Computer Vision

CS-E4850, 5 study credits

Lecturer: Juho Kannala

Lecture 4: Model estimation (fitting)

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

These topics are covered in Szeliski's book briefly, but more thoroughly in Chapter 17 of Forsyth & Ponce:

http://courses.cs.washington.edu/courses/cse455/02wi/readings/book-7-revised-a-indx.pdf

Acknowledgement: many slides from Svetlana Lazebnik, Derek Hoiem, Kristen Grauman, David Forsyth, Marc Pollefeys, and others (detailed credits on individual slides)

Relevant reading

- These topics are covered in Szeliski's book briefly, but more thoroughly in the following books:
  - Chapter 17 of Forsyth & Ponce:
    http://cmuems.com/excap/readings/forsyth-ponce-computer-vision-a-modern-approach.pdf
  - Chapter 4 of Hartley & Zisserman:
    http://cvrs.whu.edu.cn/downloads/ebooks/Multiple%20View%20Geometry%20in%20Computer%20Vision%20(Second%20Edition).pdf

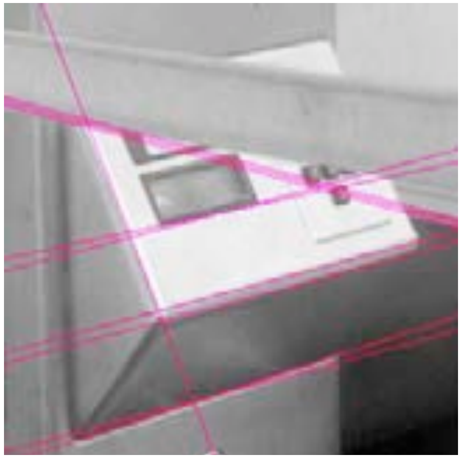
Source: S. Lazebnik

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

Source: K. Grauman

Source: S. Lazebnik

Fitting: Issues

Case study: Line detection

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

Source: S. Lazebnik

Fitting: Overview

- If we know which points belong to the line, how do we find the “optimal” line parameters?
  - Least squares
- What if there are outliers?
  - Robust fitting, RANSAC
- What if there are many lines?
  - Voting methods: RANSAC, Hough transform
- What if we’re not even sure it’s a line?
  - Model selection (not covered)

Source: S. Lazebnik

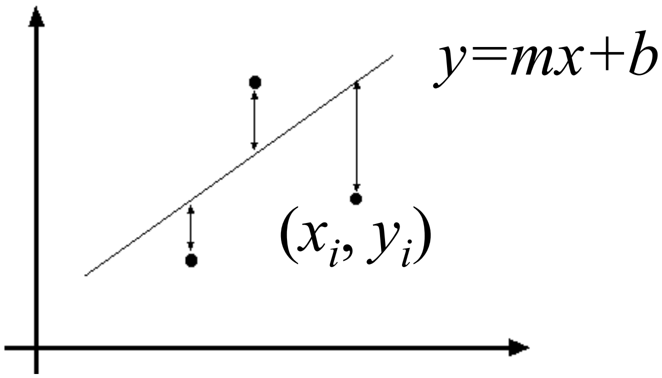
Least squares line fitting

Data: $(x_1, y_1), \ldots, (x_n, y_n)$

Line equation: $y_i = m x_i + b$

Find (m, b) to minimize

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



Source: S. Lazebnik

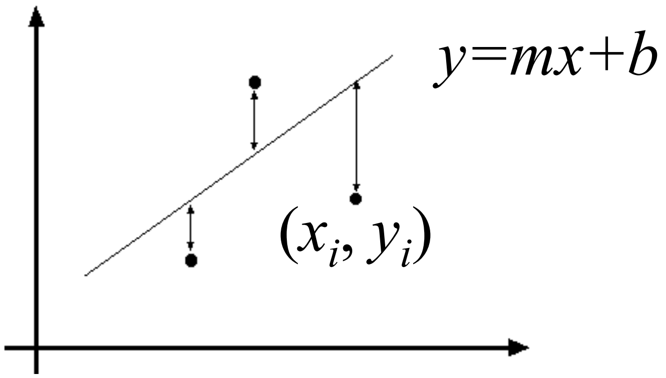
Least squares line fitting

Data: $(x_1, y_1), \ldots, (x_n, y_n)$

Line equation: $y_i = m x_i + b$

Find (m, b) to minimize

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



$$E = \|Y - XB\|^2 \quad \text{where} \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad X = \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \qquad B = \begin{bmatrix} m \\ b \end{bmatrix}$$

$$E = \|Y - XB\|^2 = (Y - XB)^T (Y - XB) = Y^T Y - 2(XB)^T Y + (XB)^T (XB)$$

$$\frac{dE}{dB} = 2X^T XB - 2X^T Y = 0$$

$$X^T XB = X^T Y$$

Normal equations: least squares solution to $XB = Y$

Source: S. Lazebnik

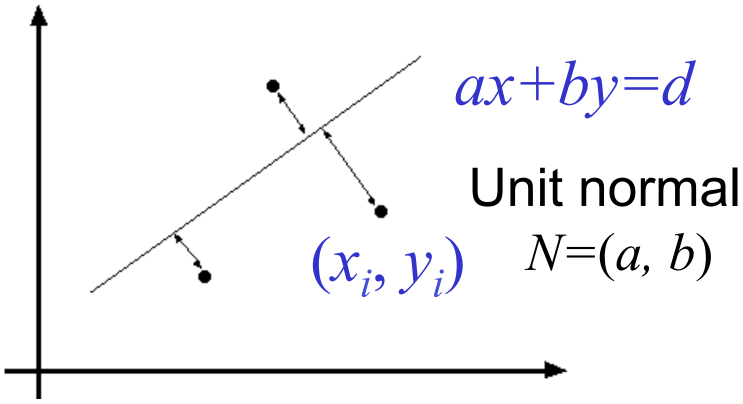
Problem with “vertical” least squares

- Not rotation-invariant
- Fails completely for vertical lines

Source: S. Lazebnik

Total least squares

Distance between point (x_i, y_i) and line $ax+by=d$ $(a^2+b^2=1)$: $|ax_i + by_i - d|$



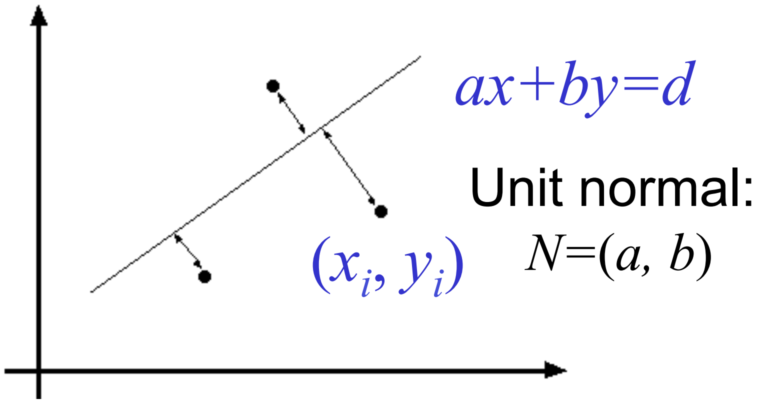
Source: S. Lazebnik

Total least squares

Distance between point (x_i, y_i) and line $ax+by=d$ $(a^2+b^2=1)$: $|ax_i + by_i - d|$

Find (a, b, d) to minimize the sum of squared perpendicular distances

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

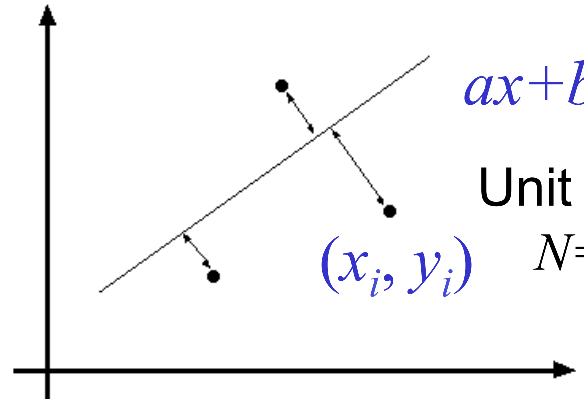


Source: S. Lazebnik

Total least squares

Distance between point (x_i, y_i) and line $ax+by=d$ $(a^2+b^2=1)$: $|ax_i + by_i - d|$

Find (a, b, d) to minimize the sum of squared perpendicular distances



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

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

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

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

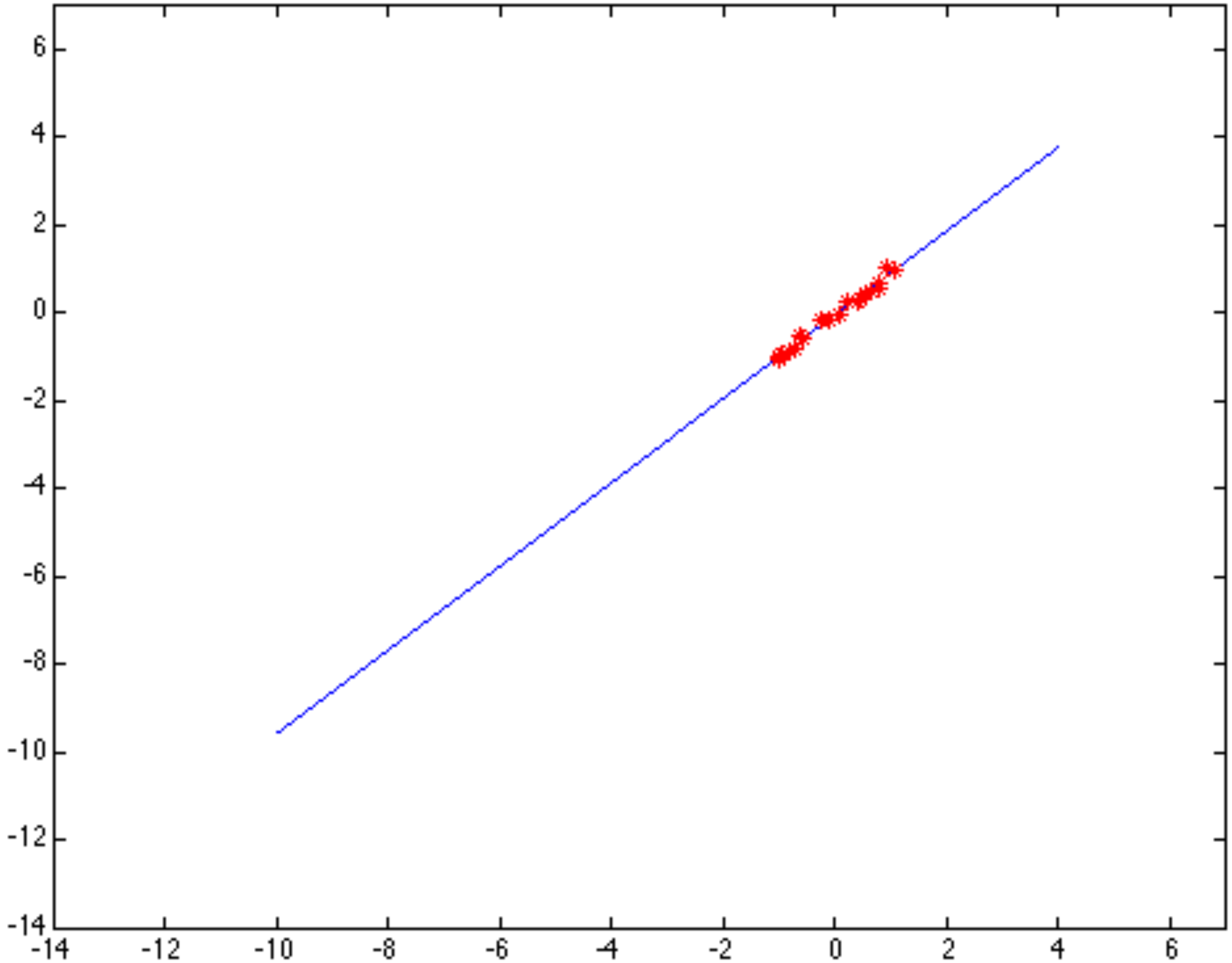
$$\frac{dE}{dN} = 2(U^T U)N = 0$$

Solution to $(U^T U)N = 0$, subject to $||N||^2 = 1$: eigenvector of $U^T U$ associated with the smallest eigenvalue (least squares solution to *homogeneous linear system* $UN = 0$)

