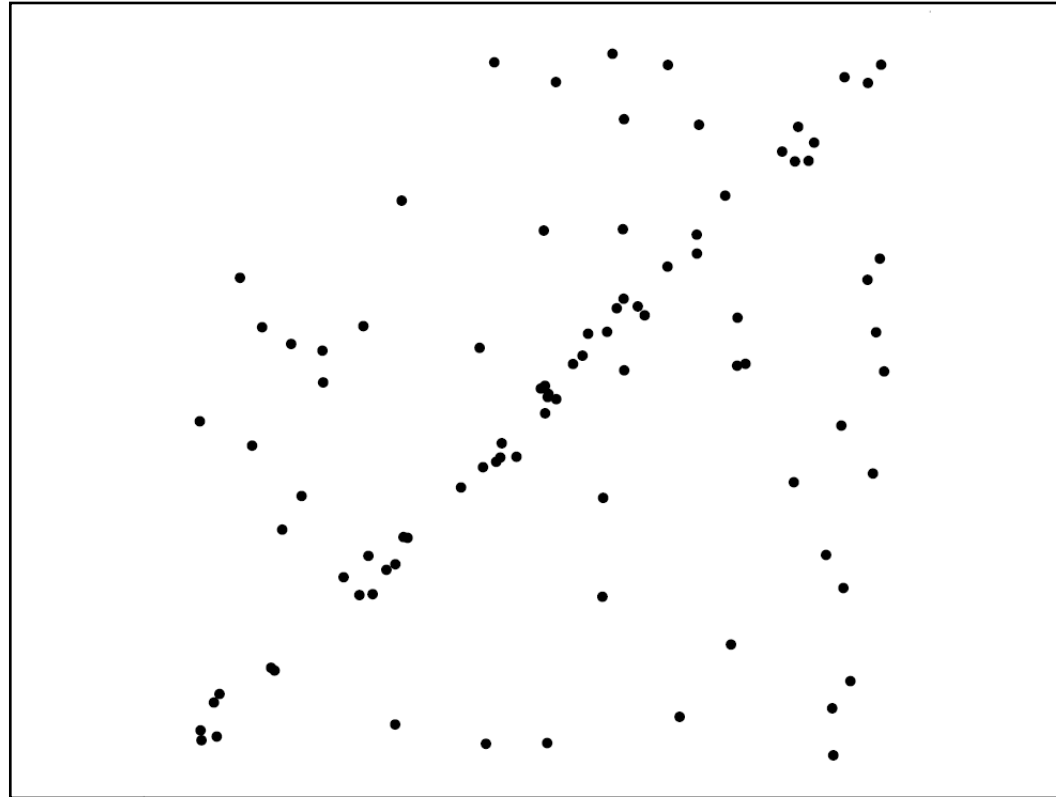




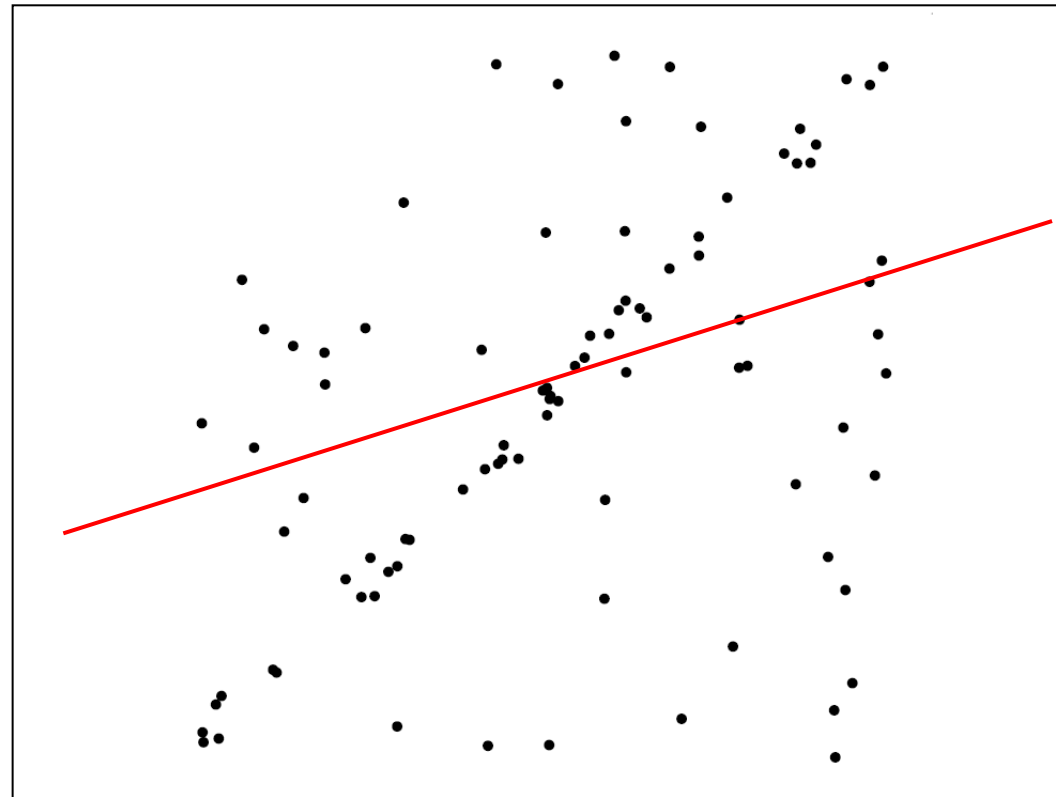
# RANSAC

- Random sample consensus: very general framework for model fitting in the presence of outliers
- Outline:
  - Randomly choose a small initial subset of points
  - Fit a model to that subset
  - Find all inlier points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the model with the most inliers

# RANSAC for line fitting example

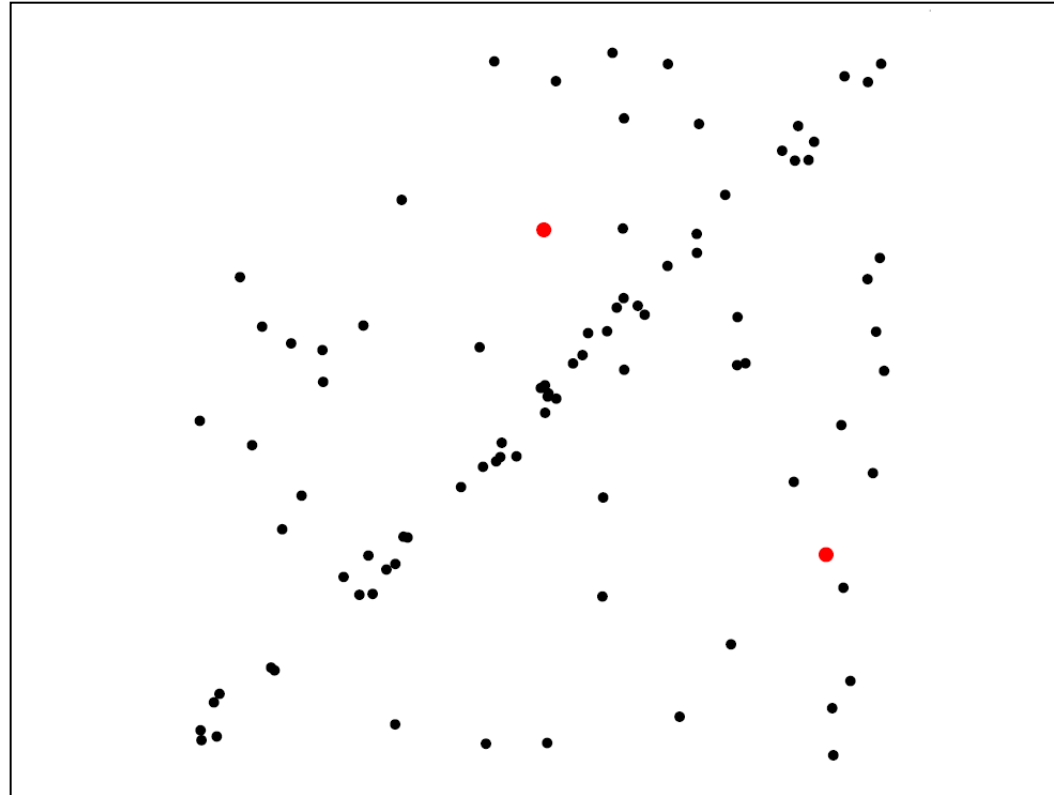


# RANSAC for line fitting example



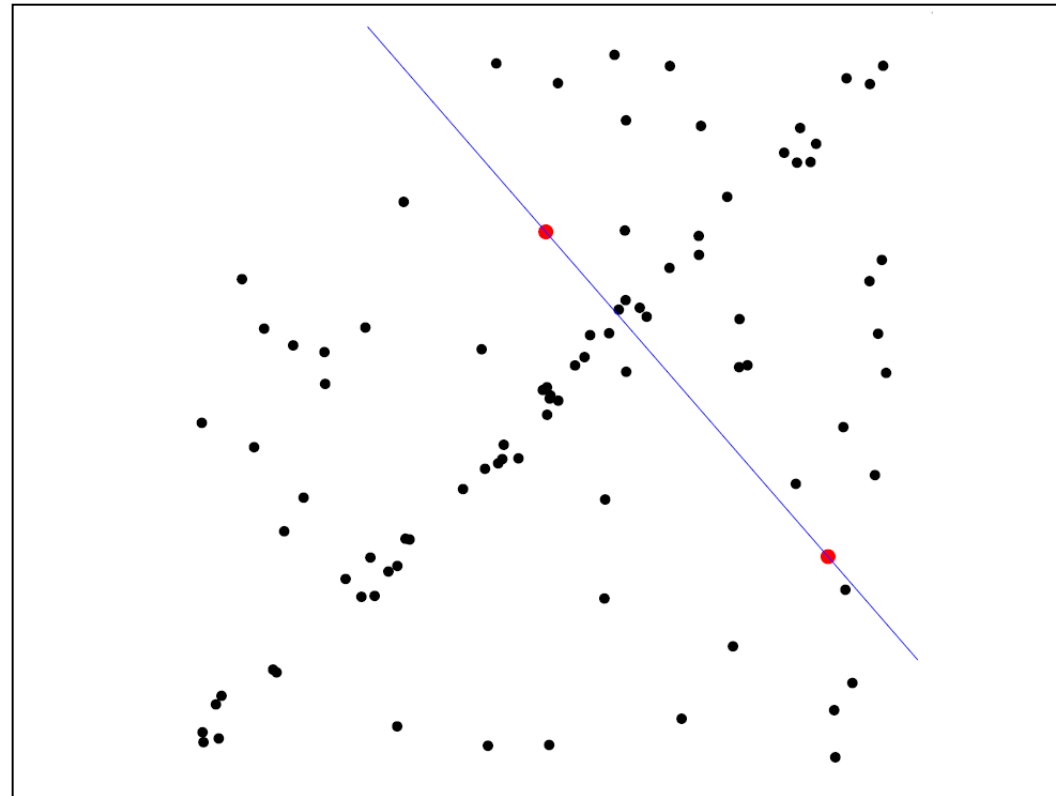
Least-squares fit

# RANSAC for line fitting example



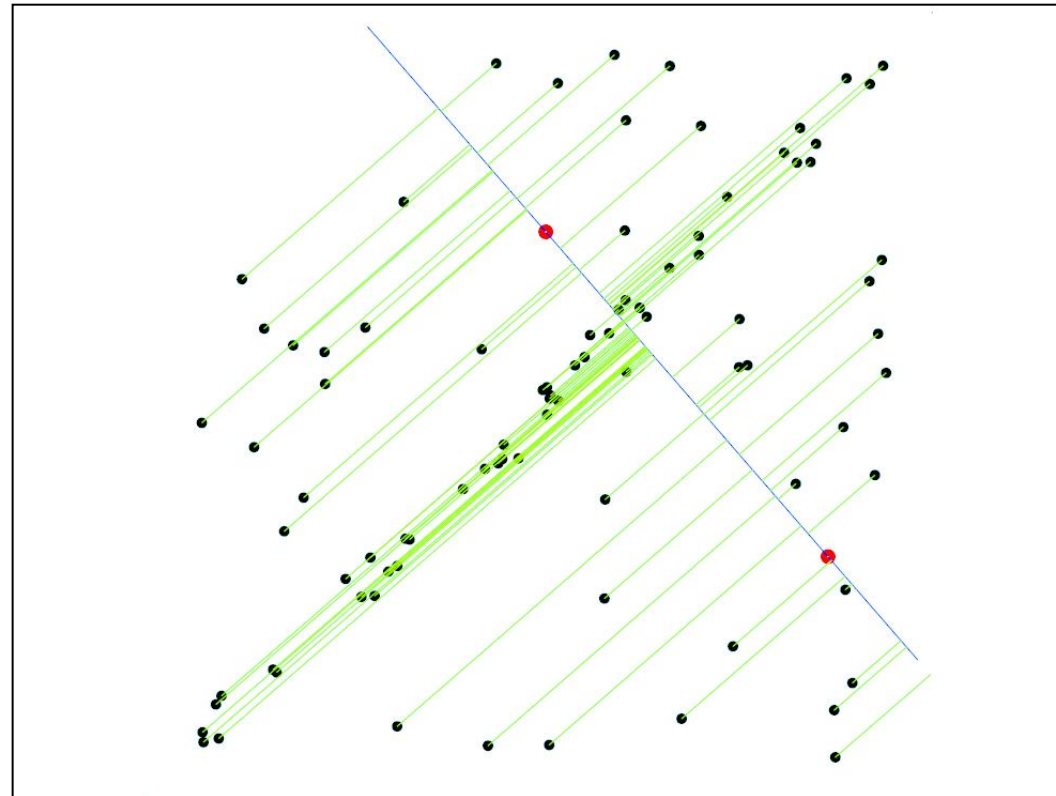
1. Randomly select minimal subset of points