Source: S. Lazebnik

Least squares: Robustness to noise

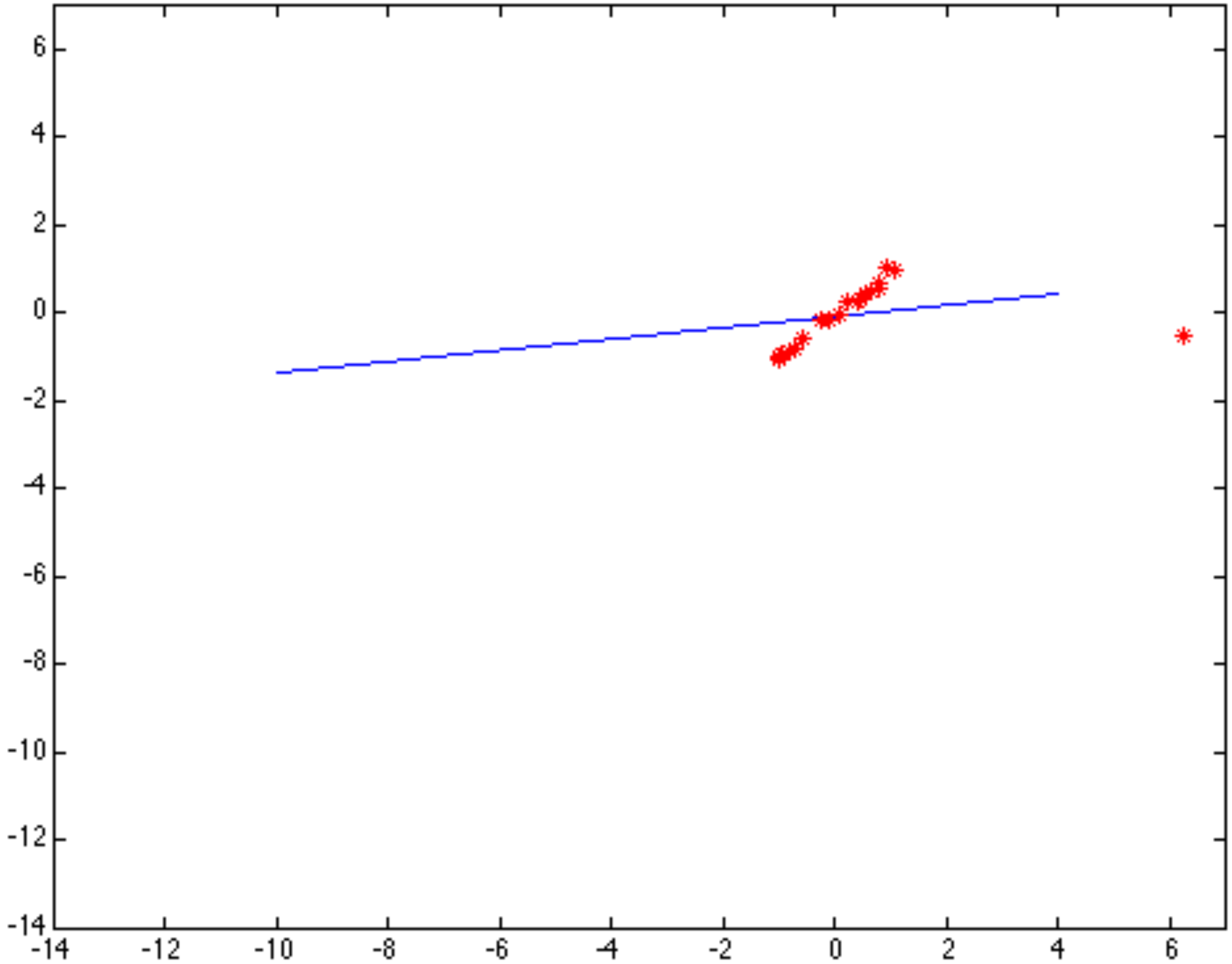
Least squares fit to the red points:



Source: S. Lazebnik

Least squares: Robustness to noise

Least squares fit with an outlier:



Problem: squared error heavily penalizes outliers

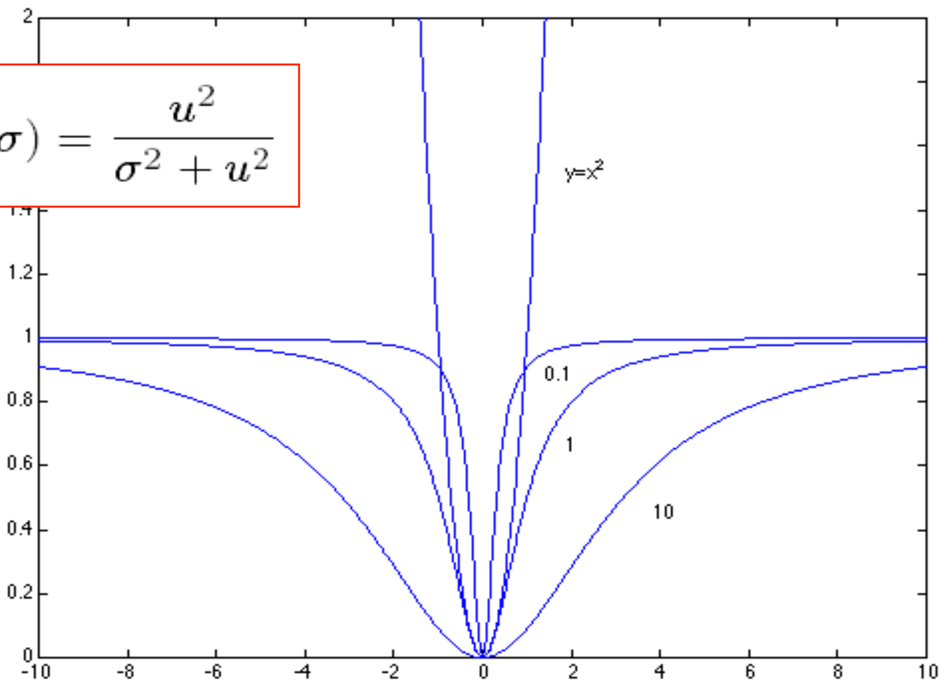
Robust estimators

- General approach: find model parameters θ that minimize

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

$r_i(x_i, \theta)$ – residual of ith point w.r.t. model parameters θ
ρ – robust function with scale parameter σ

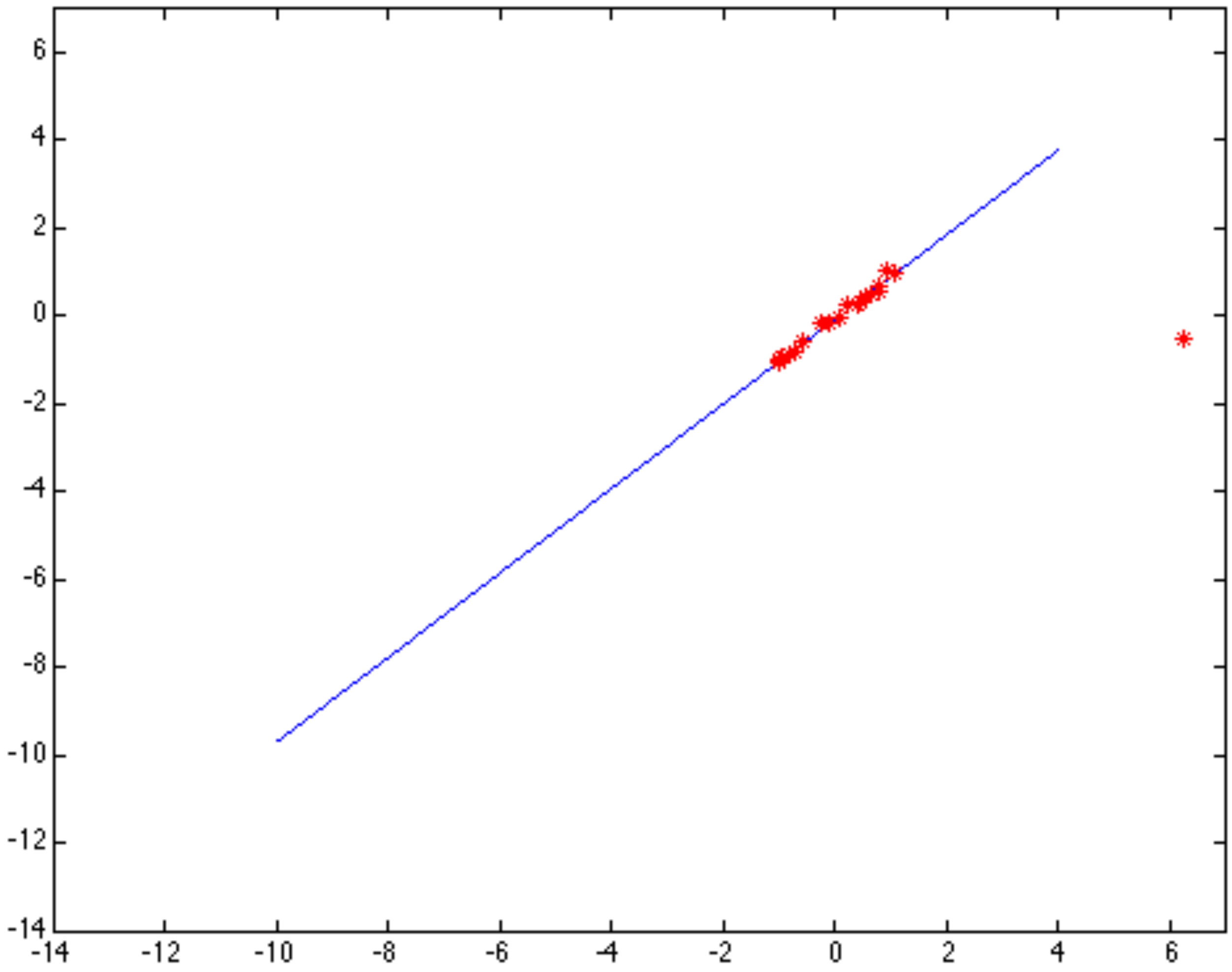
$$\rho(u;\sigma) = \frac{u^2}{\sigma^2 + u^2}$$



The robust function ρ behaves like squared distance for small values of the residual *u* but saturates for larger values of *u*