# RANSAC for line fitting example



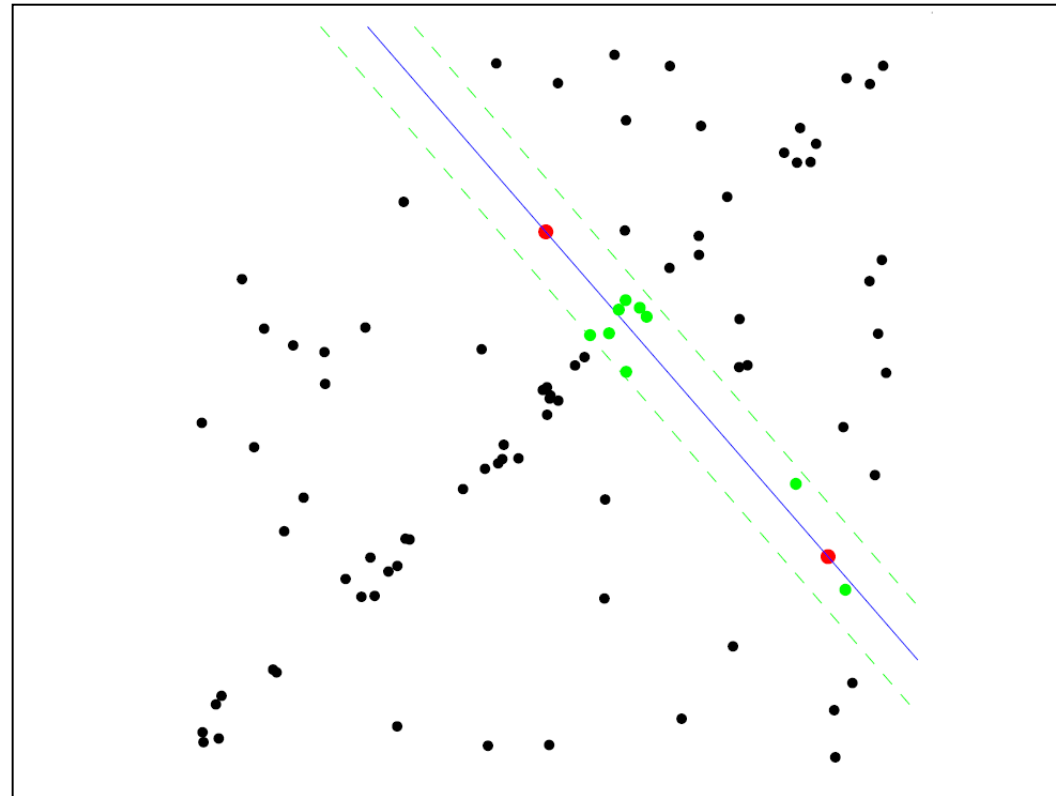
1. Randomly select minimal subset of points
2. Hypothesize a model

# RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

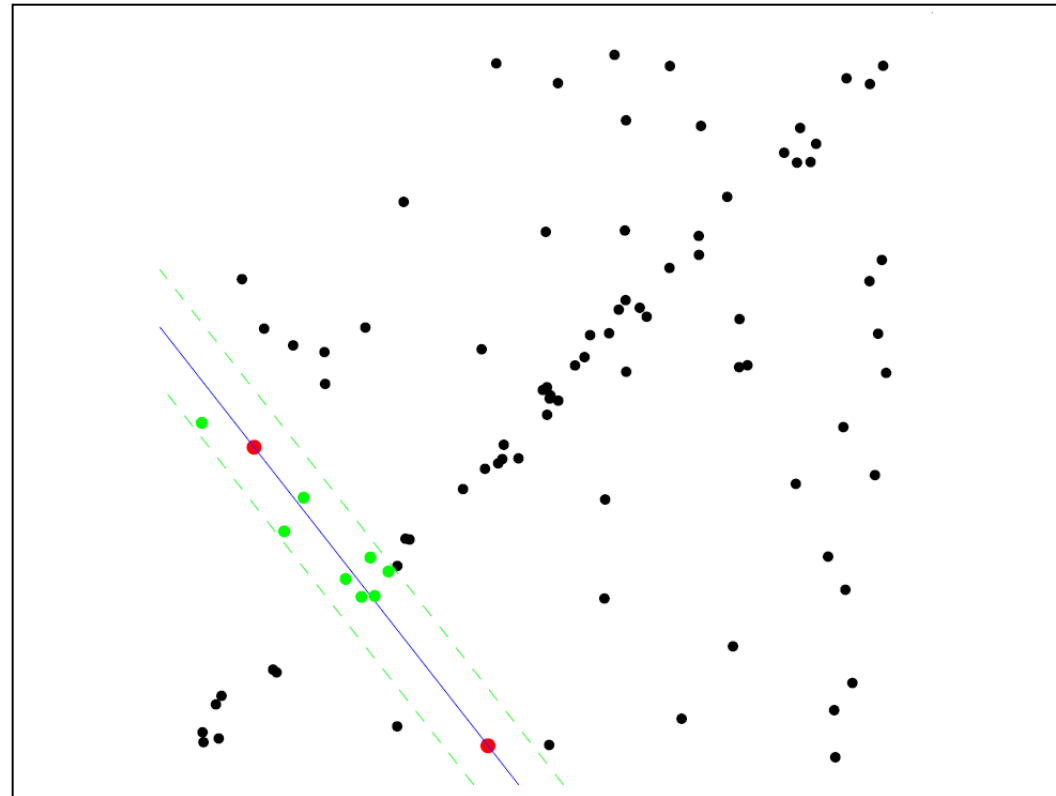
# RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

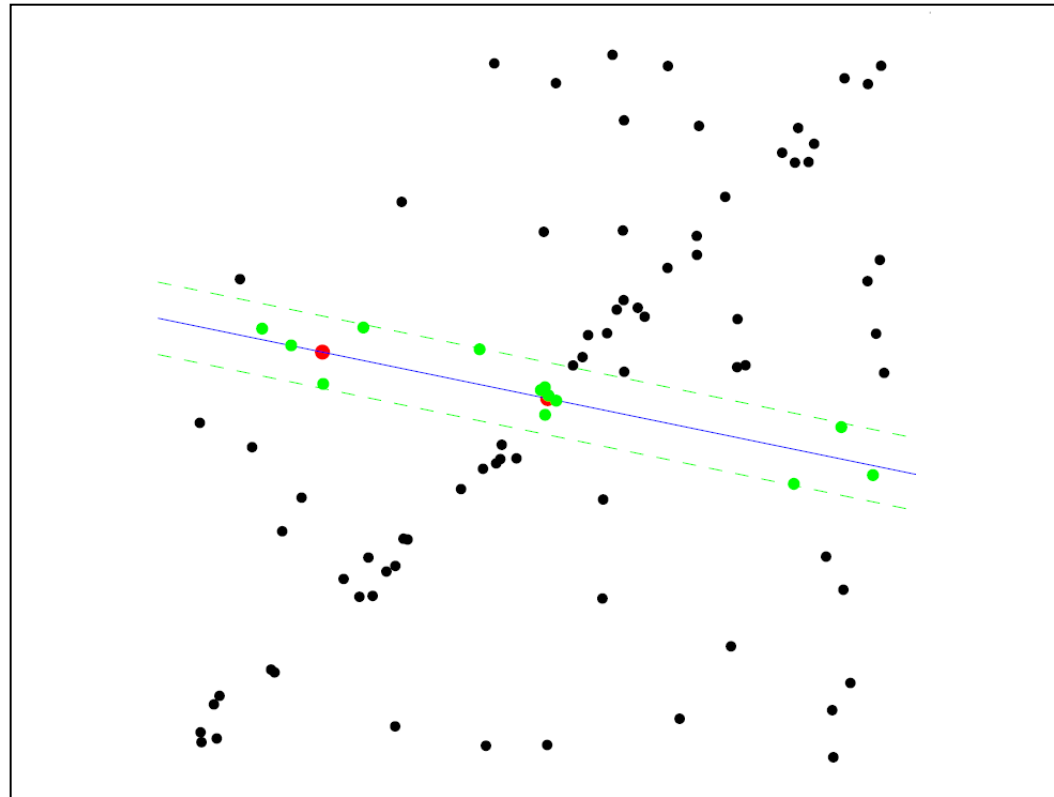


# RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

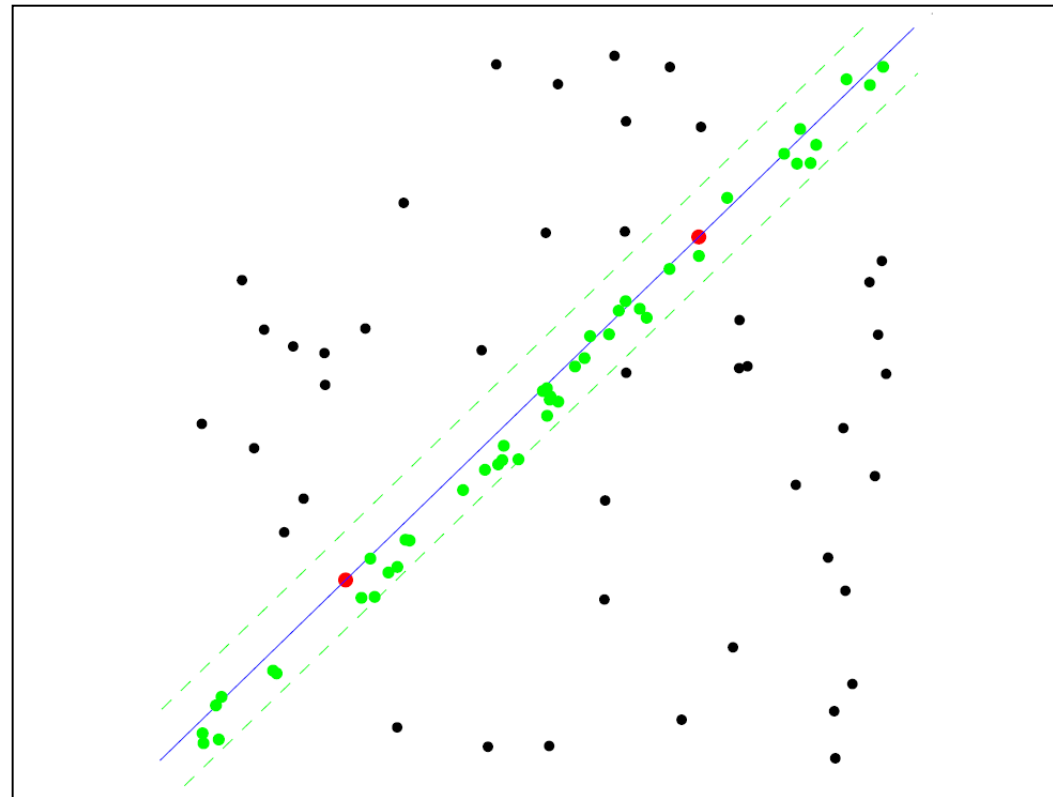
# RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

# RANSAC for line fitting example

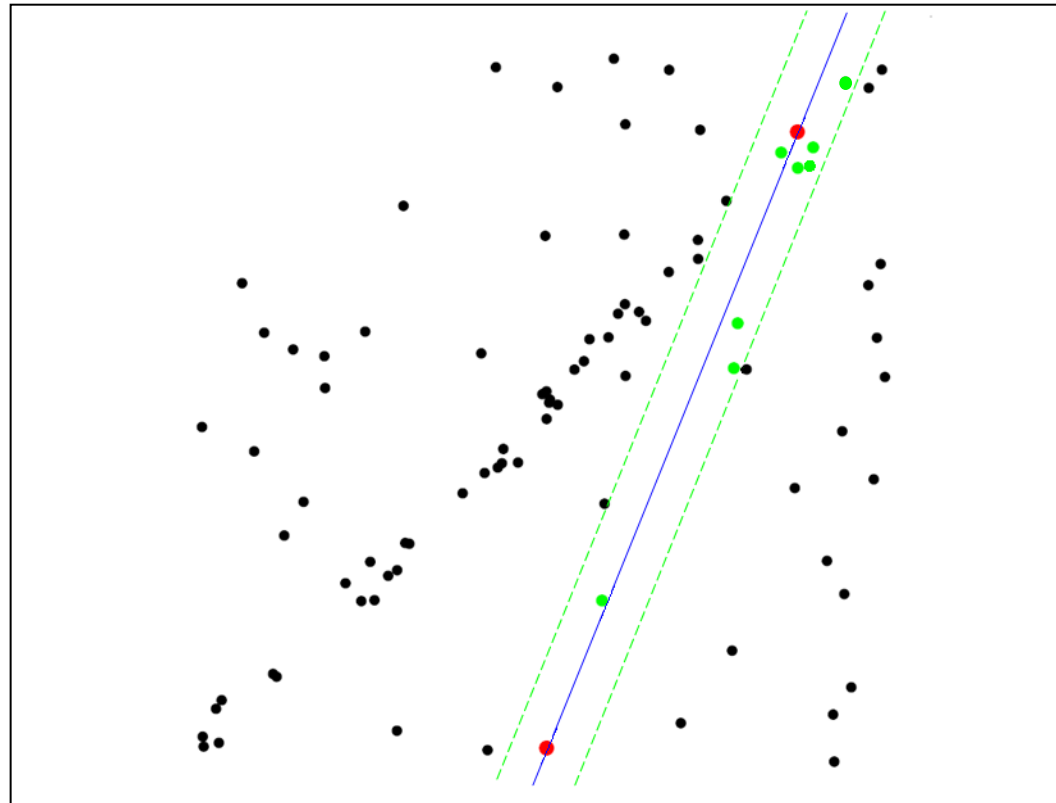
## Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

# RANSAC for line fitting example

<http://www.visual-experiments.com/demo/ransac.js/>



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

# RANSAC: Choosing the parameters

Repeat  $N$  times:

- Draw  $s$  points uniformly at random
- Fit model to these  $s$  points
- Find *inliers* to the model among the remaining points  
(points whose distance or residual w.r.t. model is less than  $t$ )
- If there are  $d$  or more inliers, accept the model and refit using all inliers



# RANSAC: Choosing the parameters

- Initial number of points  $s$ 
  - Typically minimum number needed to fit the model e.g. Line 2 points.
- Distance threshold  $t$  for inliers
  - Need suitable assumptions, e.g., given zero-mean Gaussian noise with std. dev.  $\sigma$ ,  $t = 1.96\sigma$  will give ~95% probability of capturing all inliers
- Consensus set size  $d$ 
  - Should match expected inlier ratio ( $d \approx (1 - e)$  dataset size)
- Number of samples  $N$ 
  - Choose  $N$  so that, with probability  $p$ , at least one random sample (from  $s$ ) is free from outliers (e.g.  $p = 0.99$ )
  - Assuming an outlier ratio of  $e$ :

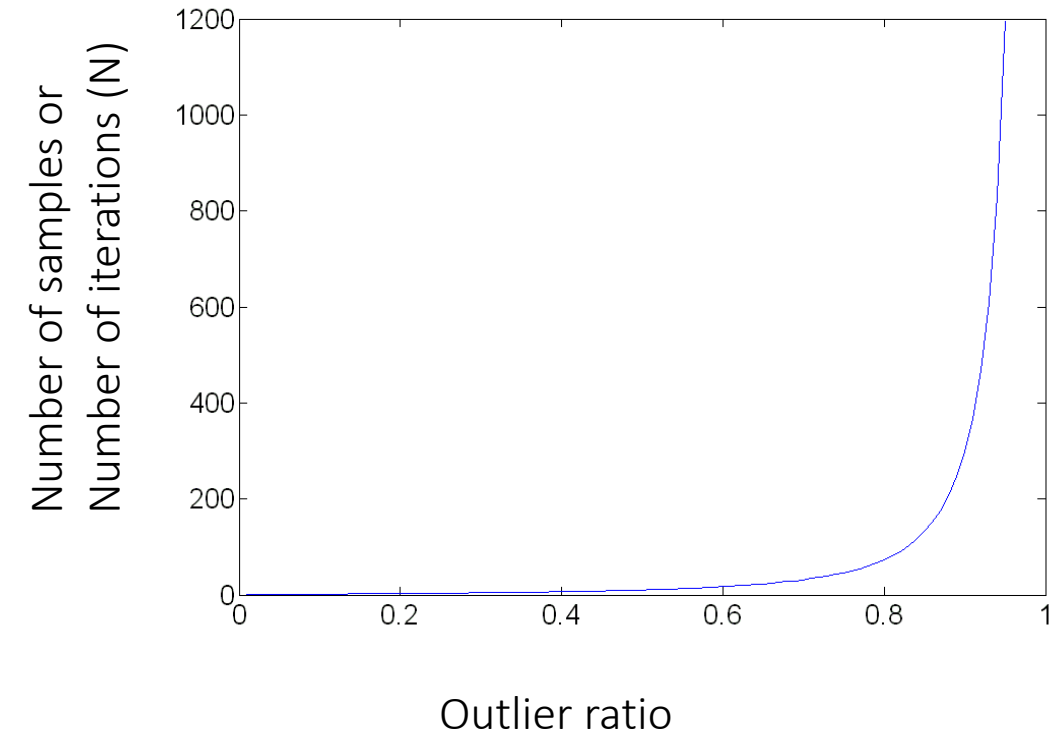
$$(1 - (1 - e)^s)^N = 1 - p$$
$$N = \log(1 - p) / \log(1 - (1 - e)^s)$$