Source: S. Lazebnik

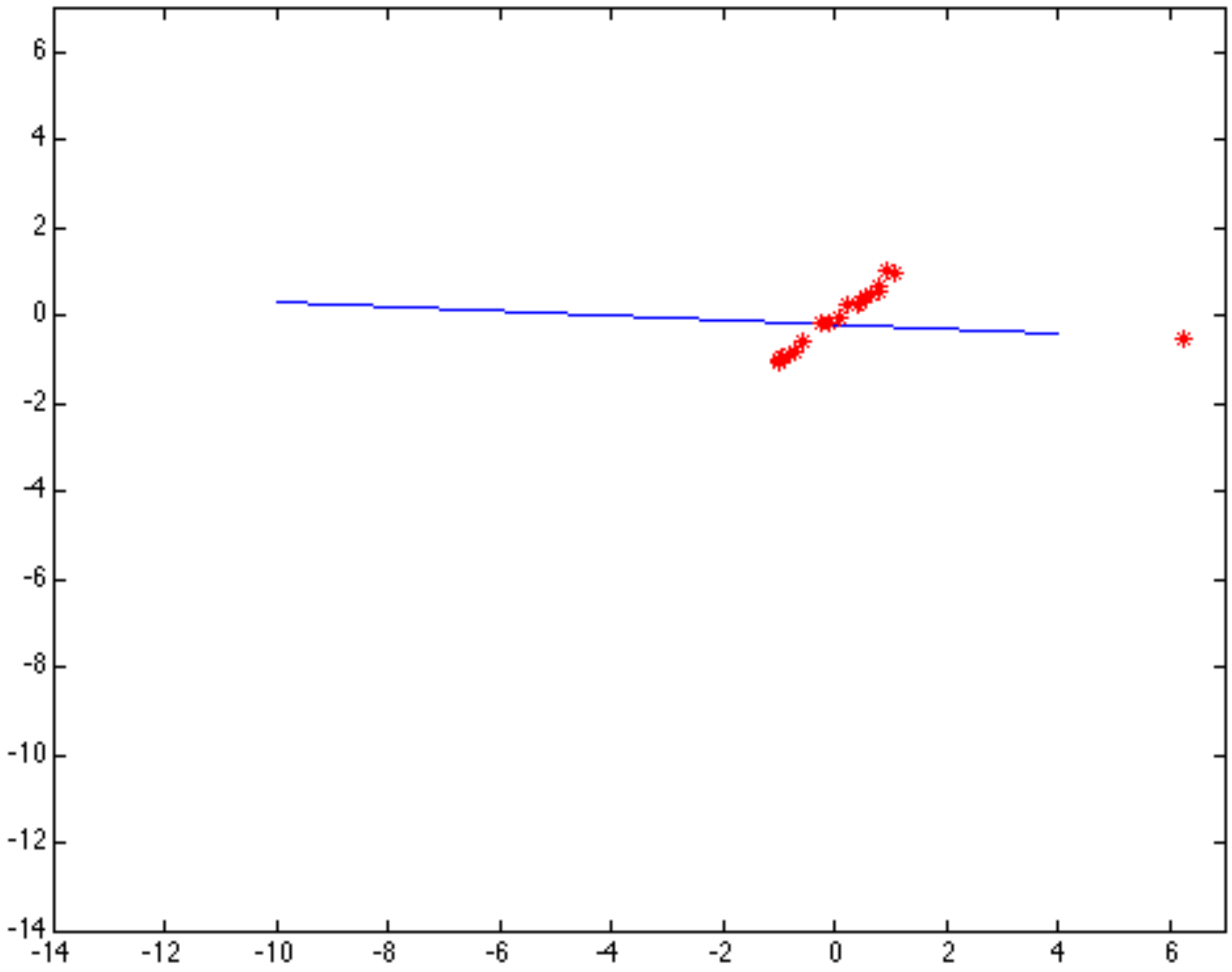
Choosing the scale: Just right



The effect of the outlier is minimized

Source: S. Lazebnik

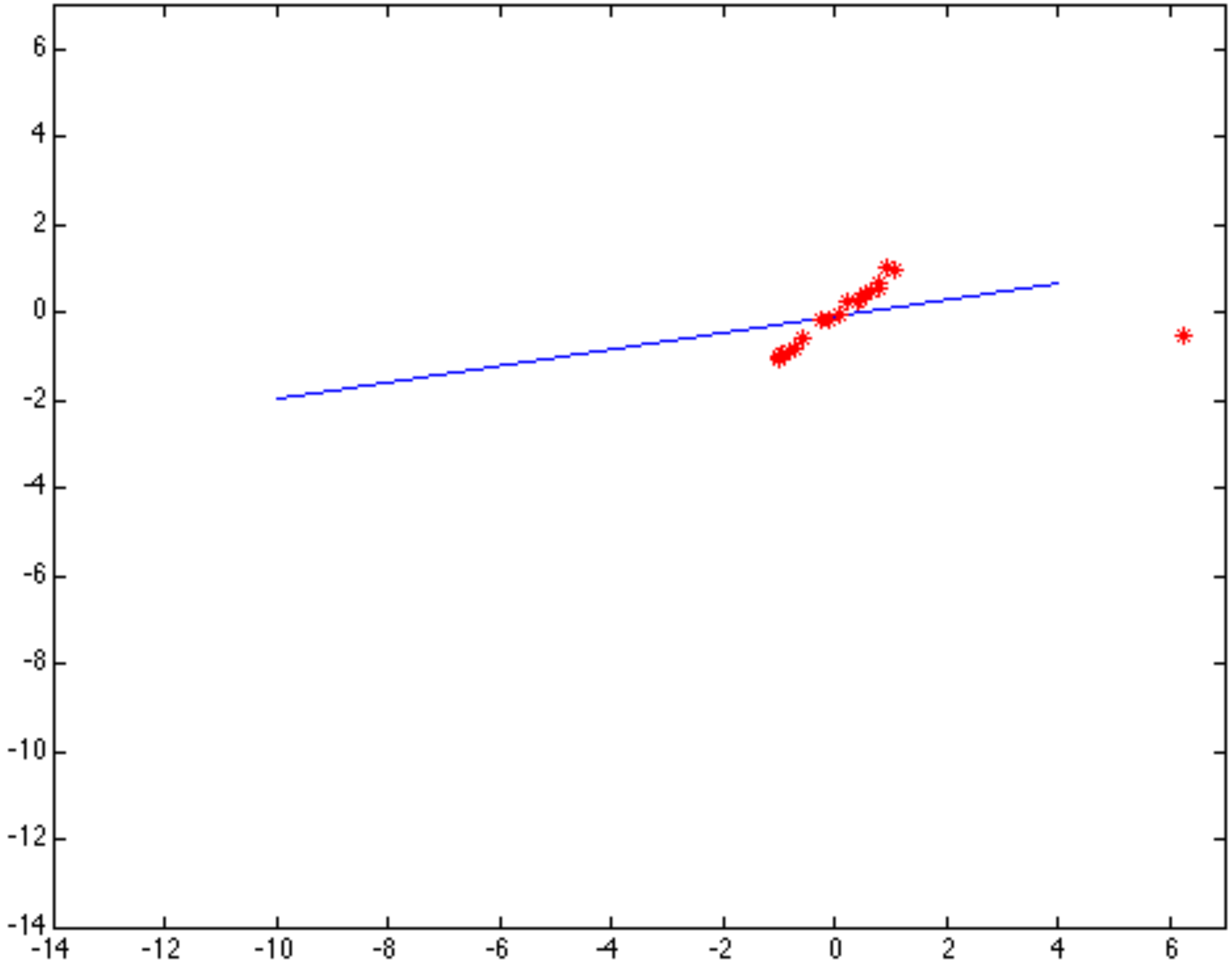
Choosing the scale: Too small



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

Source: S. Lazebnik

Choosing the scale: Too large



Behaves much the same as least squares

Source: S. Lazebnik

Robust estimation: Details

- Robust fitting is a nonlinear optimization problem that must be solved iteratively
- Least squares solution can be used for initialization
- Scale of robust function should be chosen adaptively based on median residual

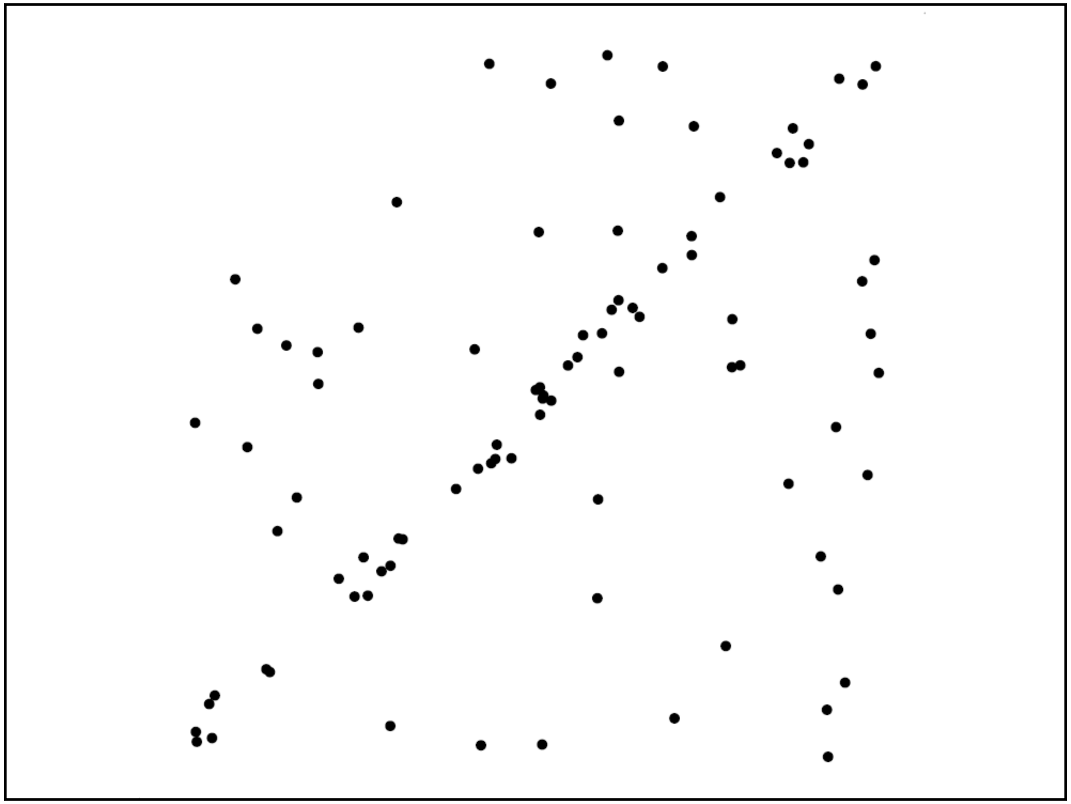
Source: S. Lazebnik

RANSAC

- Robust fitting can deal with a few outliers – what if we have very many?
- Random sample consensus (RANSAC): Very general framework for model fitting in the presence of outliers
- Outline
  - Choose a small subset of points uniformly at random
  - Fit a model to that subset
  - Find all remaining points that are “close” to the model and reject the rest as outliers
  - Do this many times and choose the best model

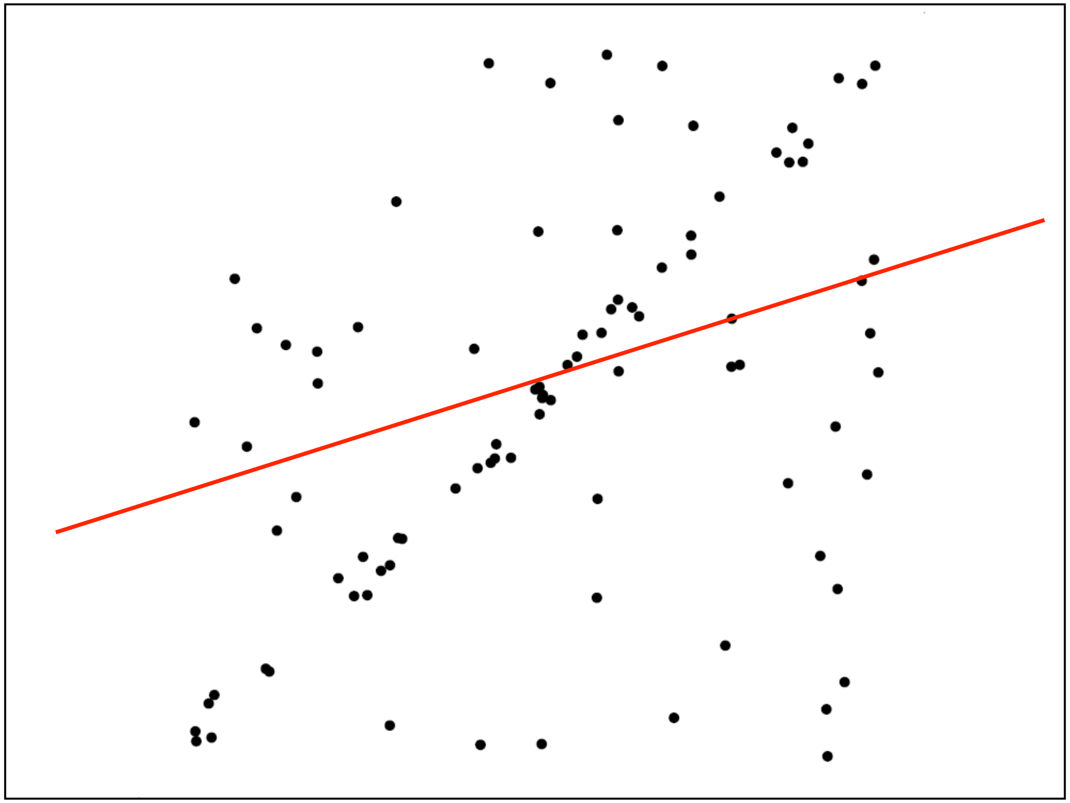
M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp 381-395, 1981.

RANSAC for line fitting example



Source: R. Raguram

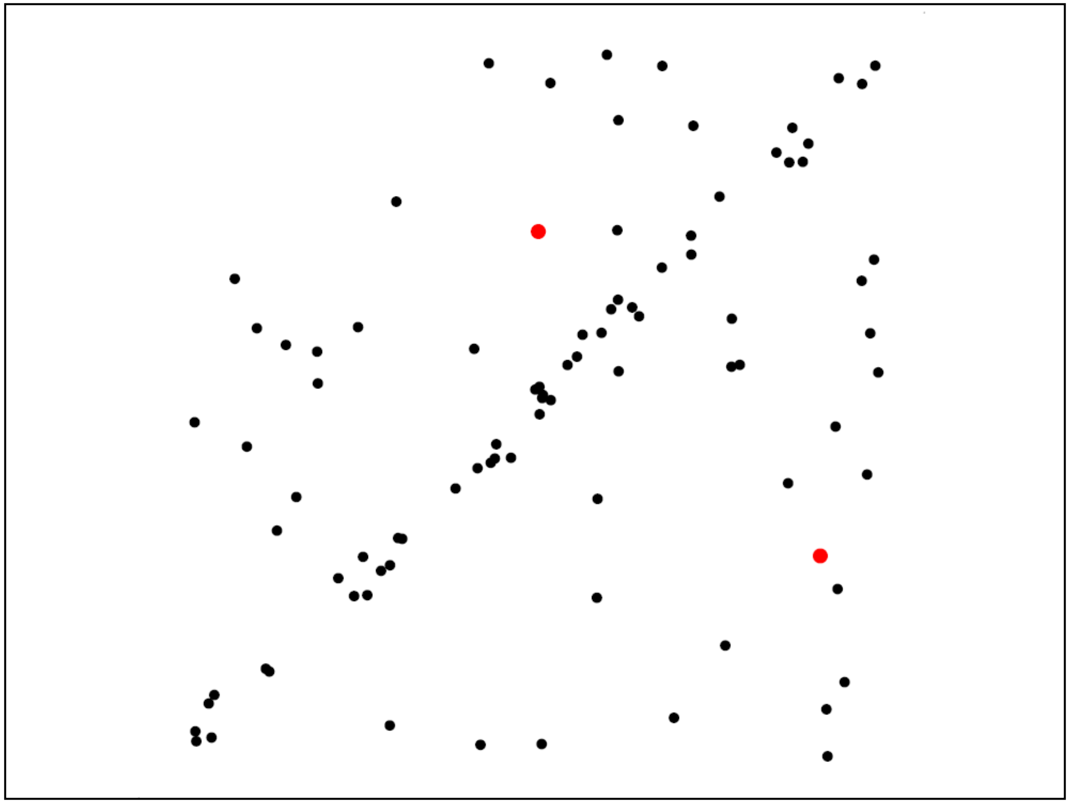
RANSAC for line fitting example



Least-squares fit

Source: R. Raguram

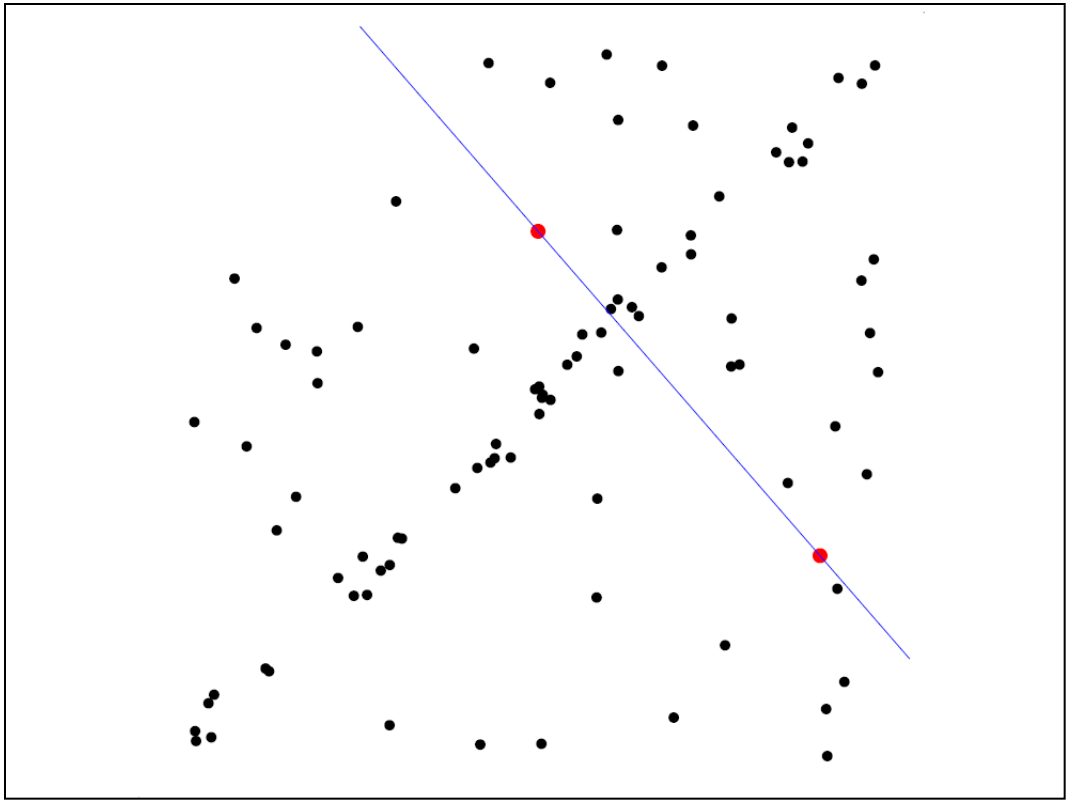
RANSAC for line fitting example



1. Randomly select minimal subset of points

Source: R. Raguram

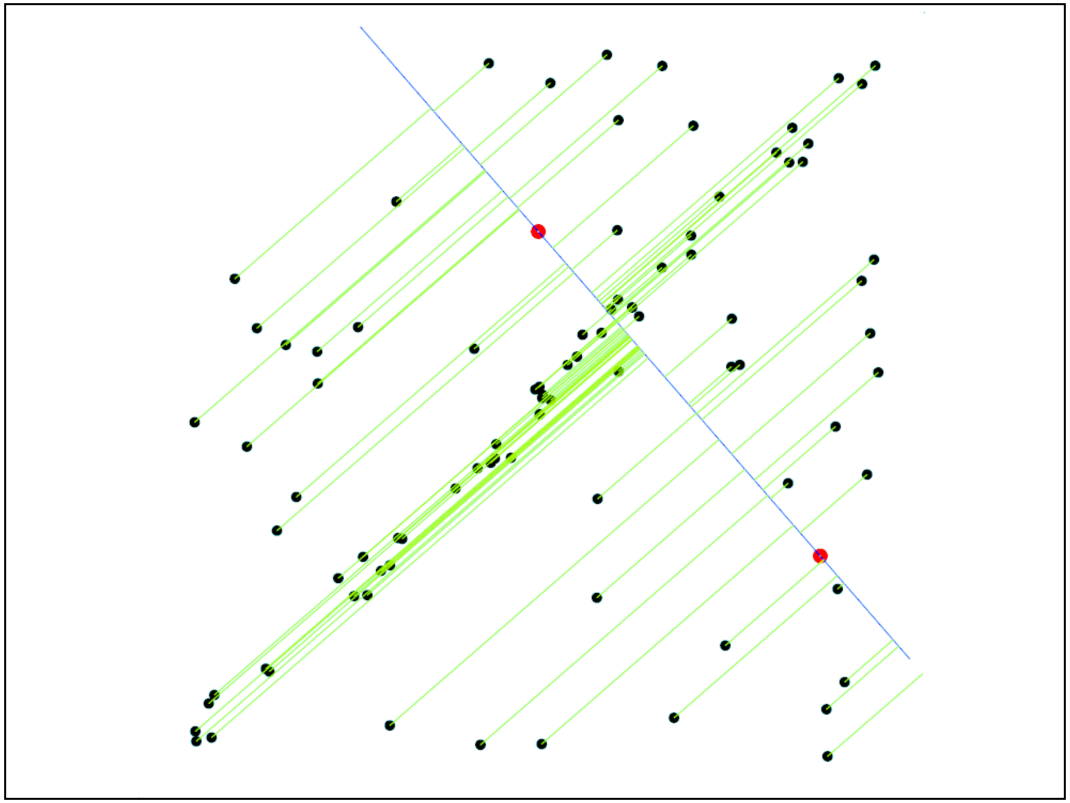
RANSAC for line fitting example



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

Source: R. Raguram

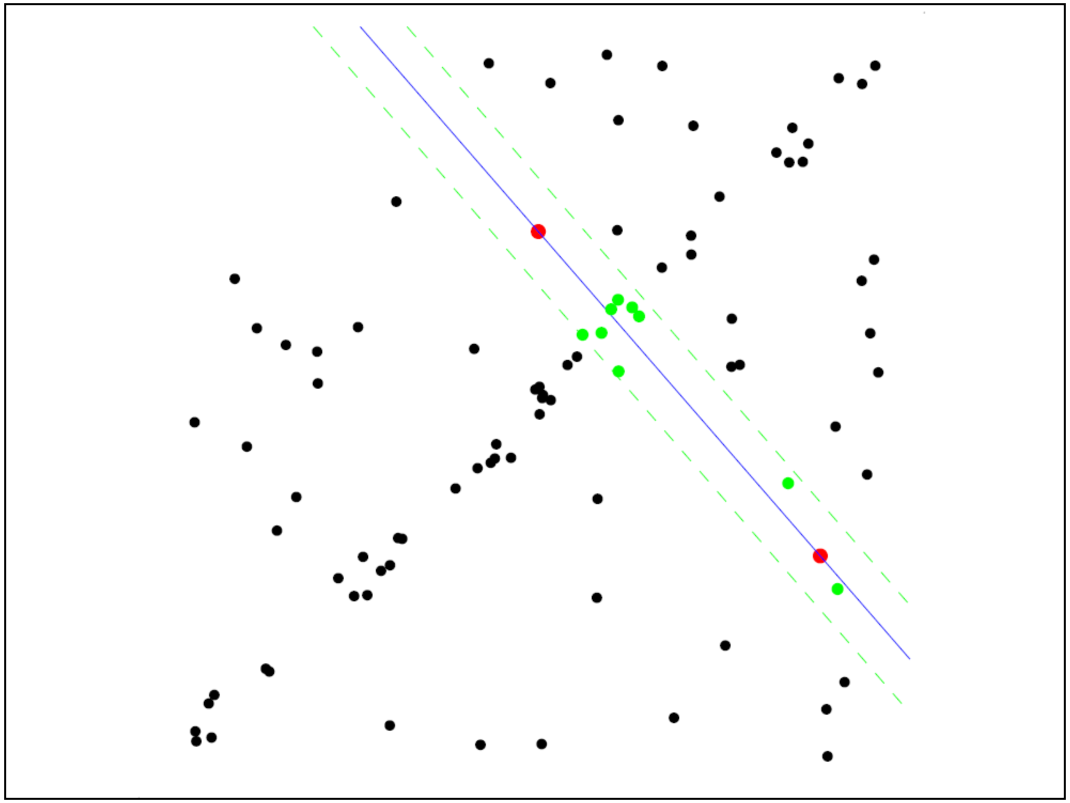
RANSAC for line fitting example



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

Source: R. Raguram

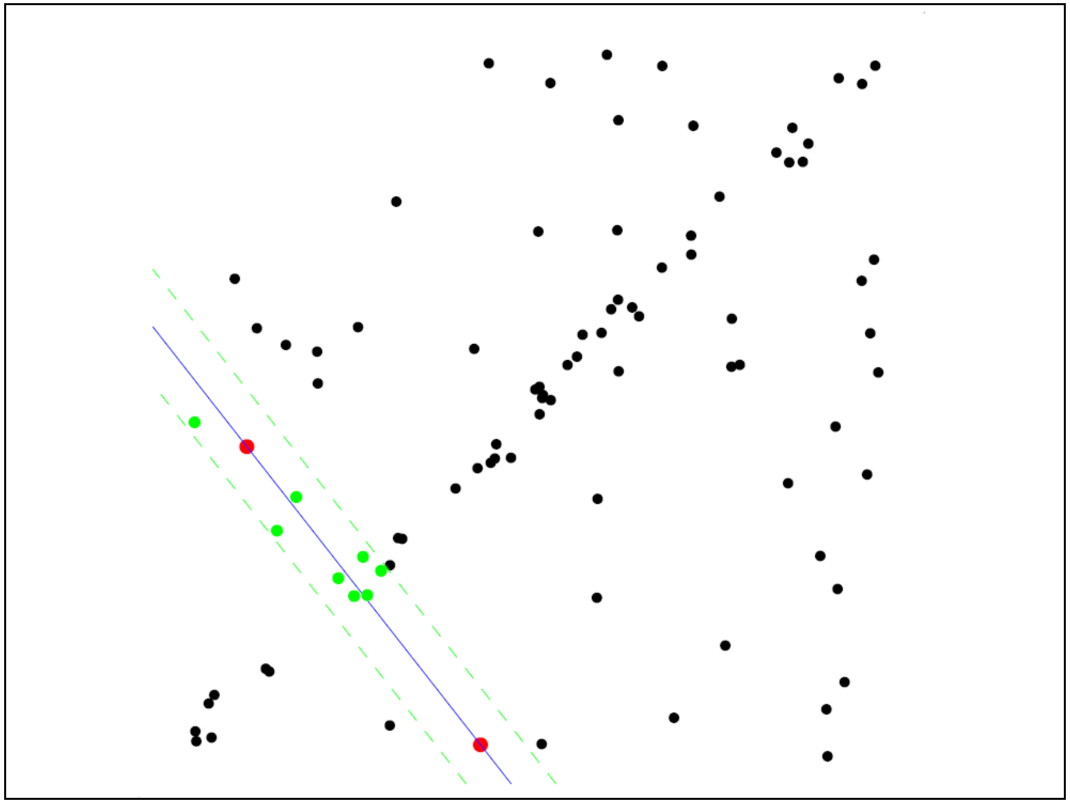
RANSAC for line fitting example



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

Source: R. Raguram

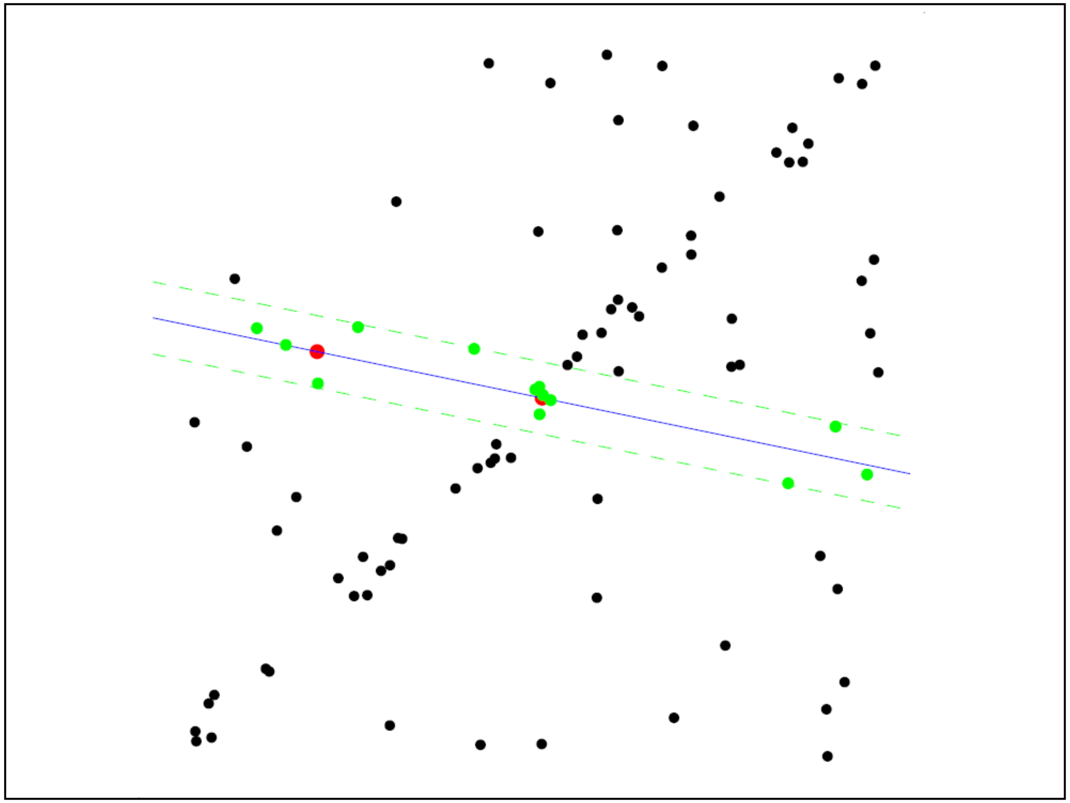
RANSAC for line fitting example



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

Source: R. Raguram

RANSAC for line fitting example

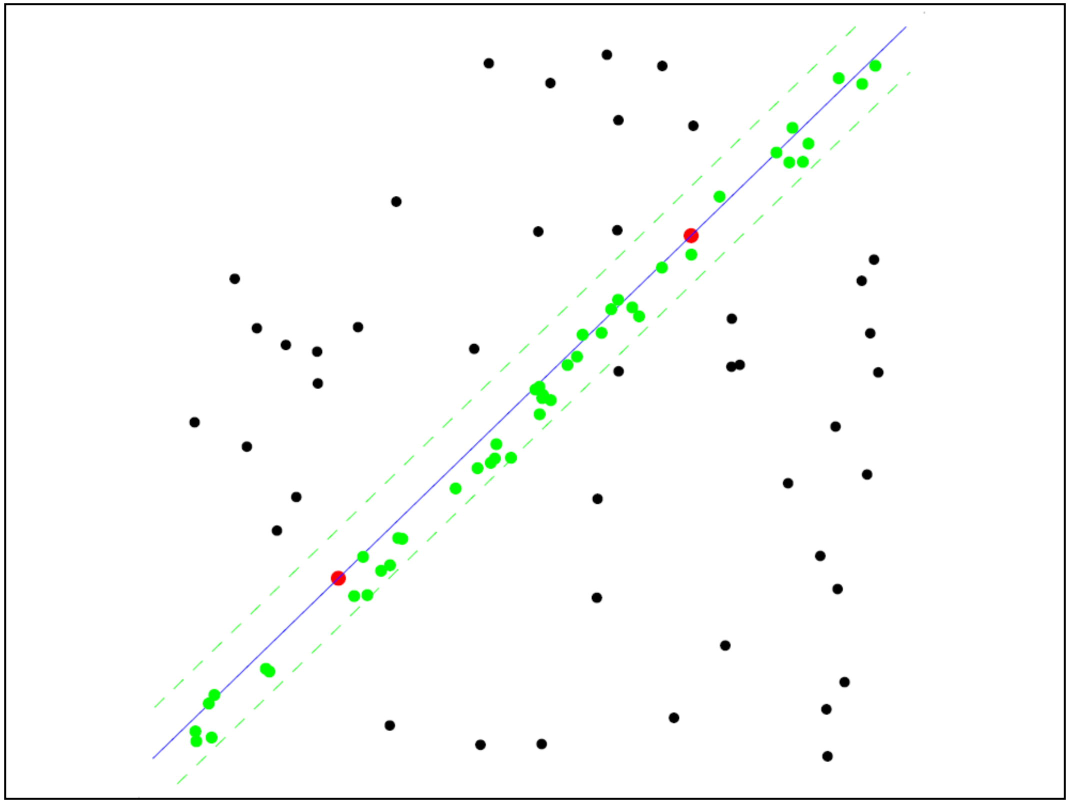


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

RANSAC for line fitting example

Uncontaminated sample

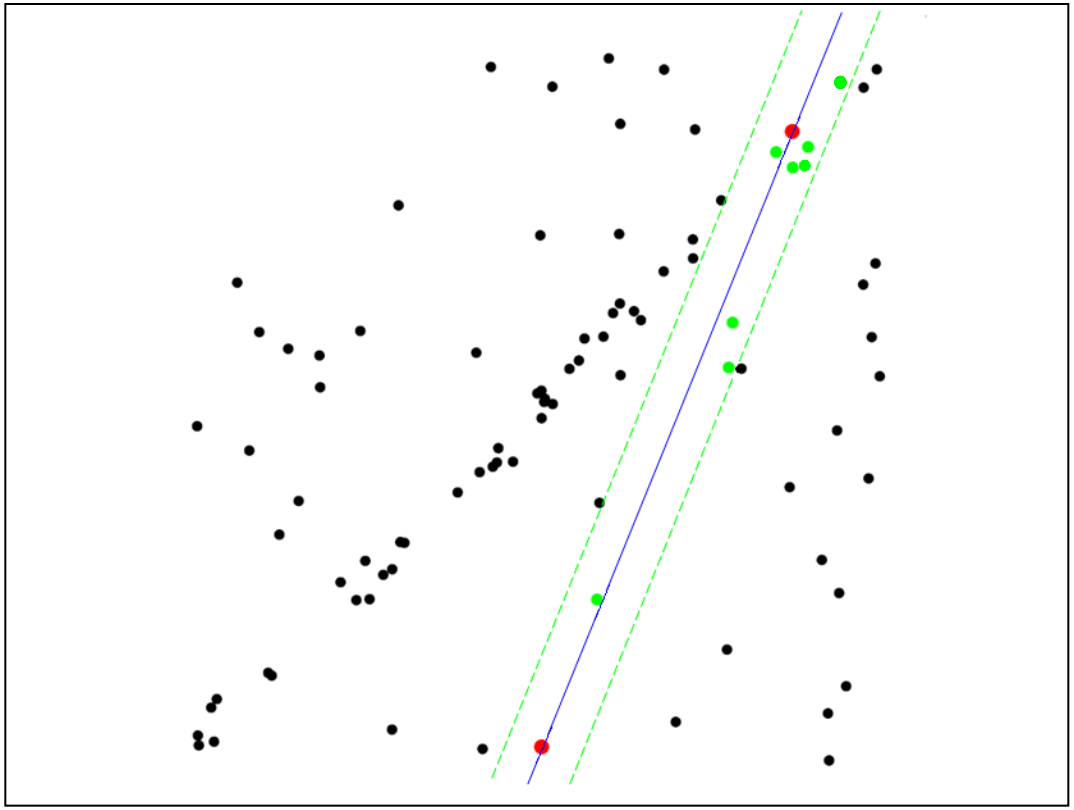


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Source: S. Lazebnik

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

Source: R. Raguram

Source: S. Lazebnik

RANSAC for line fitting

Repeat ***N*** times:

- Draw ***s*** points uniformly at random
- Fit line to these ***s*** points
- Find *inliers* to this line among the remaining points (i.e., points whose distance from the line is less than ***t***)
- If there are ***d*** or more inliers, accept the line and refit using all inliers

Choosing the parameters

- Initial number of points ***s***
  - Typically minimum number needed to fit the model
- Distance threshold ***t***
  - Choose ***t*** so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples ***N***
  - Choose ***N*** so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

Source: M. Pollefeys

Choosing the parameters

- Initial number of points ***s***
  - Typically minimum number needed to fit the model
- Distance threshold ***t***
  - Choose ***t*** so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples ***N***
  - Choose ***N*** so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)