# RANSAC: Choosing the parameters

- Number of samples vs. outlier ratio:

		proportion of outliers $e$							
		s	5%	10%	20%	25%	30%	40%	50%
Initial Number of samples ( $s$ ) 	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	
									
		Number of samples or Number of iterations (N)							

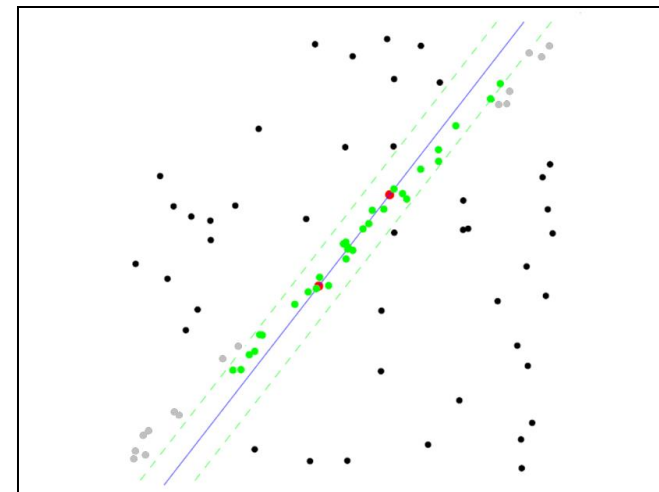
- $N = \log(1 - p) / \log(1 - (1 - e)^s)$
- Set  $p=0.99$  – chance of getting good sample
- e.g.,  $s = 2, e = 5\% \Rightarrow N=2$  ,  $s = 2, e = 50\% \Rightarrow N=17$





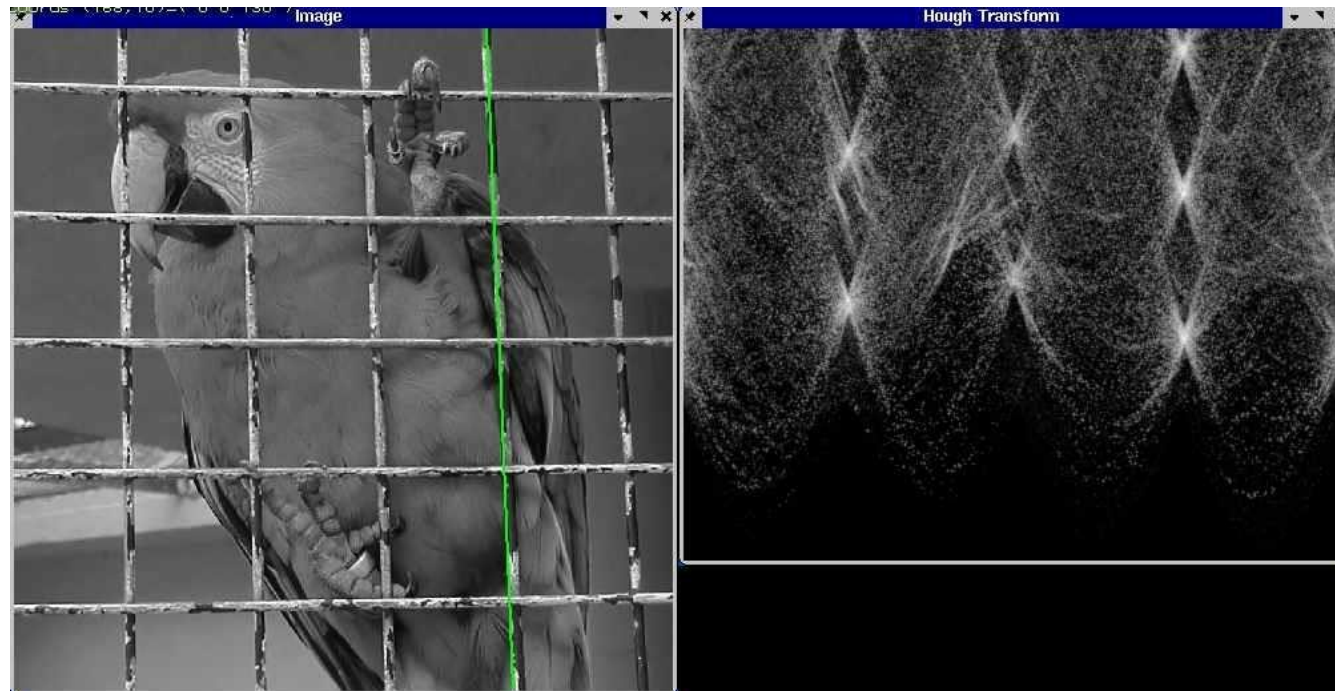
# RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Doesn't work well for low inlier ratios (too many iterations, or can fail completely)
  - Can't always get a good initialization of the model based on the minimum number of samples



# Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC
- Hough transform



# Hough transform

- Possibly the earliest voting scheme – but still useful!
  - Discretize parameter space into bins
  - For each feature point in the image, put a vote in every bin in the parameter space that could have generated this point
  - Find bins that have the most votes

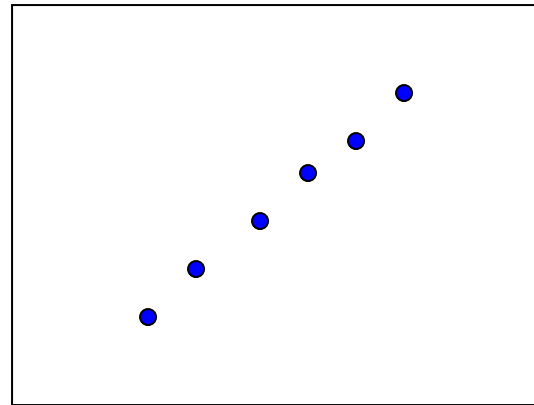
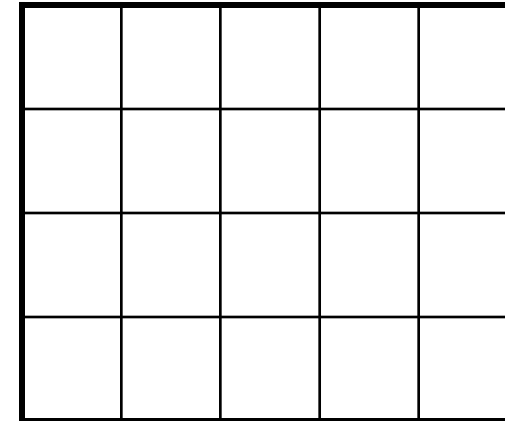
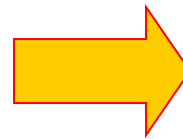


Image space



Hough parameter space

# Hough transform

- What does a **line** in the image space correspond to?
  - A **point** in the parameter space

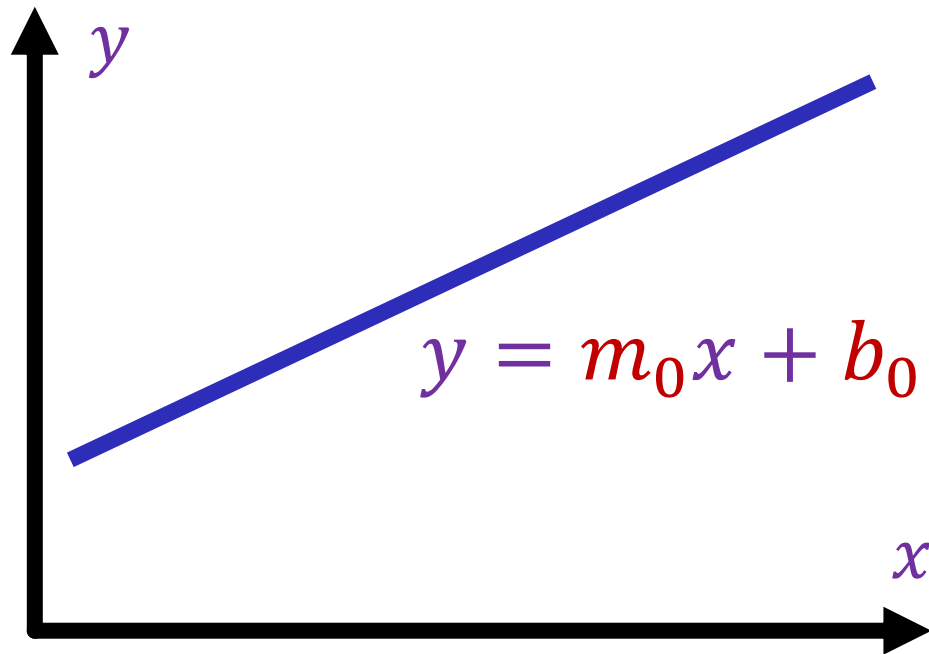
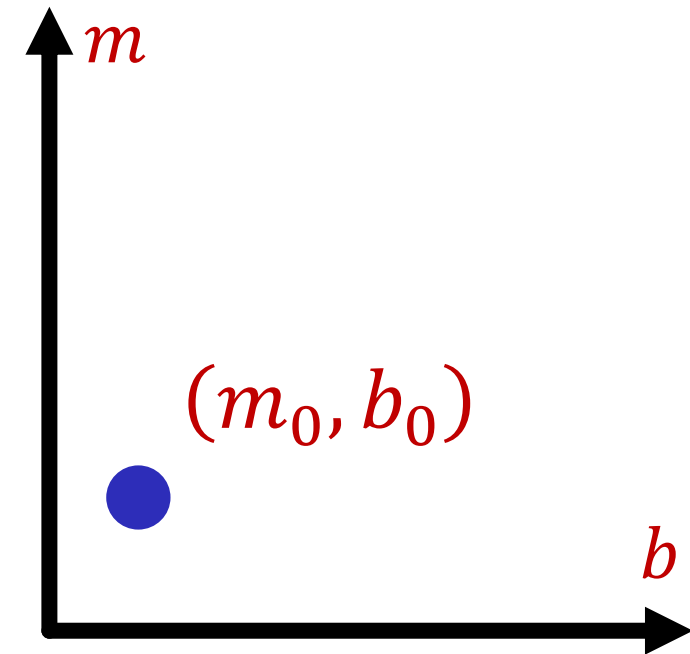


Image space



Parameter space

# Hough transform

- What does a **point** in the image space correspond to?
  - A **line** in the parameter space: all  $(m, b)$  that satisfy  $-b = x_0 m - y_0$

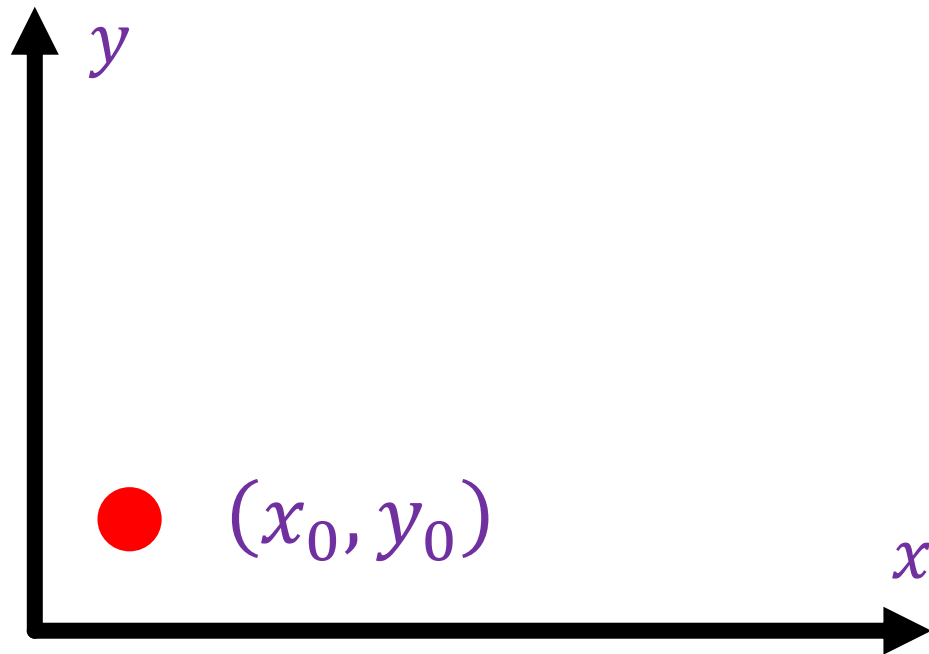
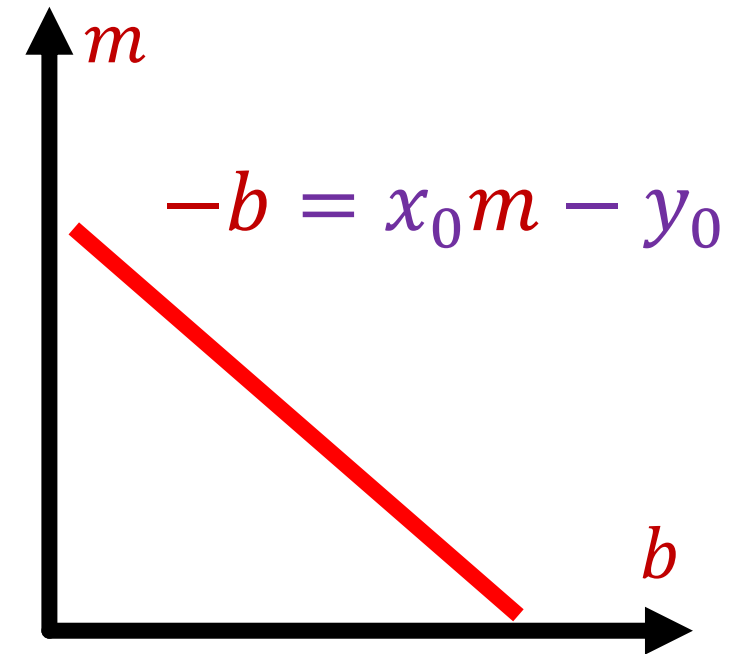


Image space



Parameter space

# Hough transform

- What about **two points** in the image space?
  - A **point** in the parameter space, corresponding to the unique line that passes through both points

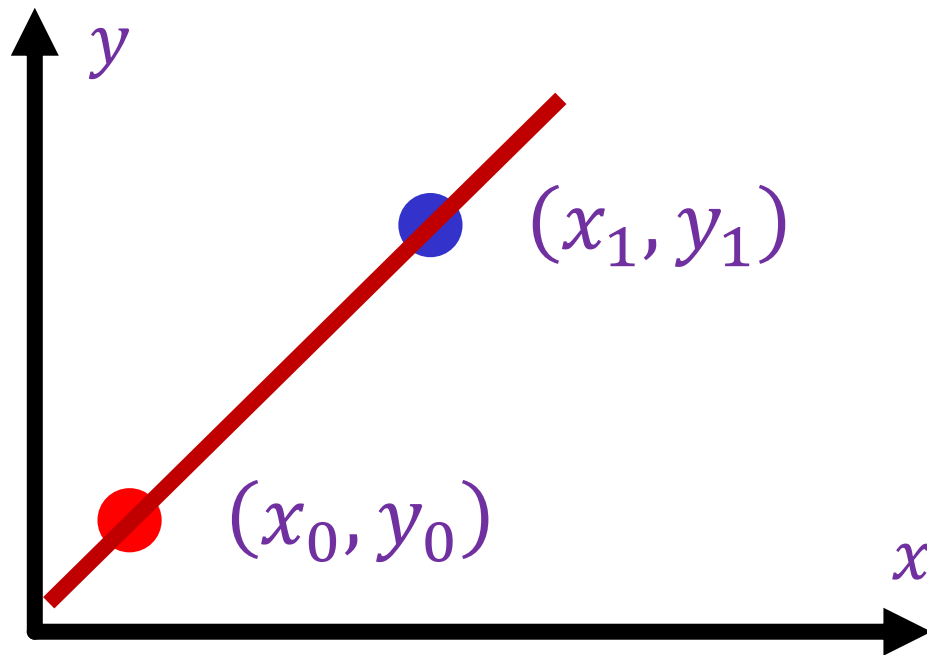
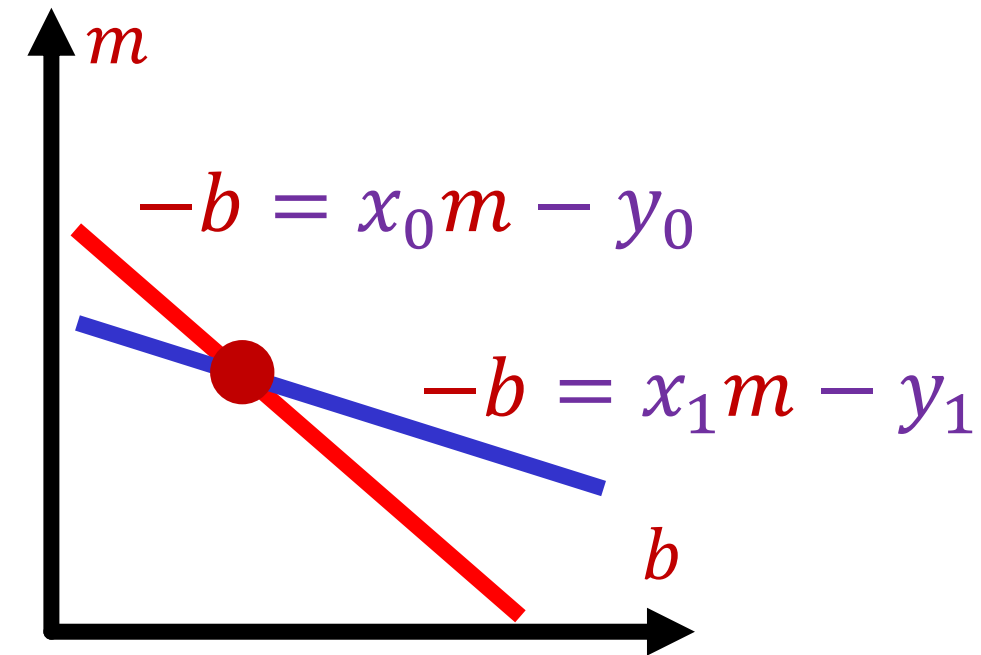


Image space



Parameter space

# Hough transform

- What about **many points** in the image space?
  - Plot all the lines in the parameter space and try to find a spot where a large number of them intersect

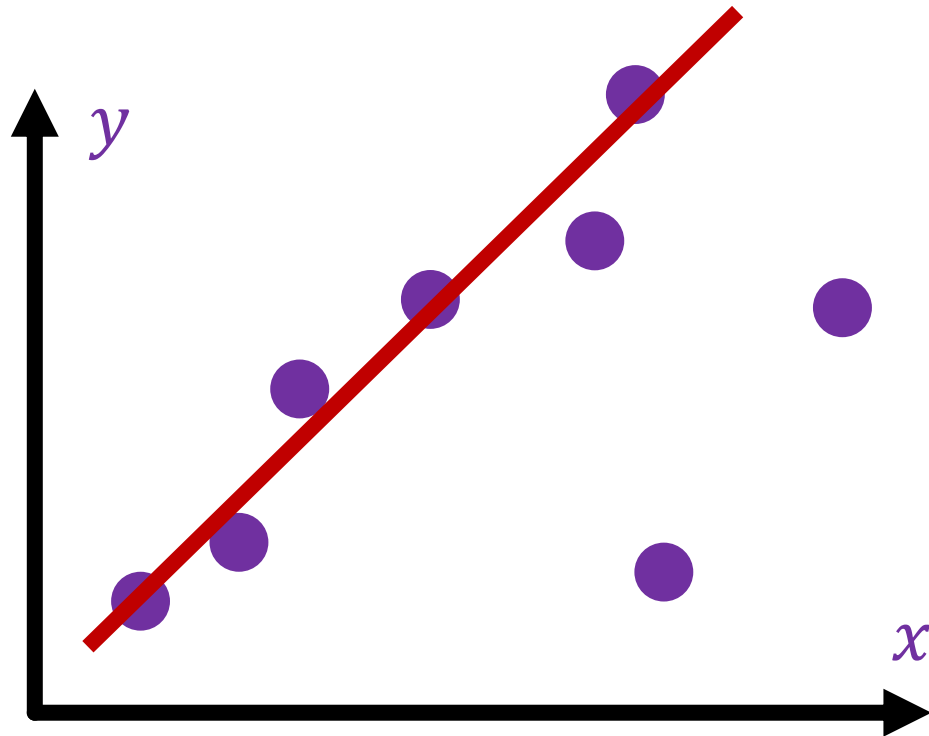
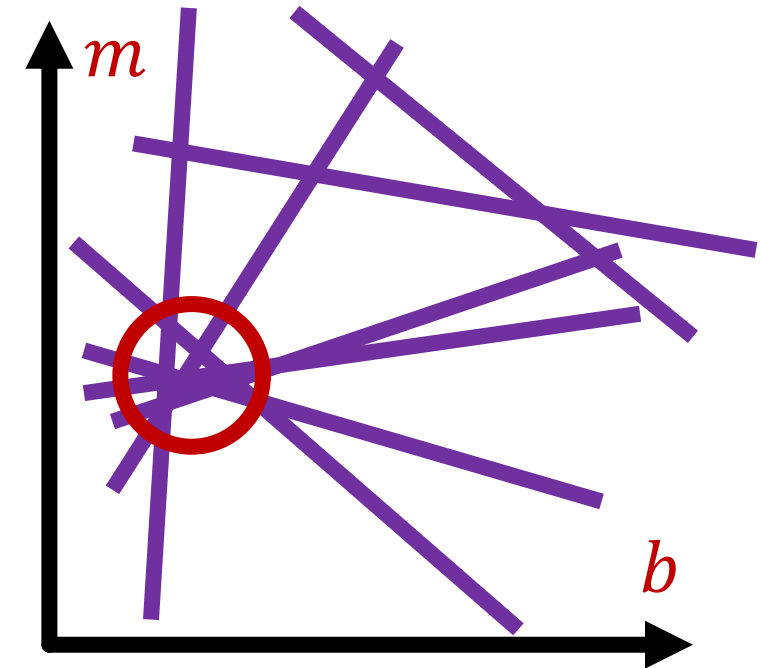


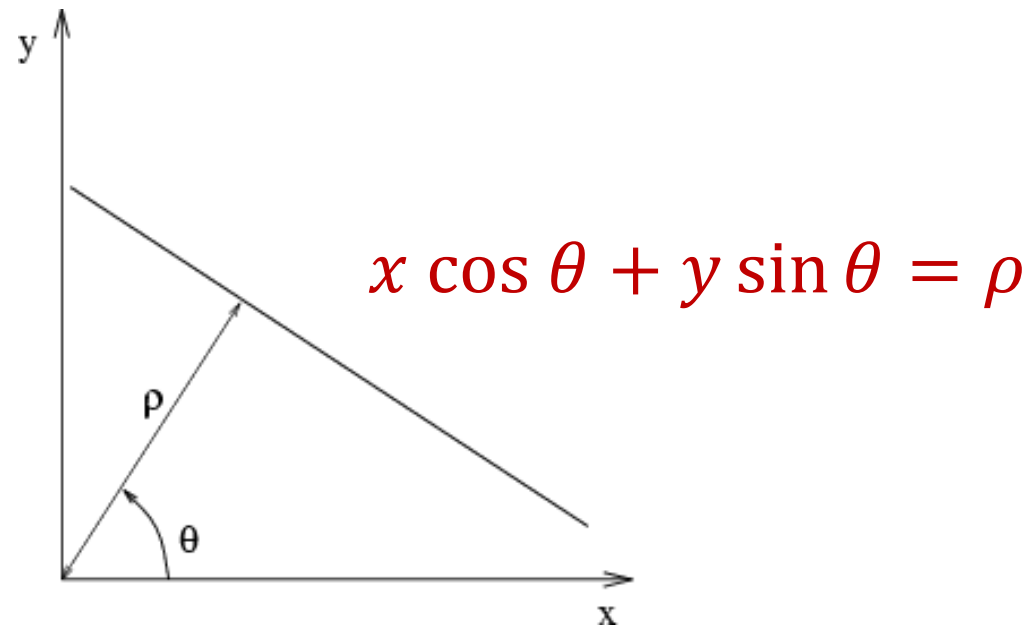
Image space



Parameter space

# Parameter space representation

- In practice, we don't want to use the  $(m, b)$  space!
  - Unbounded parameter domains
  - Vertical lines require infinite  $m$
- Alternative: *polar representation*
  - Each image point  $(x, y)$  yields a *sinusoid* in the  $(\theta, \rho)$  parameter space





# Algorithm outline

- Initialize accumulator  $H$  to all zeros

- For each feature point  $(x, y)$

For  $\theta = 0$  to  $180$

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

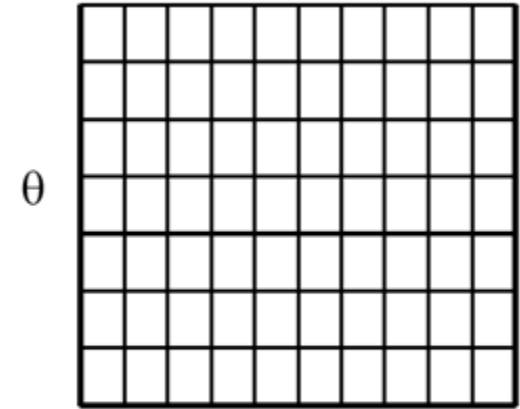
$$H(\theta, \rho) += 1$$

- Find the value(s) of  $(\theta, \rho)$  where  $H(\theta, \rho)$  is a local maximum (perform NMS on the accumulator array)

- The detected line in the image is given by

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

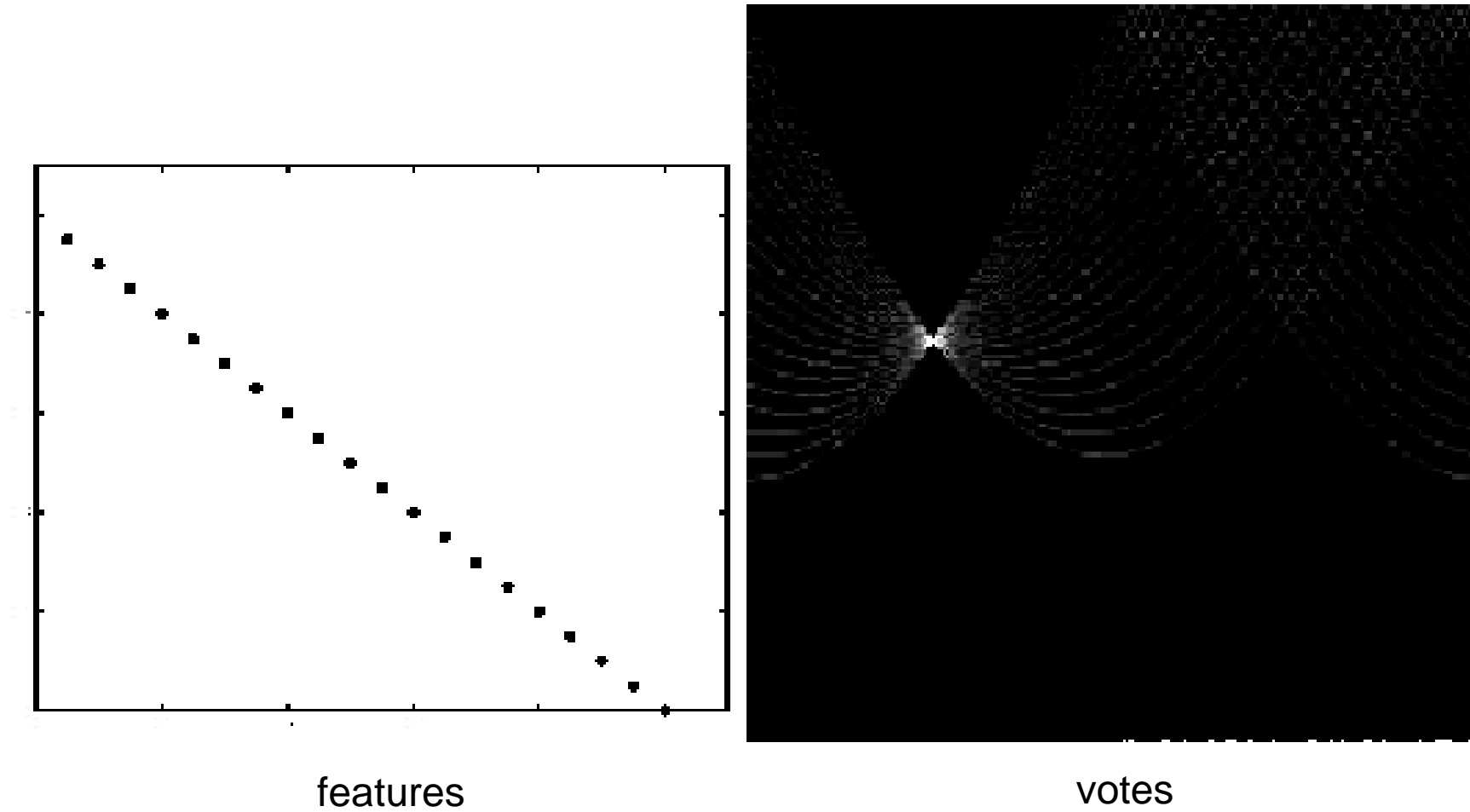
H: accumulator array (votes)