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

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

	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

Source: M. Pollefeys

Choosing the parameters

- Initial number of points ***s***
  - Typically minimum number needed to fit the model
- Distance threshold ***t***
  - Choose ***t*** so probability for inlier is *p* (e.g. 0.95)
  - Zero-mean Gaussian noise with std. dev. σ: $t^2=3.84\sigma^2$
- Number of samples ***N***
  - Choose ***N*** so that, with probability *p*, at least one random sample is free from outliers (e.g. *p*=0.99) (outlier ratio: *e*)
- Consensus set size ***d***
  - Should match expected inlier ratio

Source: M. Pollefeys

Adaptively determining the number of samples

- Outlier ratio *e* is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield *e*=0.2
- Adaptive procedure:
  - *N*=∞, *sample_count* =0
  - While *N* >*sample_count*
    - Choose a sample and count the number of inliers
    - If inlier ratio is highest of any found so far, set e = 1 – (number of inliers)/(total number of points)
    - Recompute *N* from *e:*

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

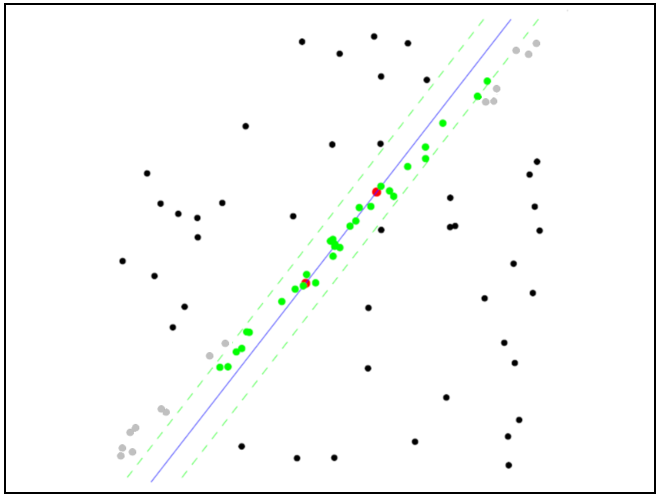
    - Increment the *sample_count* by 1

Source: M. Pollefeys

Source: S. Lazebnik

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



Source: S. Lazebnik

Fitting: Review

- Least squares
- Robust fitting
- RANSAC

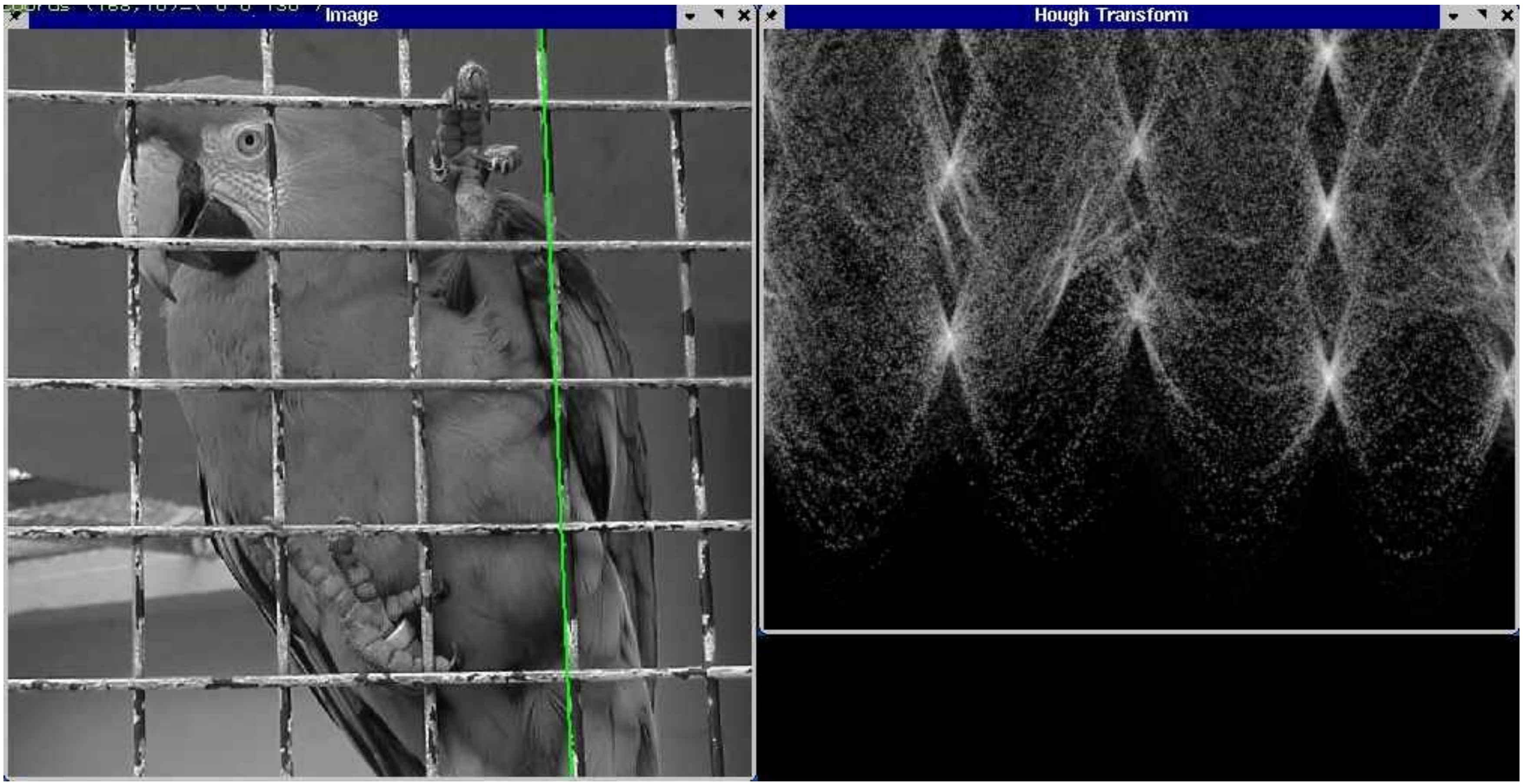
Source: S. Lazebnik

Fitting: Review

- ✓ If we know which points belong to the line, how do we find the "optimal" line parameters?
  - ✓ Least squares
- ✓ What if there are outliers?
  - ✓ Robust fitting, RANSAC
- What if there are many lines?
  - Voting methods: RANSAC, Hough transform

Source: S. Lazebnik

Fitting: The Hough transform



Source: S. Lazebnik

Voting schemes

- Let each feature vote for all the models that are compatible with it
- Hopefully the noise features will not vote consistently for any single model
- Missing data doesn’t matter as long as there are enough features remaining to agree on a good model

Source: S. Lazebnik

Hough transform

- An early type of voting scheme
- General outline:
  - 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

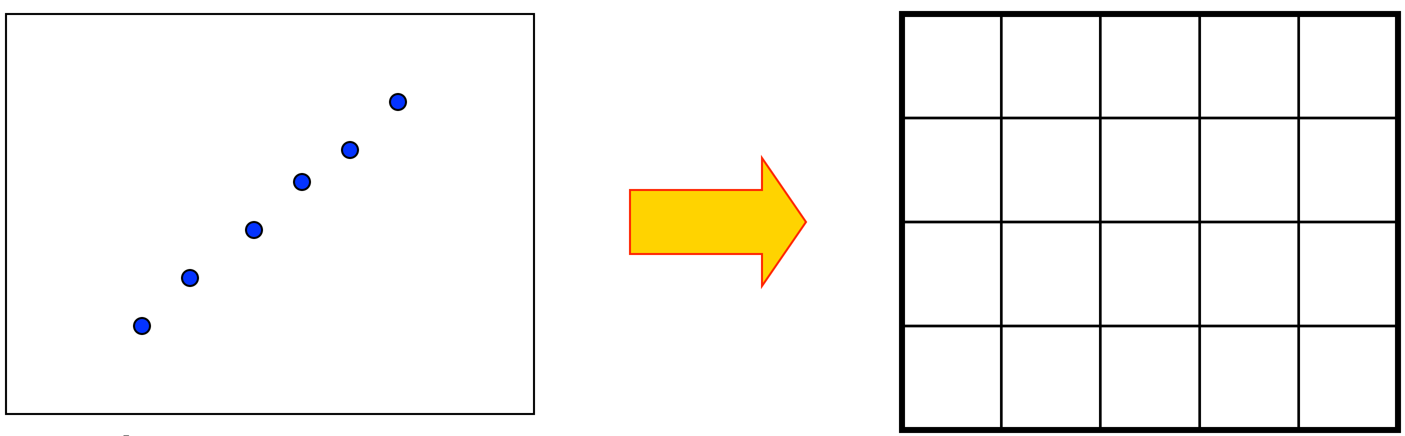


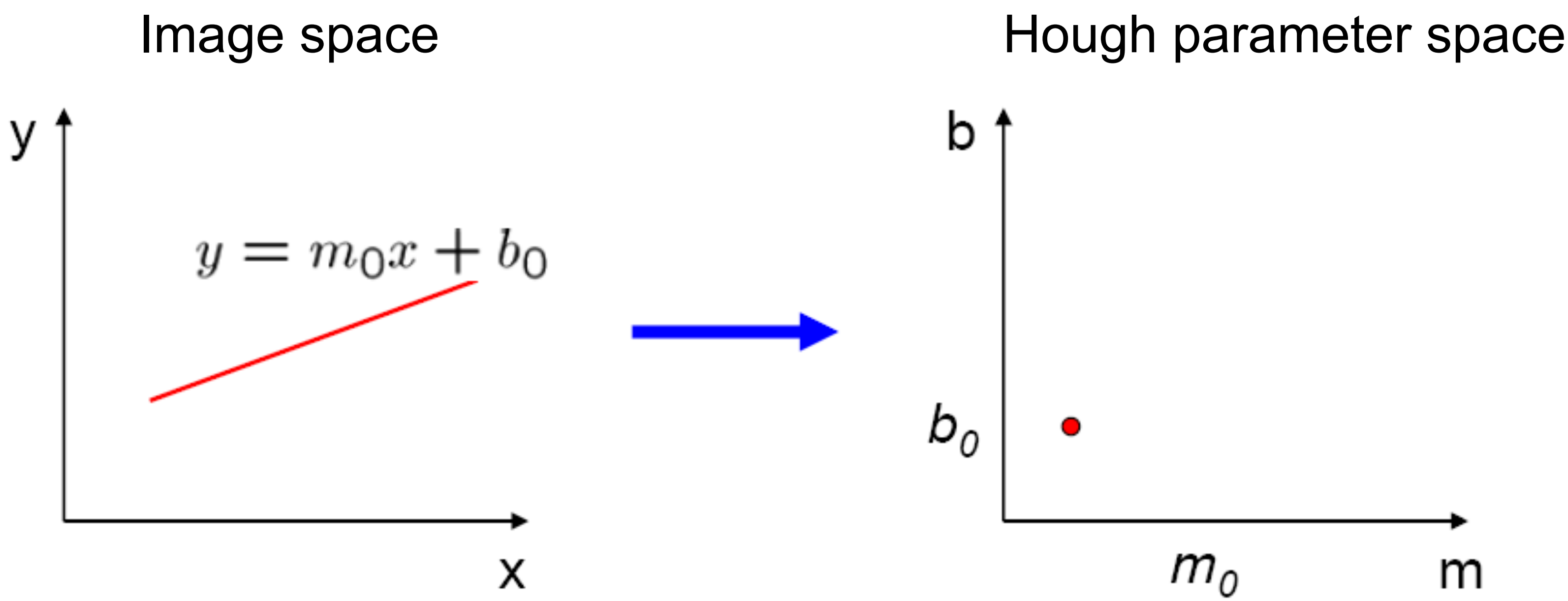
Image space   Hough parameter space

P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Source: S. Lazebnik

Parameter space representation

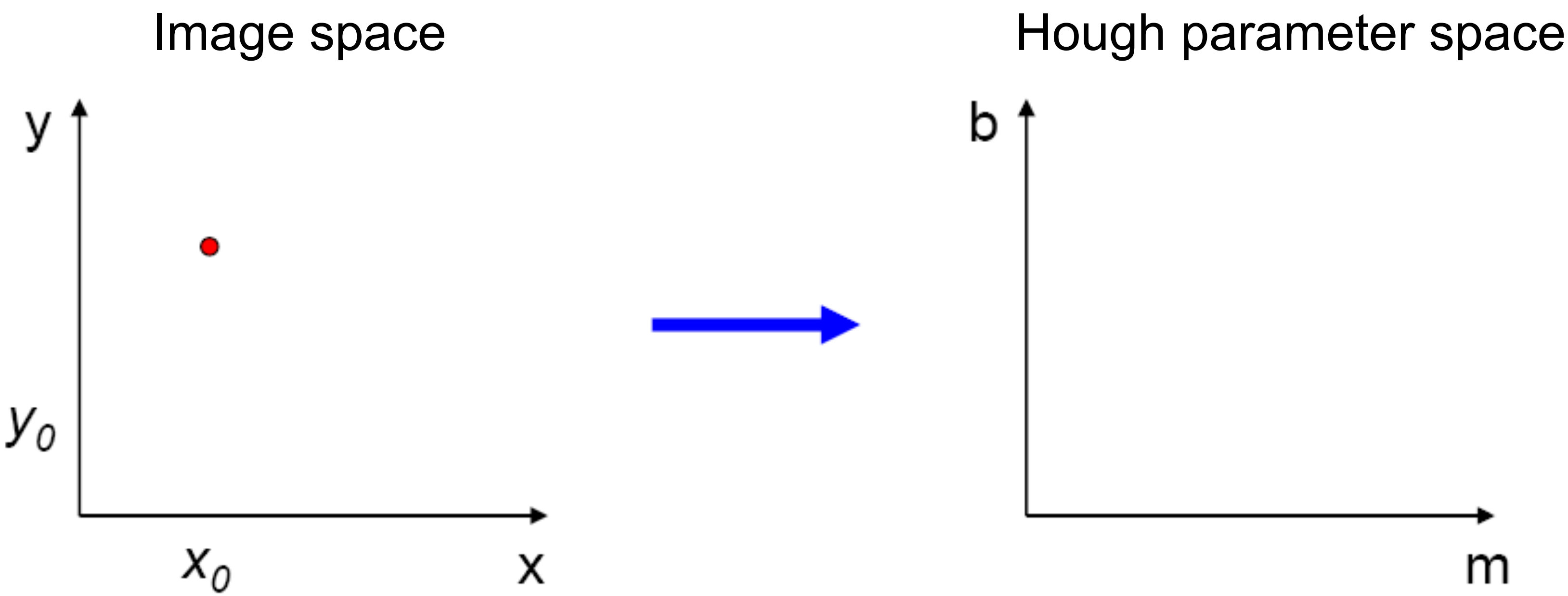
- A line in the image corresponds to a point in Hough space



Source: S. Seitz

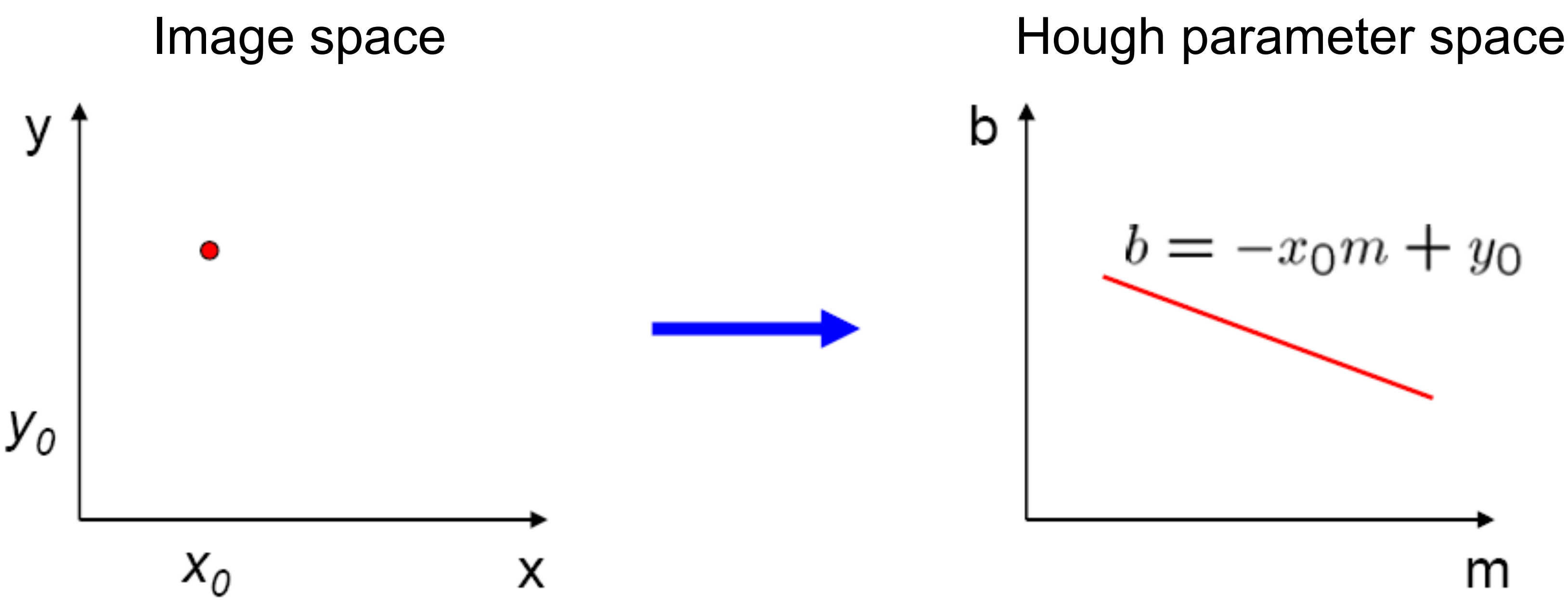
Parameter space representation

- What does a point (x_0, y_0) in the image space map to in the Hough space?



Parameter space representation

- What does a point (x_0, y_0) in the image space map to in the Hough space?
  - Answer: the solutions of $b = -x_0 m + y_0$
  - This is a line in Hough space

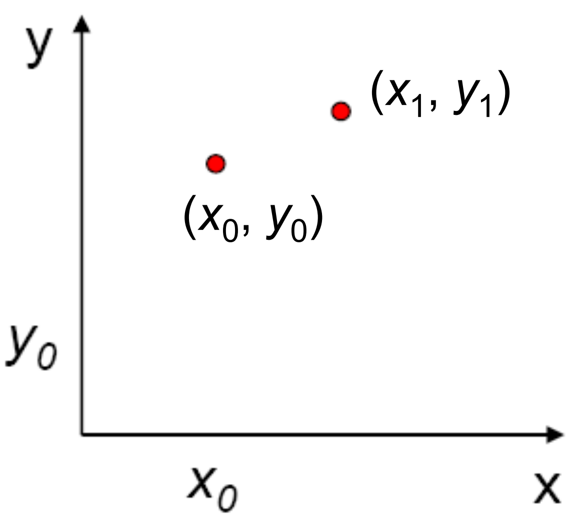


Source: S. Lazebnik

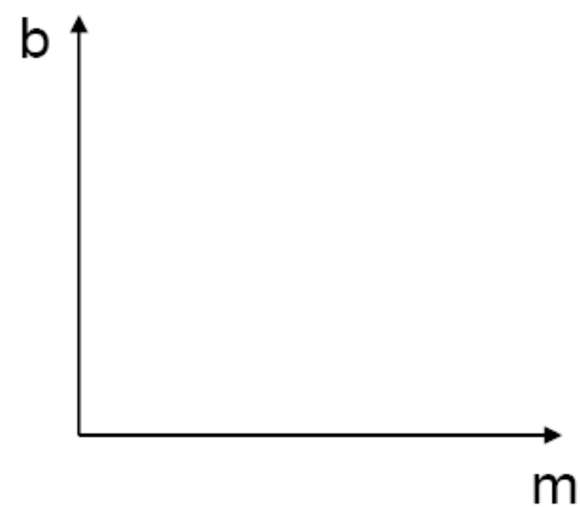
Parameter space representation

- Where is the line that contains both (x_0, y_0) and (x_1, y_1)?

Image space



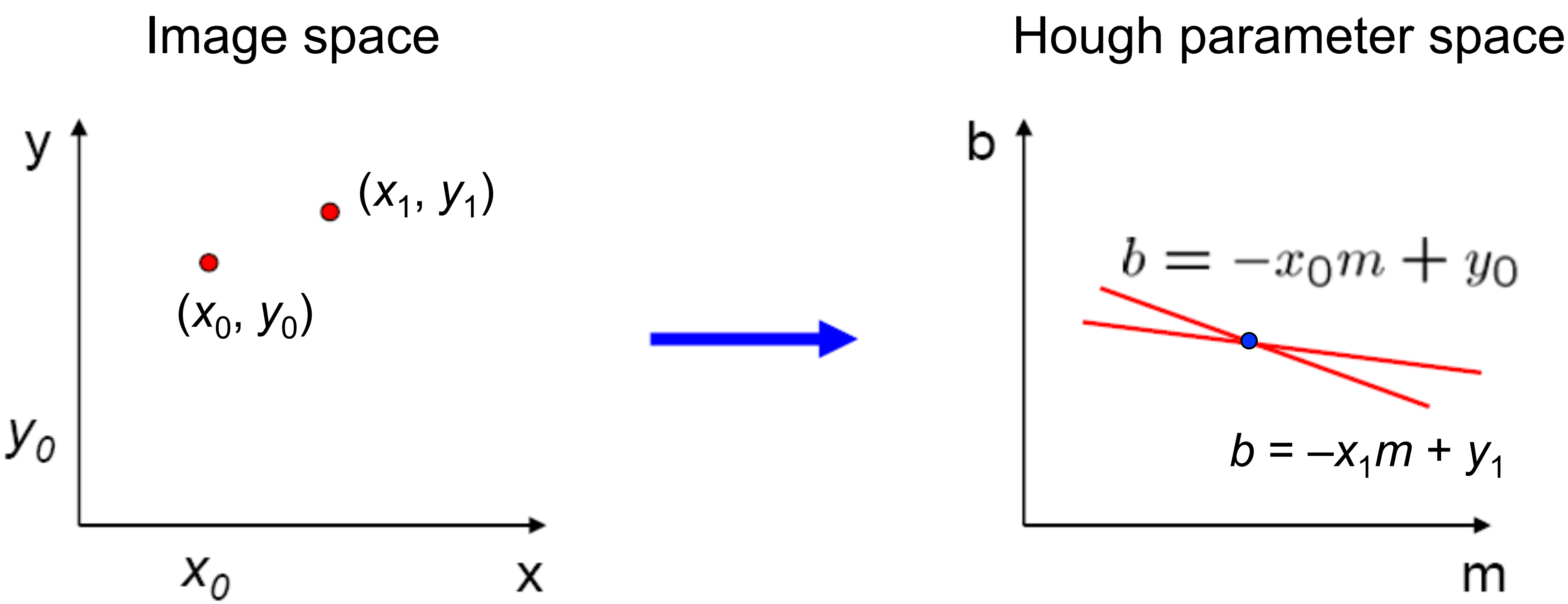
Hough parameter space



Source: S. Lazebnik

Parameter space representation

- Where is the line that contains both (x_0, y_0) and (x_1, y_1)?
  - It is the intersection of the lines $b = –x_0m + y_0$ and $b = –x_1m + y_1$