<https://www.youtube.com/watch?v=4zHbl-fFill>

[https://www.youtube.com/watch?v=XRBC\\_xkZREg](https://www.youtube.com/watch?v=XRBC_xkZREg)

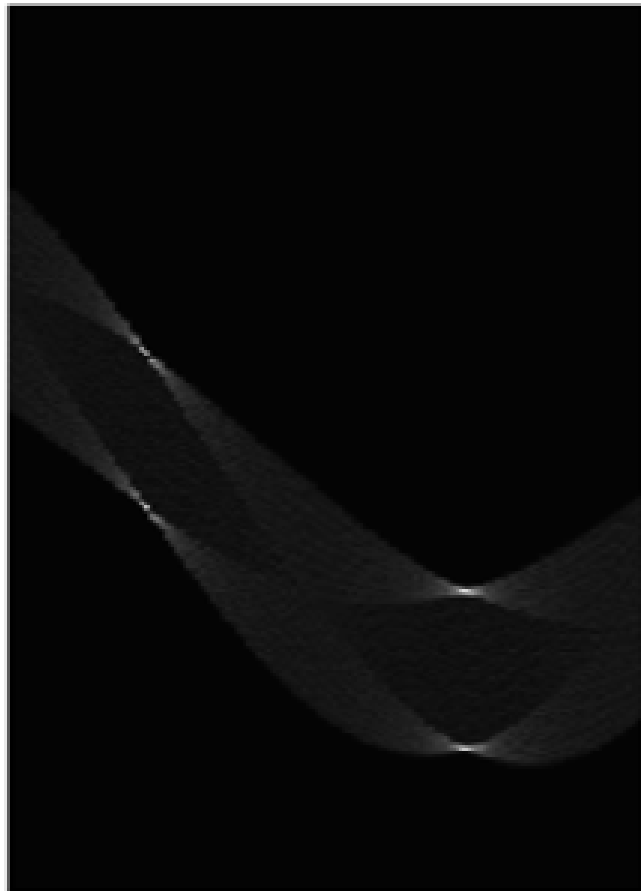
# Basic illustration



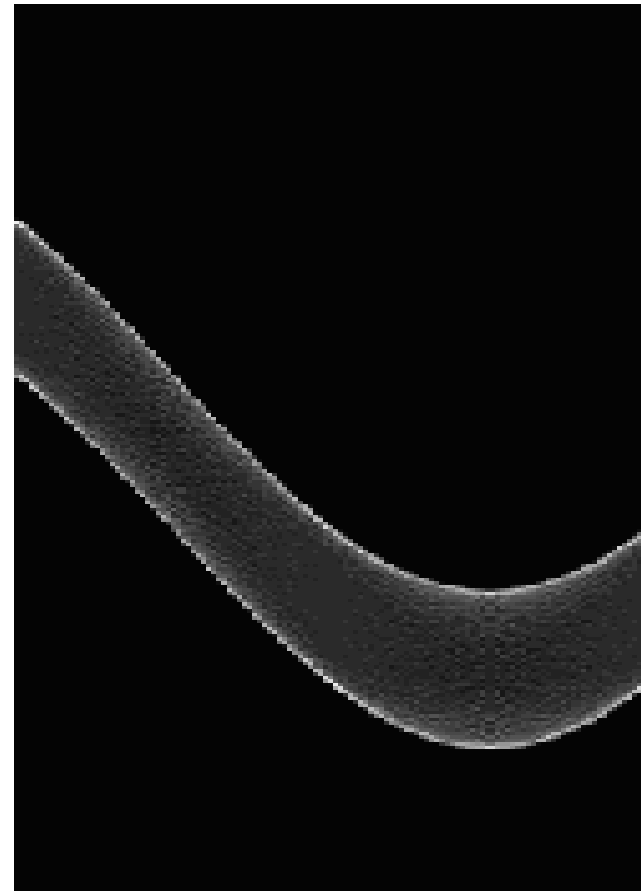
[Hough transform demo](#)

# Other shapes

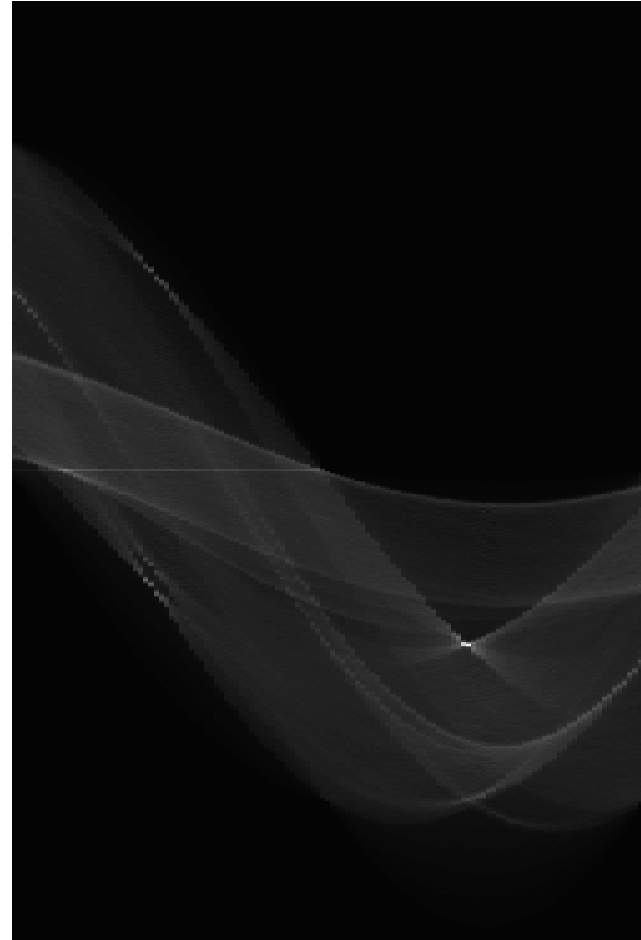
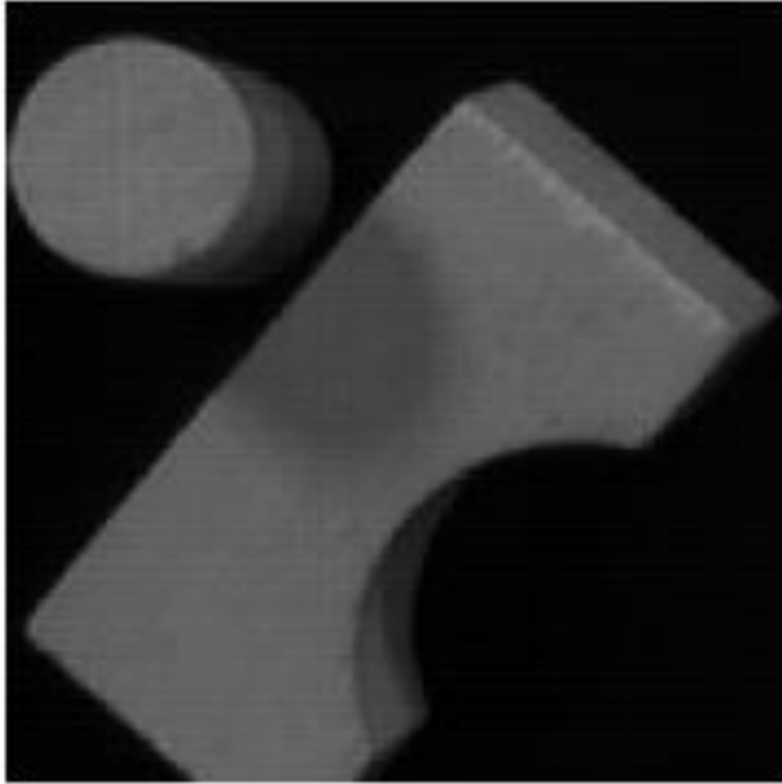
Square



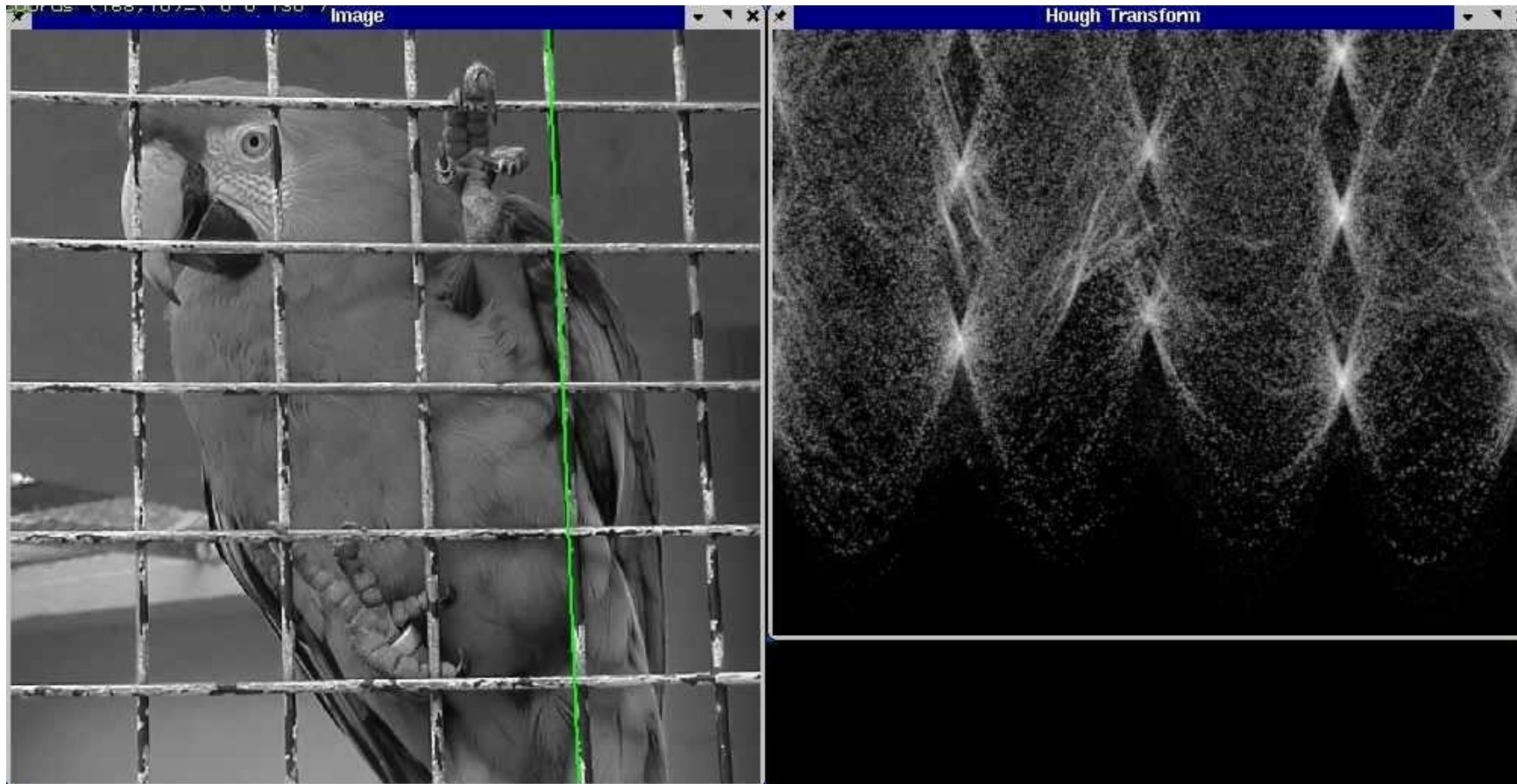
Circle



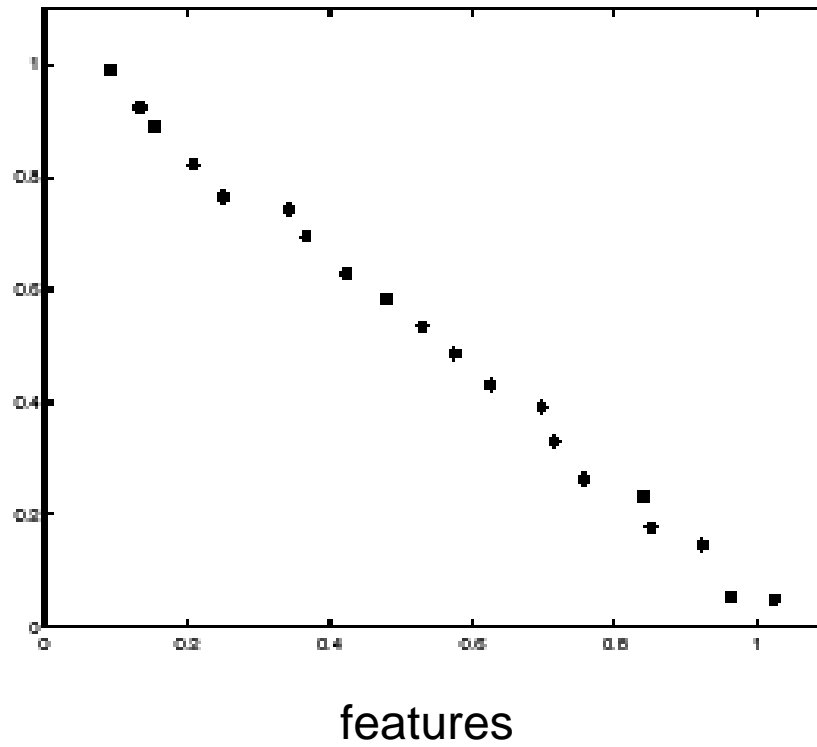
Several lines



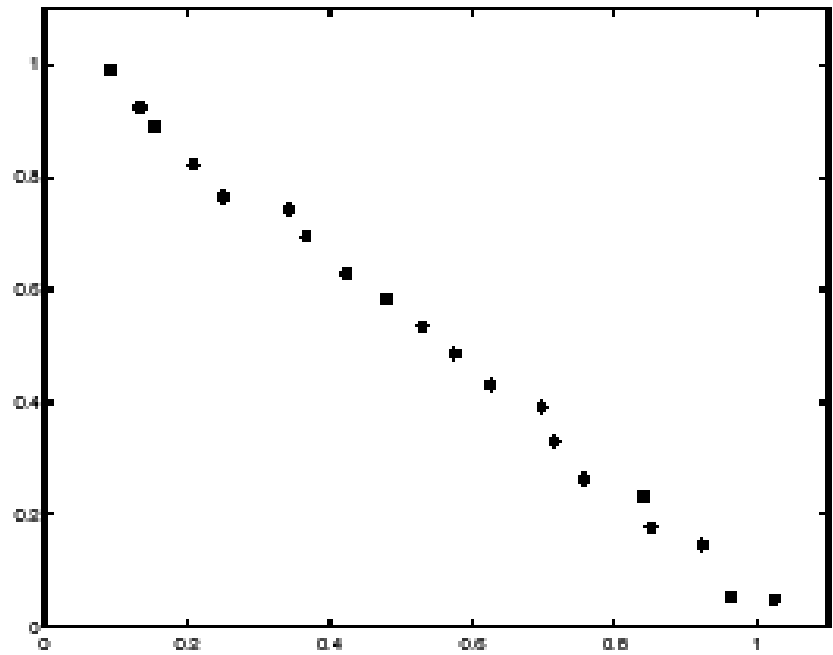
# A more complicated image



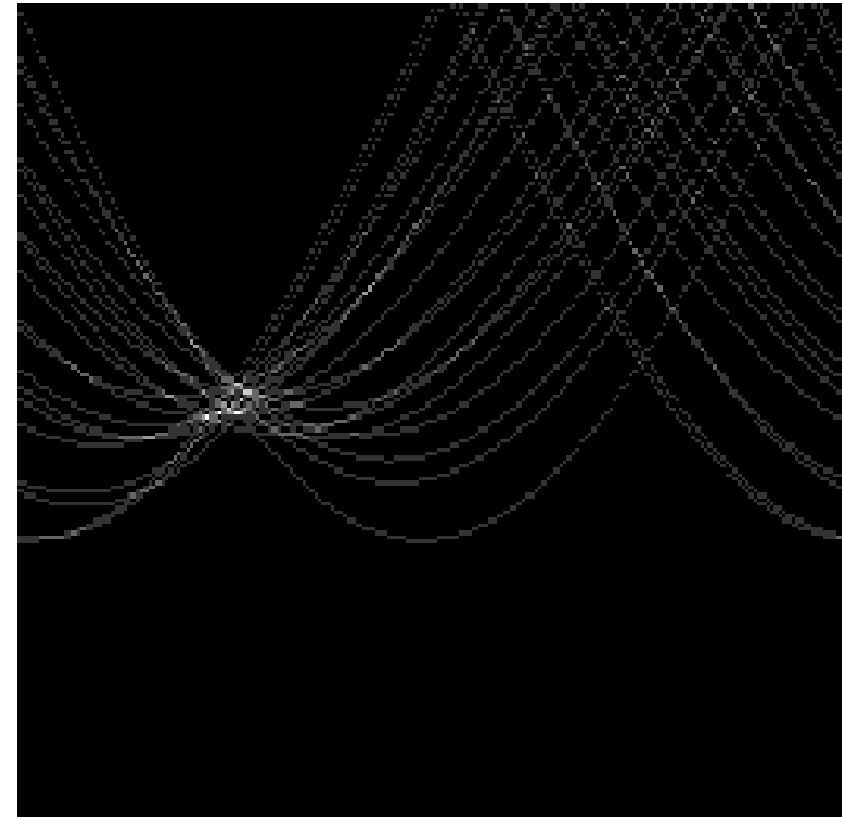
# Effect of noise



# Effect of noise



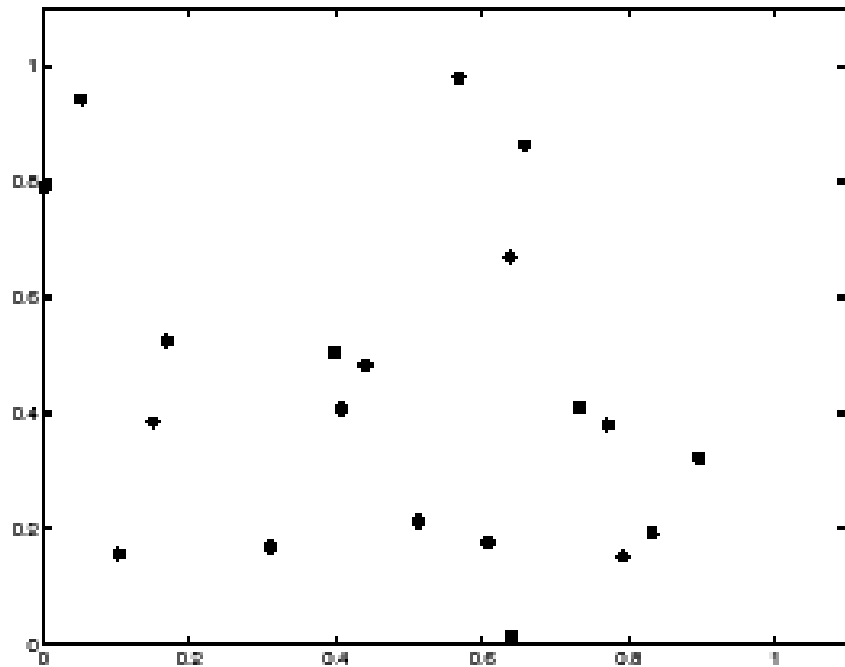
features



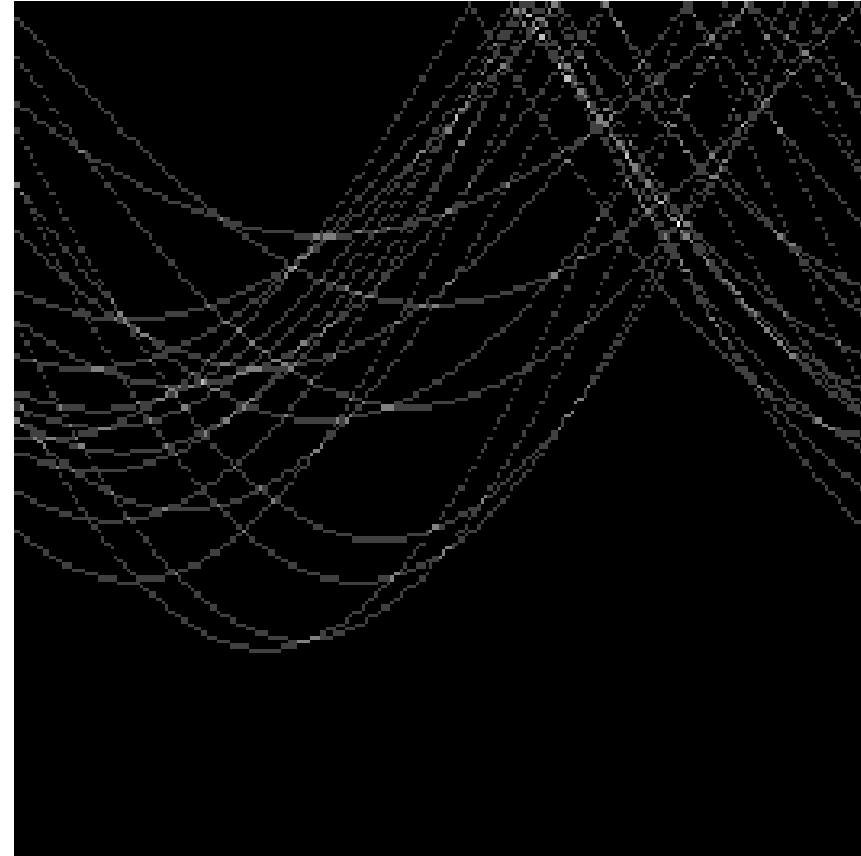
votes

Peak gets fuzzy and hard to locate

# Effect of outliers



features



votes

Uniform noise can lead to spurious peaks in the array



# Dealing with noise

- How to 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
- Increment neighboring bins (smoothing in accumulator array)
- Try to get rid of irrelevant features
  - E.g., take only edge points with significant gradient magnitude

# Hough transform: Pros and cons

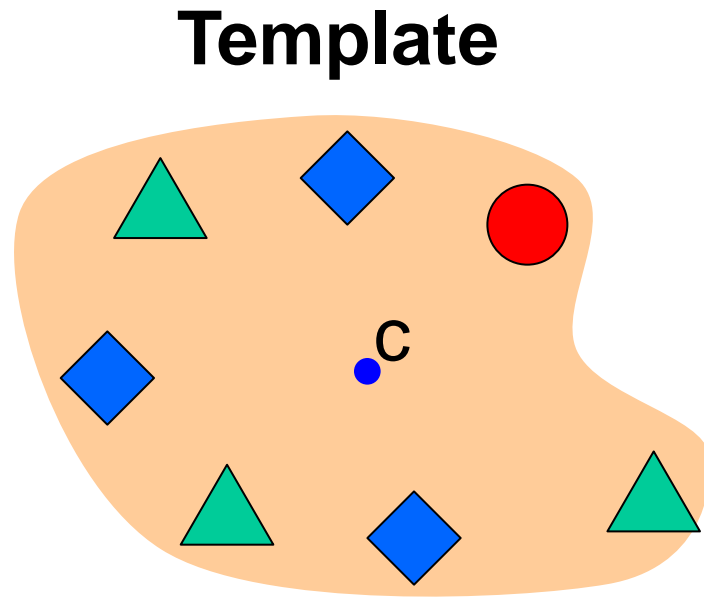
- Pros
  - Can deal with non-locality and occlusion
  - Can detect multiple instances of a model
  - Some robustness to noise: noise points unlikely to contribute consistently to any single bin
  - Leads to a surprisingly general strategy for object localization (more on this next)
- Cons
  - Complexity increases exponentially with the number of model parameters – in practice, not used beyond three or four dimensions
  - Non-target shapes can produce spurious peaks in parameter space
  - It's hard to pick a good grid size

# Fitting: Overview

- Least squares line fitting
- Robust fitting
- RANSAC
- Hough transform
- Generalized Hough transform

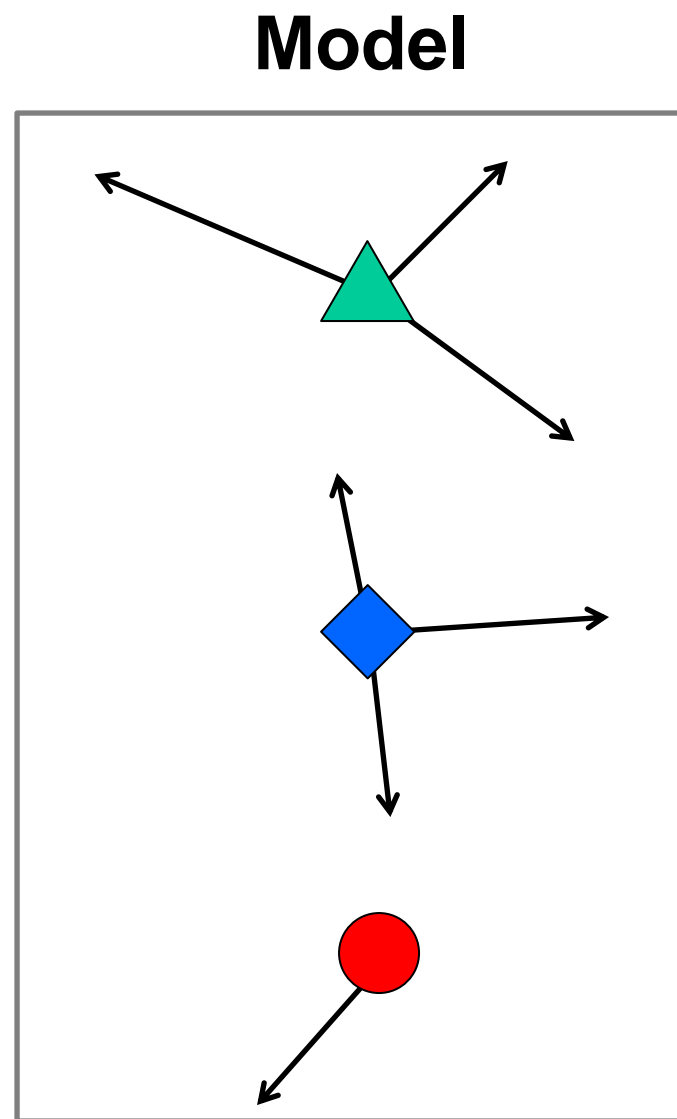
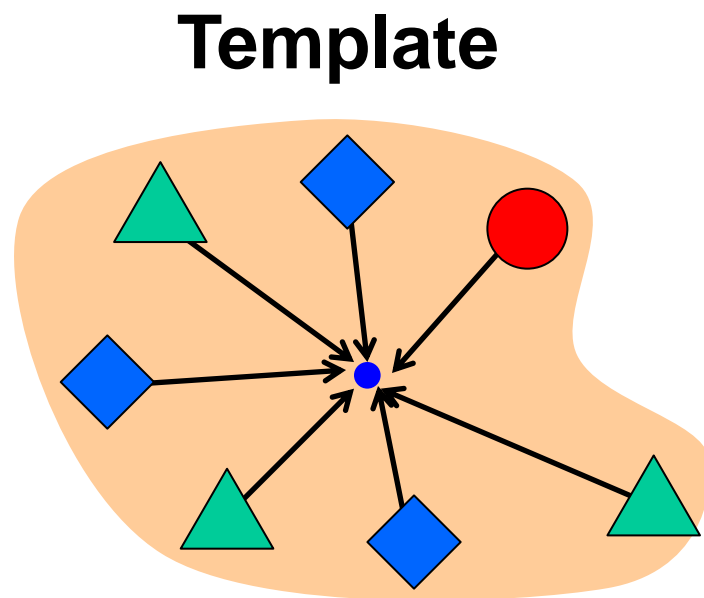
# Generalized Hough transform

- We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration



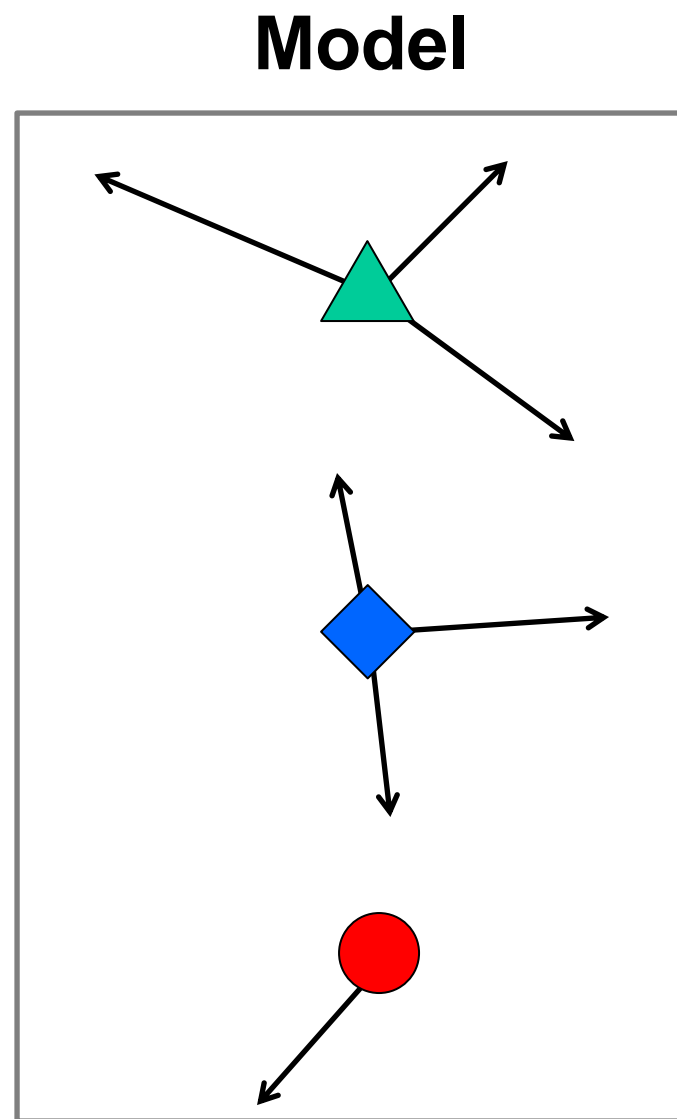
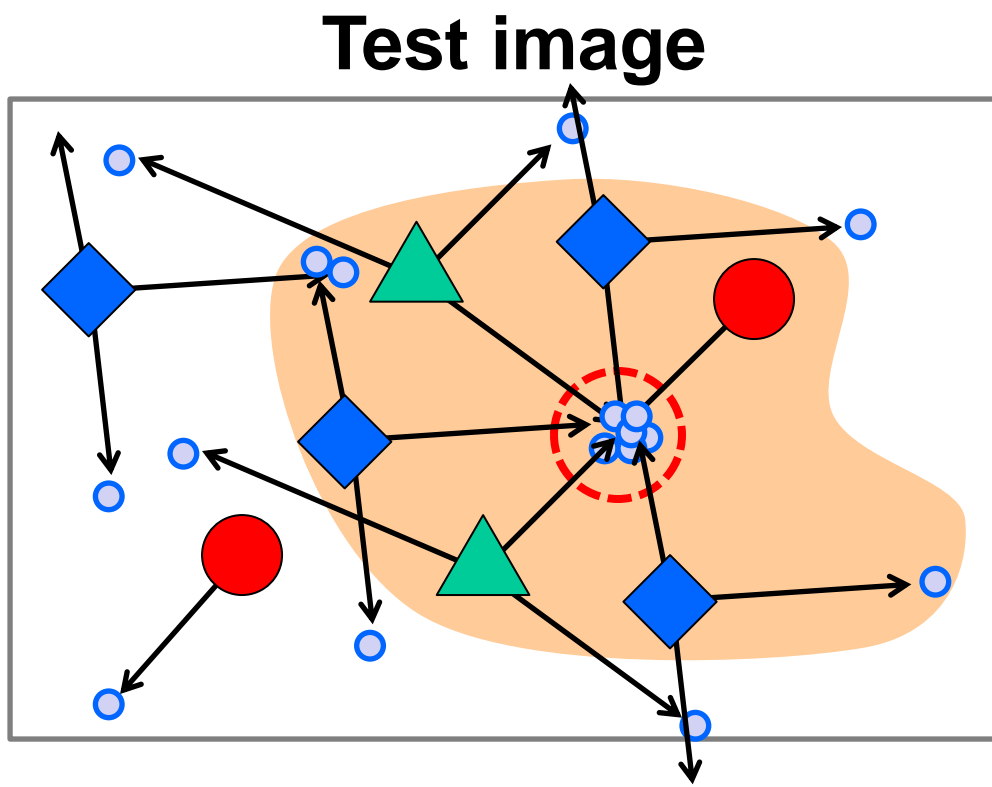
# Generalized Hough transform