Source: S. Lazebnik

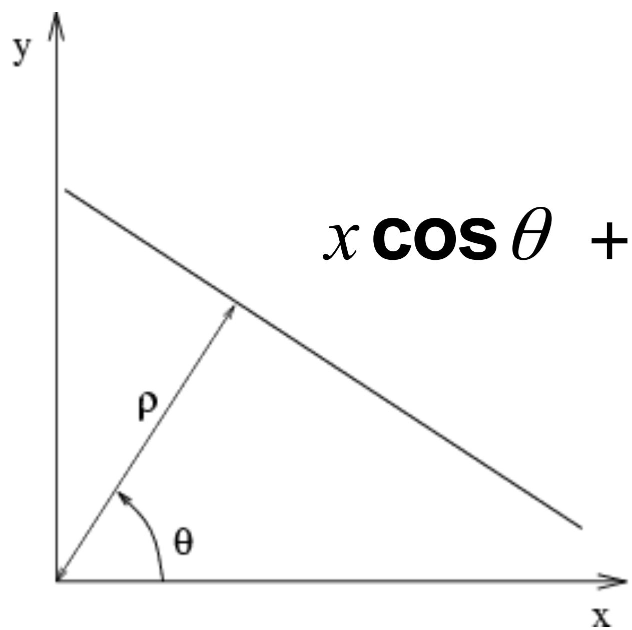
Parameter space representation

- Problems with the (m,b) space:
  - Unbounded parameter domains
  - Vertical lines require infinite m

Source: S. Lazebnik

Parameter space representation

- Problems with the (m,b) space:
  - Unbounded parameter domains
  - Vertical lines require infinite m
- Alternative: *polar representation*



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

Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

Algorithm outline

- Initialize accumulator H to all zeros
- For each feature point (x,y) in the image

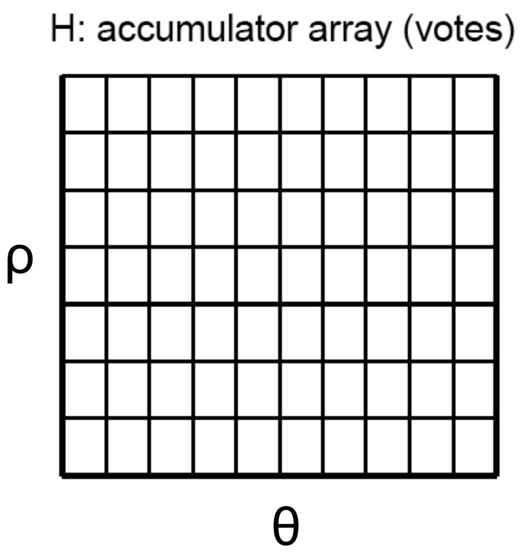
  For θ = 0 to 180

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

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

  end

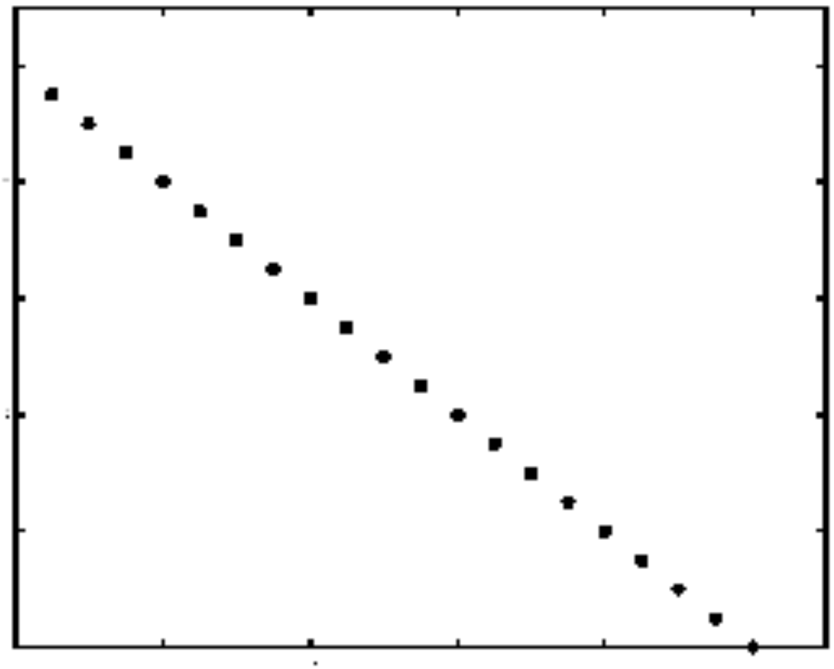
  end



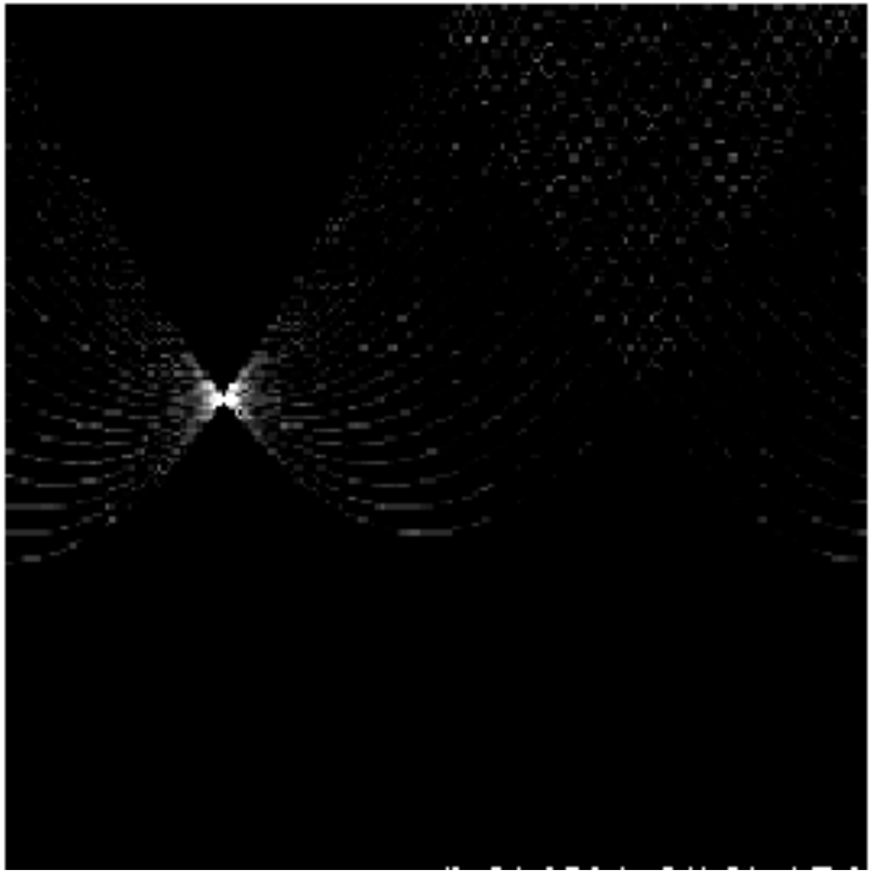
- Find the value(s) of (θ, ρ) where $H(\theta, \rho)$ is a local maximum
  - The detected line in the image is given by $\rho = x \cos \theta + y \sin \theta$

Source: S. Lazebnik

Basic illustration



features



votes

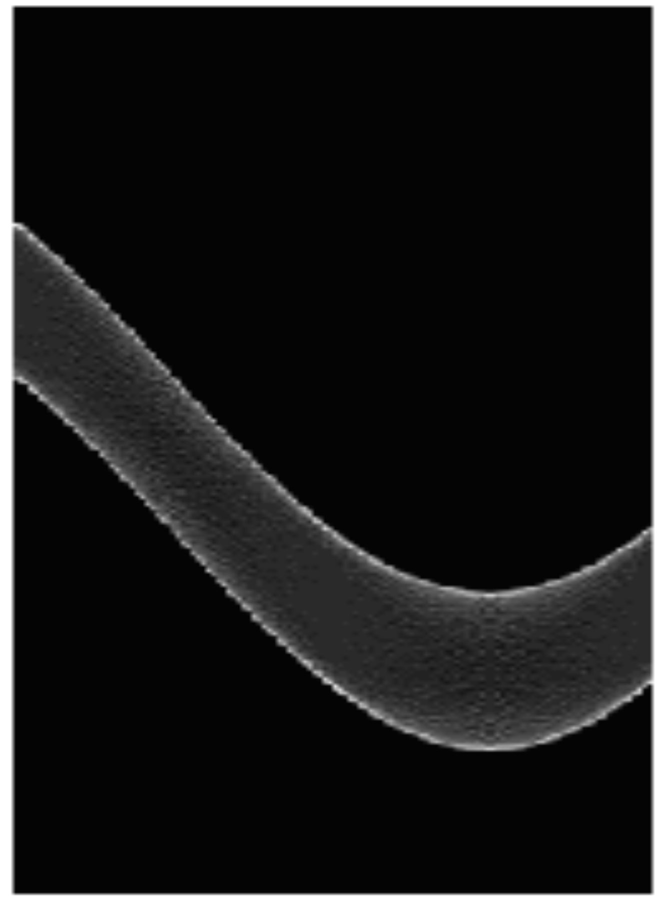
[Hough transform demo]