- Template representation: for each type of landmark point, store all possible displacement vectors towards the center



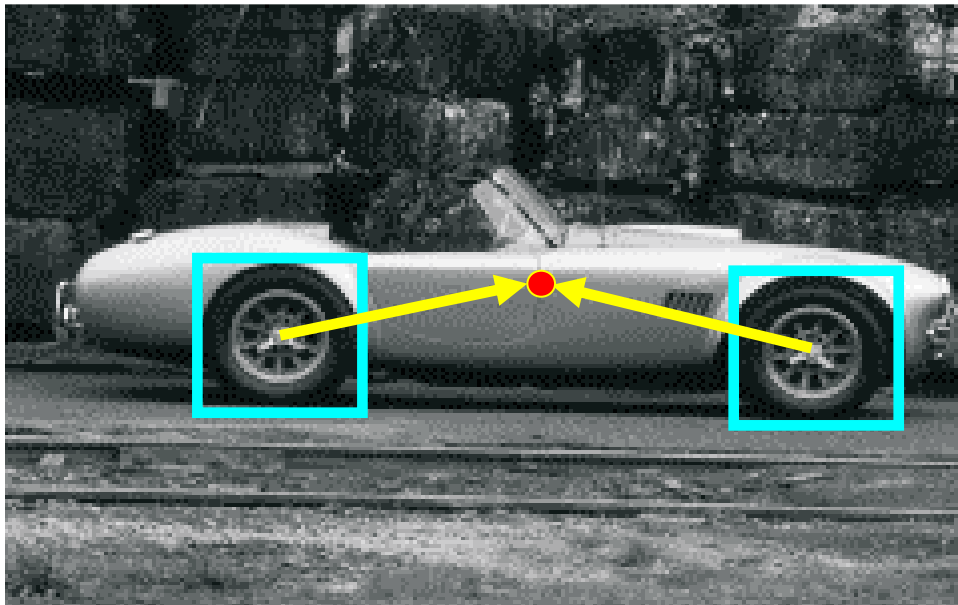
# Generalized Hough transform

- Detecting the template:
  - For each feature in a new image, look up that feature type in the model and vote for the possible center locations associated with that type in the model

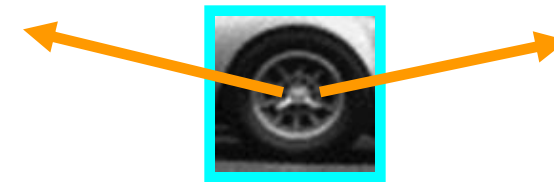


# Application in recognition

- Index displacements by “visual codeword”



training image



visual codeword with  
displacement vectors

B. Leibe, A. Leonardis, and B. Schiele, [Combined Object Categorization and Segmentation with an Implicit Shape Model](#), ECCV Workshop on Statistical Learning in Computer Vision 2004

# Application in recognition

- Index displacements by “visual codeword”



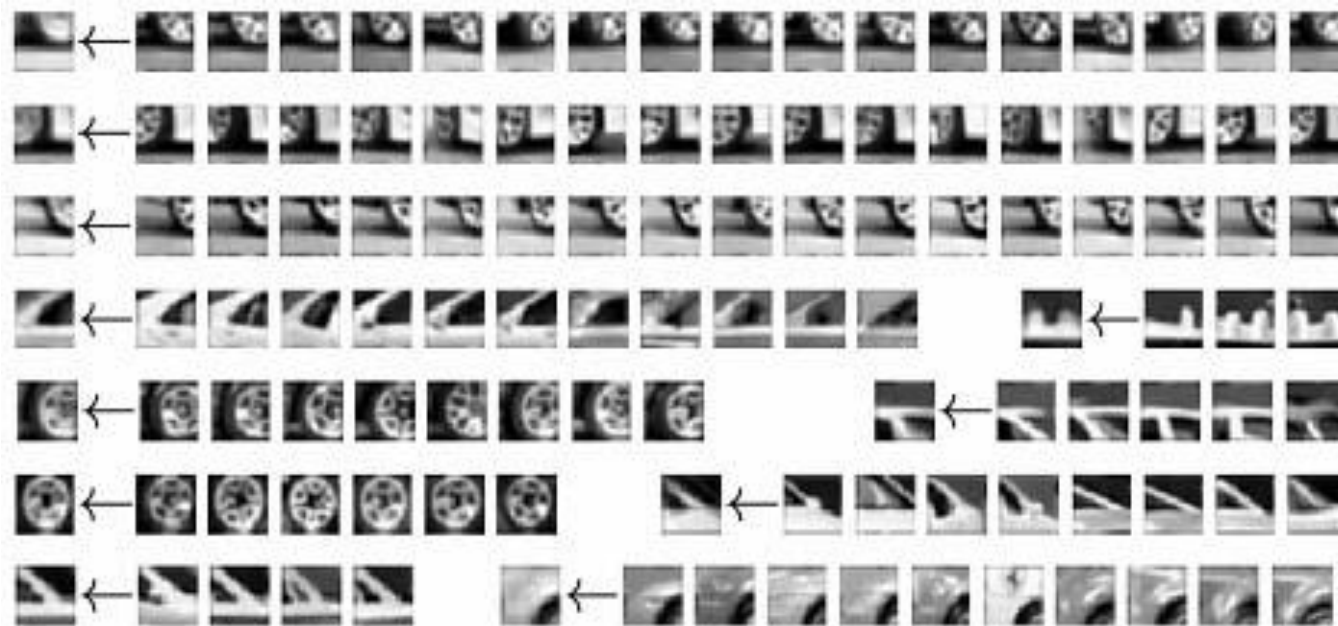
test image

B. Leibe, A. Leonardis, and B. Schiele, [Combined Object Categorization and Segmentation with an Implicit Shape Model](#), ECCV Workshop on Statistical Learning in Computer Vision 2004



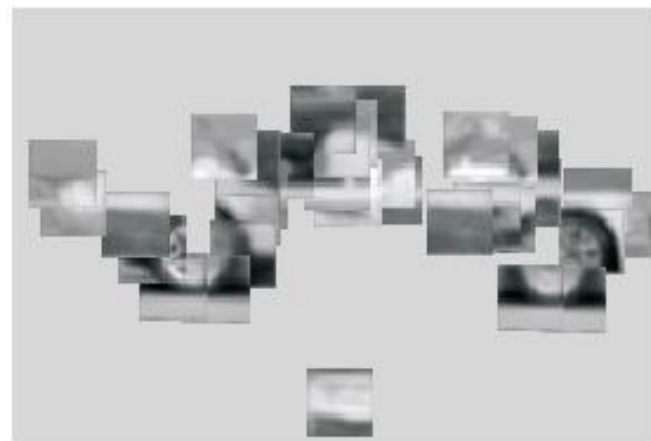
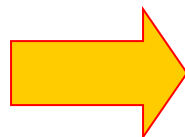
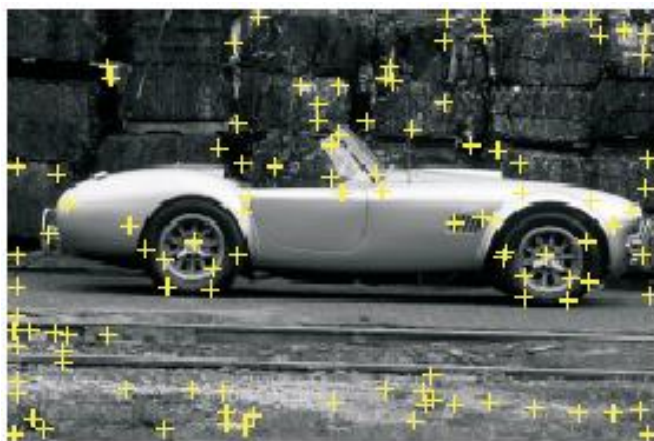
# Implicit shape models: Training

1. Build *codebook* of patches around extracted interest points using clustering (more on this later in the course)



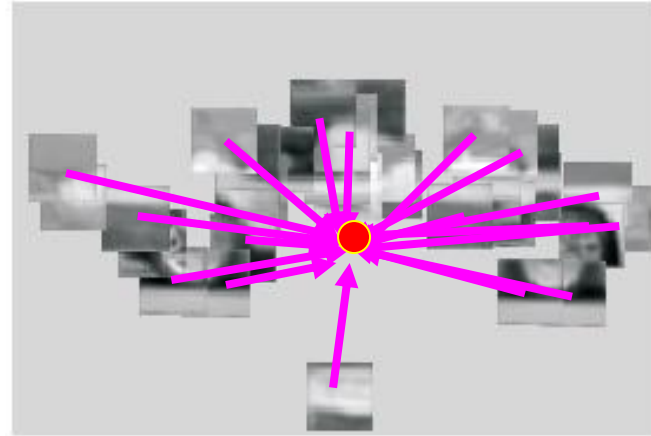
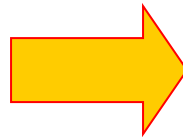
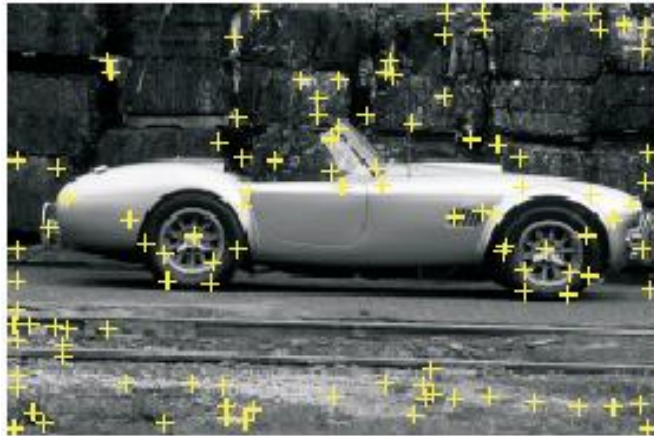
# Implicit shape models: Training

1. Build *codebook* of patches around extracted interest points using clustering
2. Map the patch around each interest point to closest codebook entry



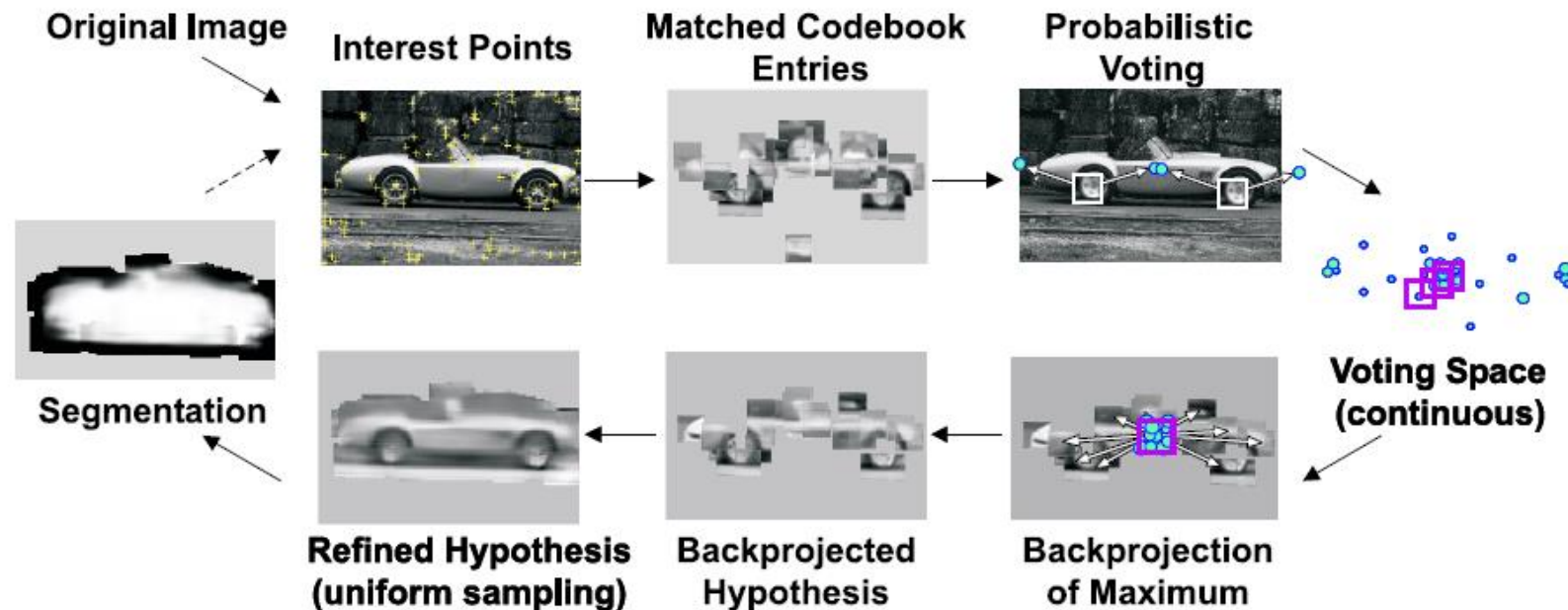
# Implicit shape models: Training

1. Build *codebook* of patches around extracted interest points using clustering
2. Map the patch around each interest point to closest codebook entry
3. For each codebook entry, store all positions it was found, relative to object center

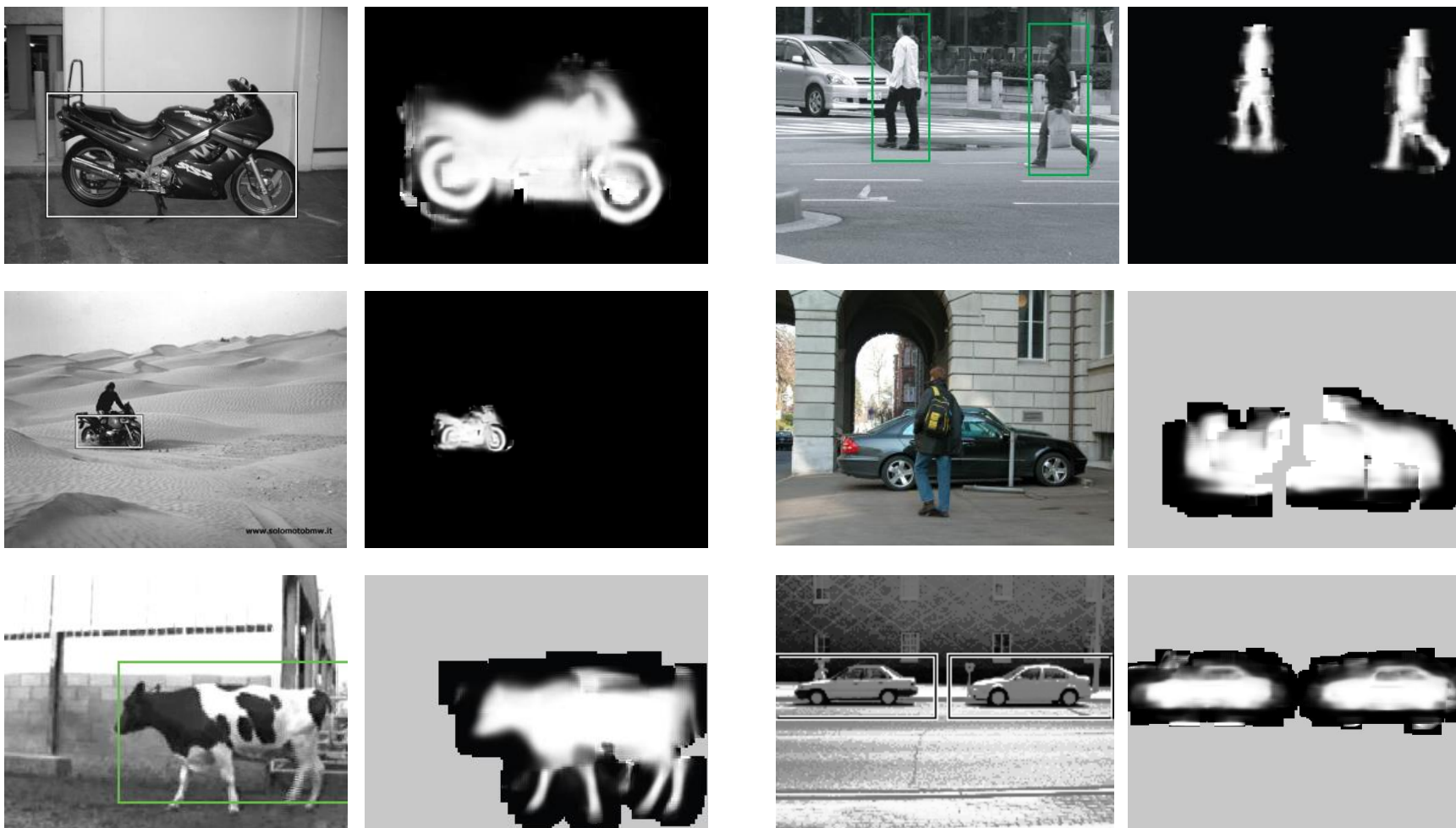


# Implicit shape models: Testing

1. Given test image, extract patches, match to codebook entry
2. Cast votes for possible positions of object center
3. Search for maxima in voting space
4. Extract weighted segmentation mask based on stored masks for the codebook occurrences

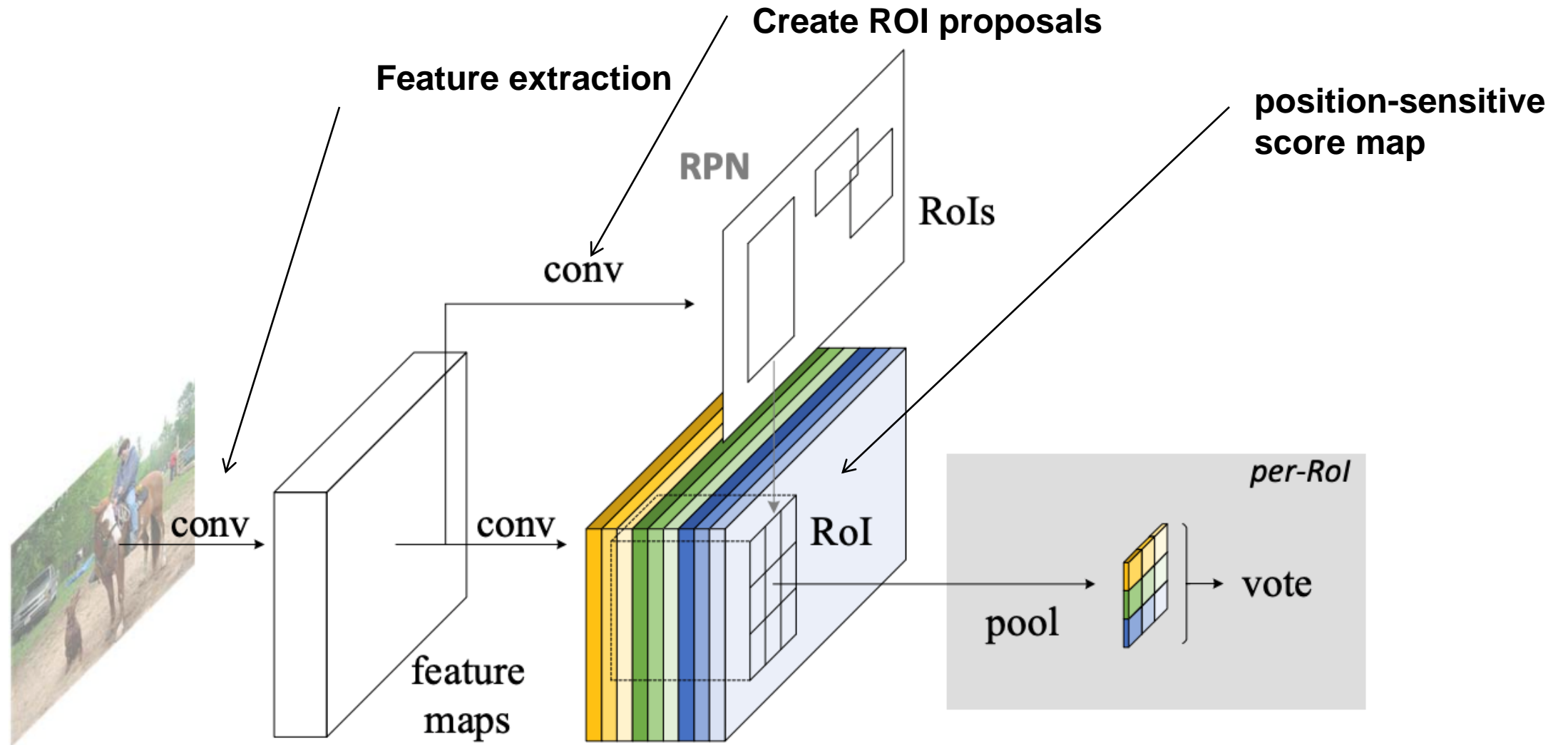


# Additional examples



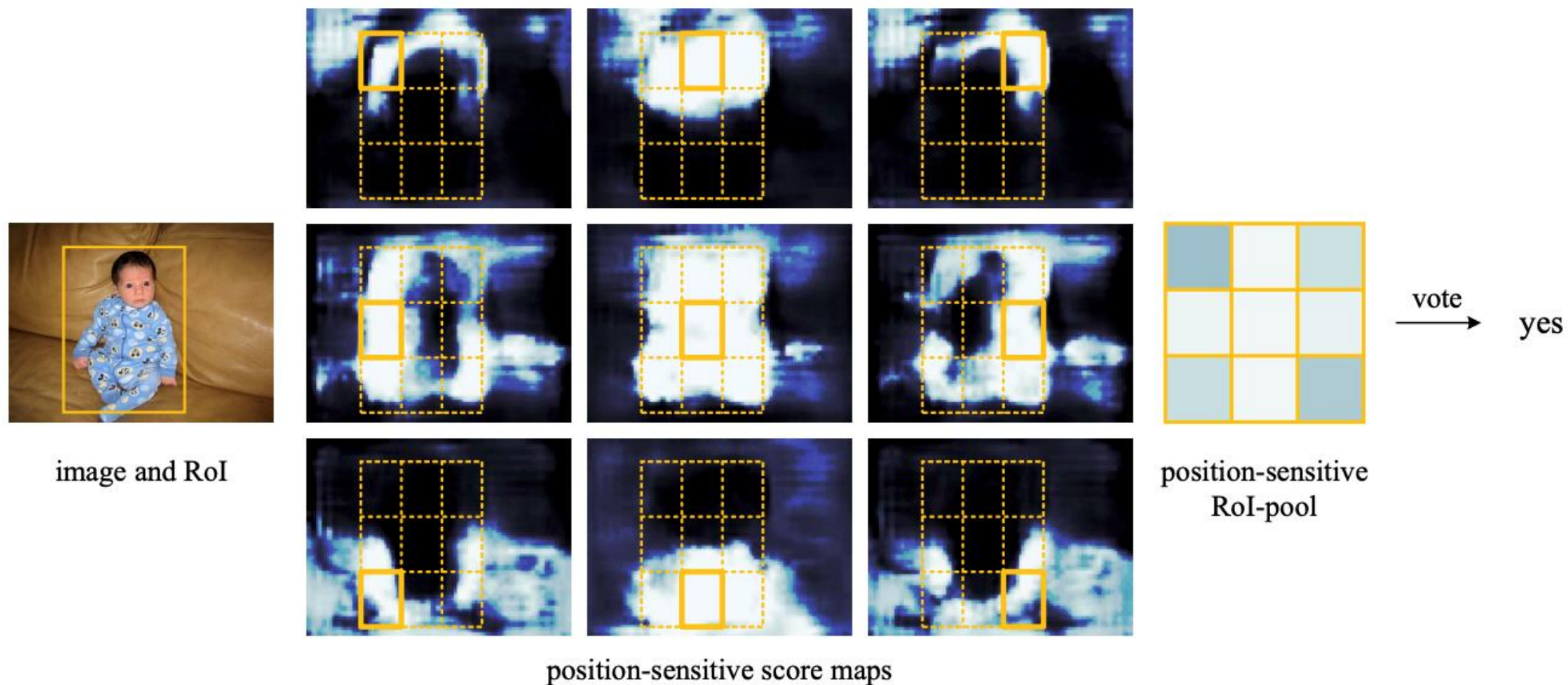
B. Leibe, A. Leonardis, and B. Schiele, [Robust Object Detection with Interleaved Categorization and Segmentation](#), IJCV 77 (1-3), pp. 259-289, 2008.

# A more recent example: Voting for detection

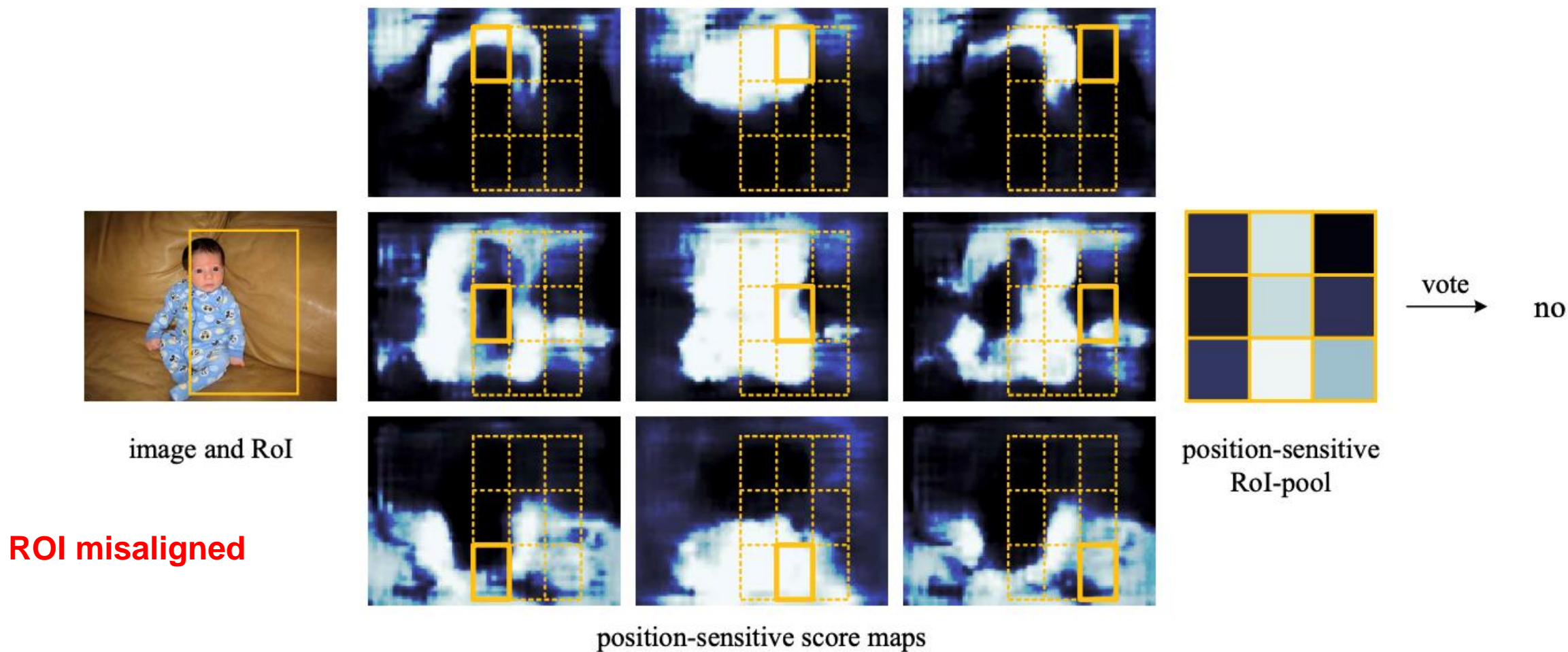




# A more recent example: Voting for detection



# A more recent example: Voting for detection





# Convolutional Hough matching networks