Source: S. Lazebnik

Other shapes

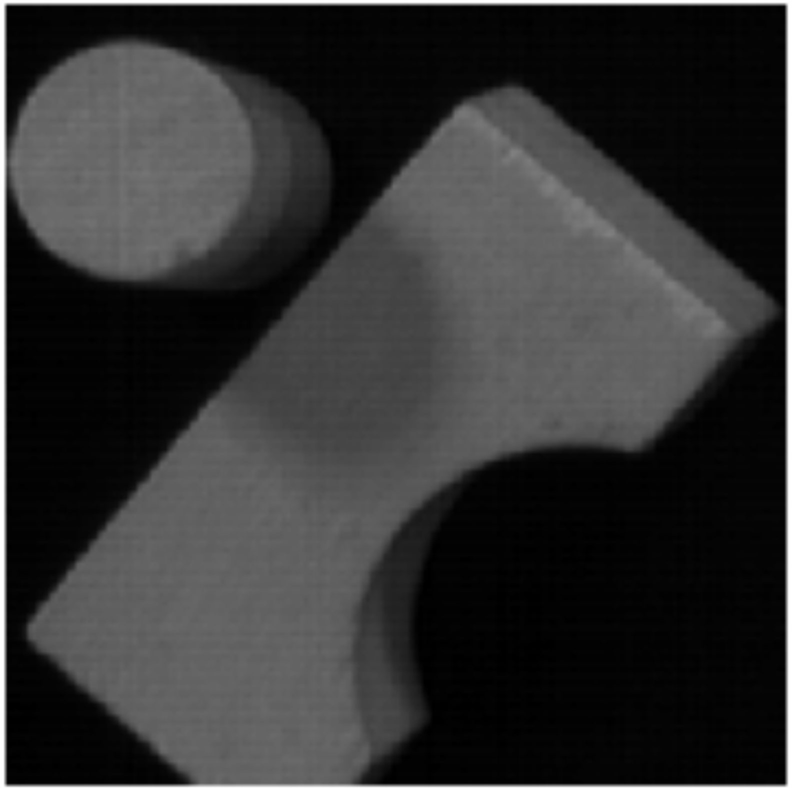
Square

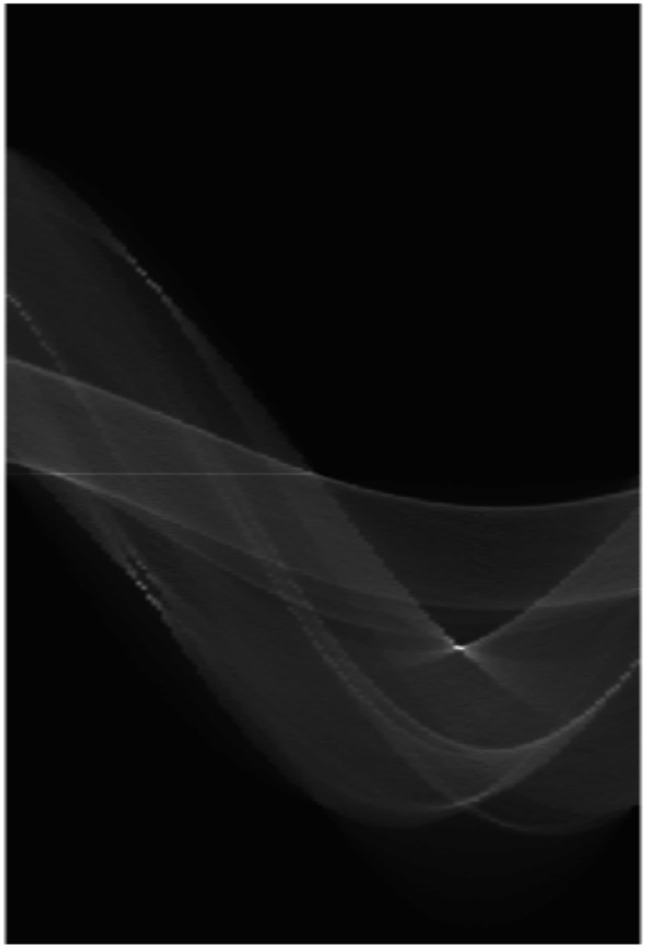
Circle



Source: S. Lazebnik

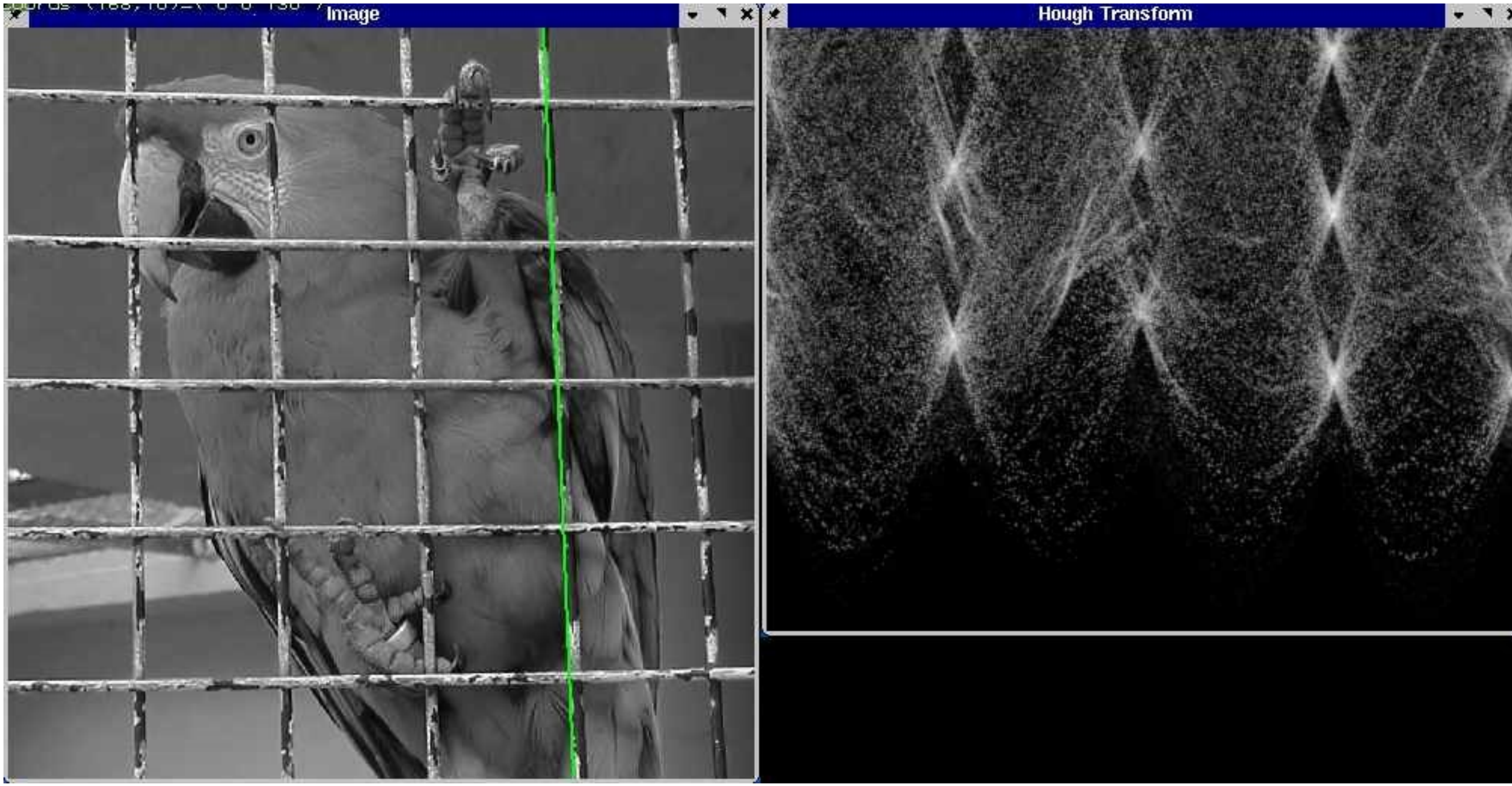
Several lines





Source: S. Lazebnik

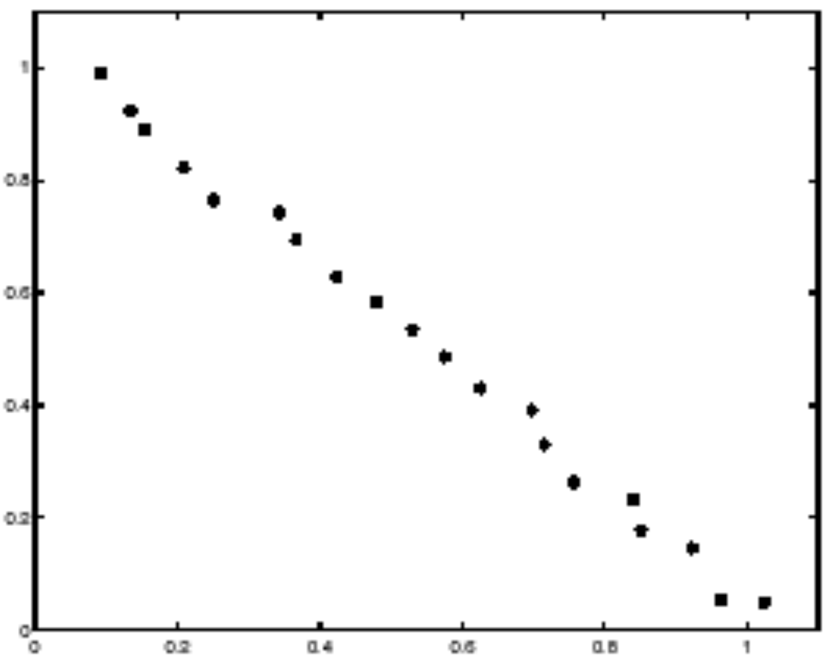
A more complicated image



http://ostatic.com/files/images/ss_hough.jpg

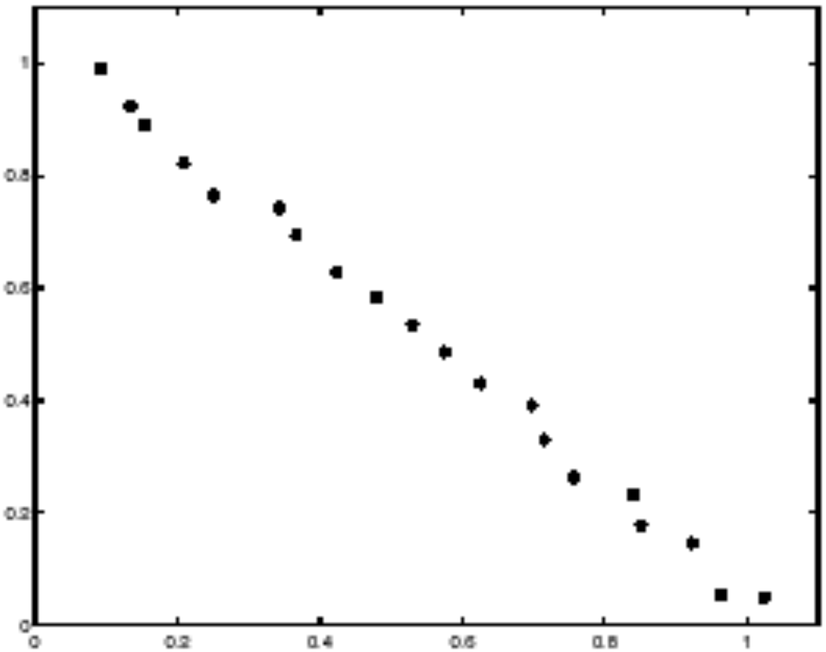
Source: S. Lazebnik

Effect of noise

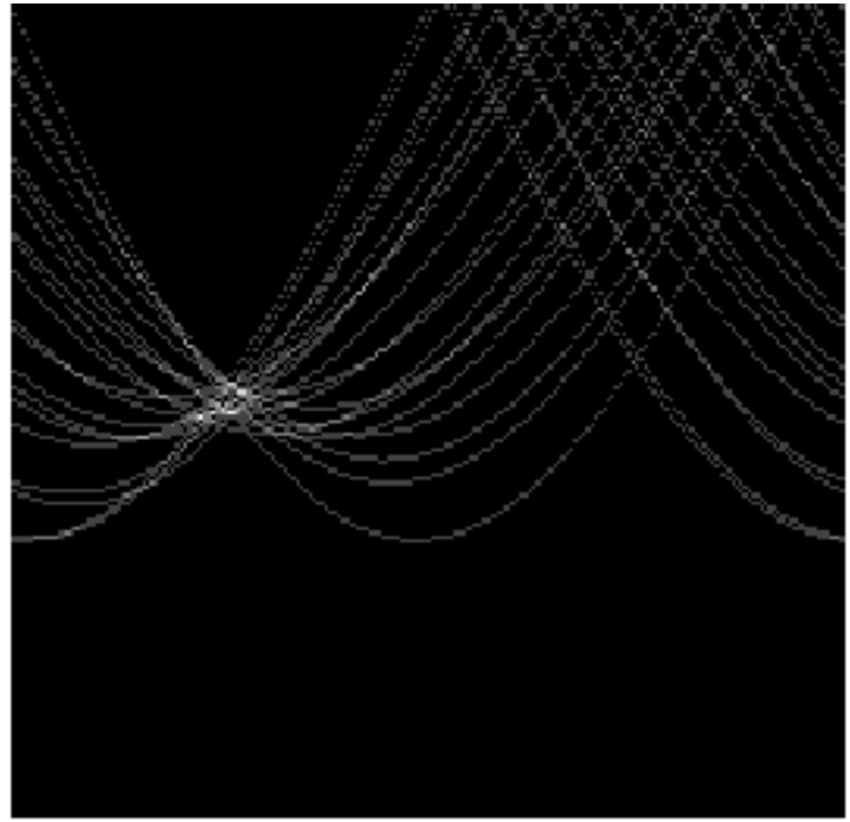


features

Effect of noise



features



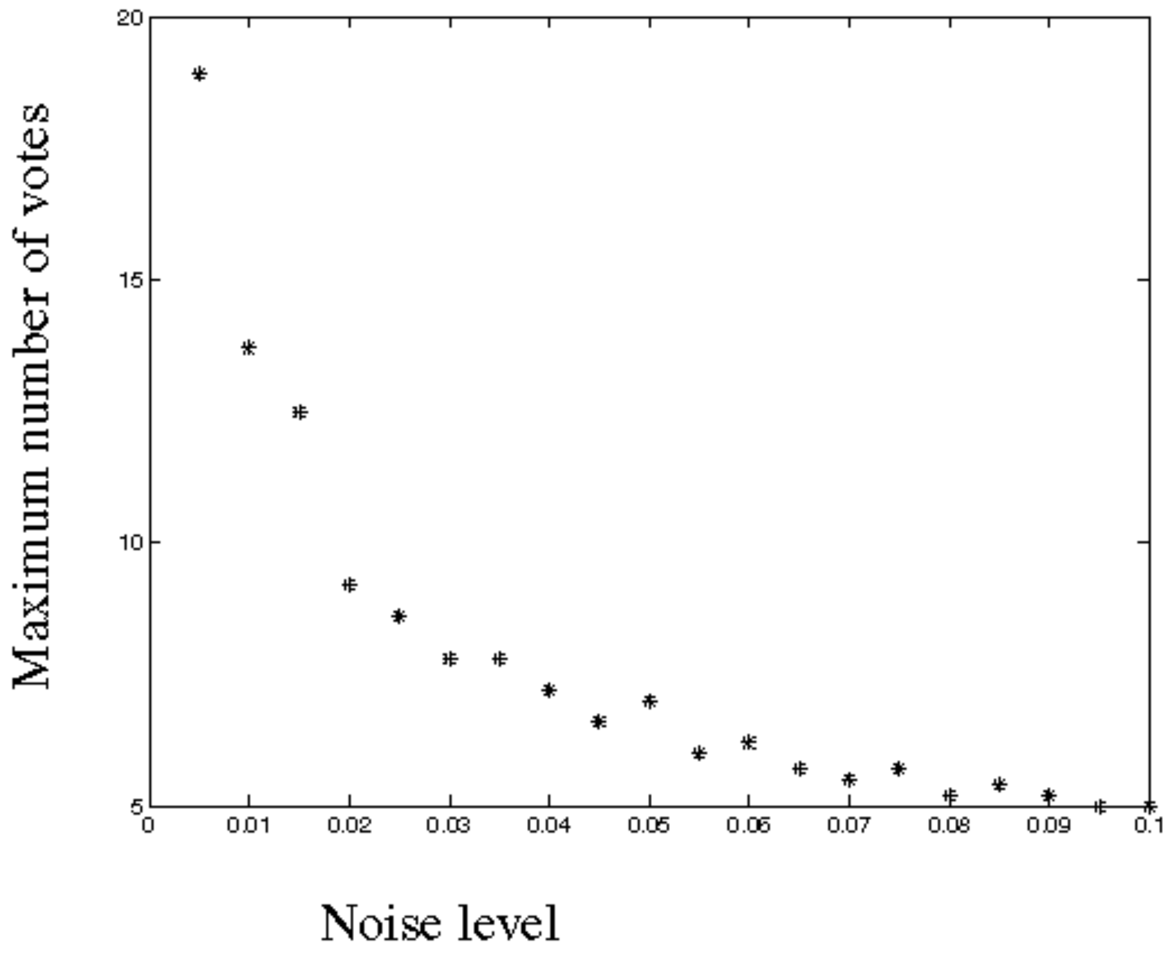
votes

Peak gets fuzzy and hard to locate

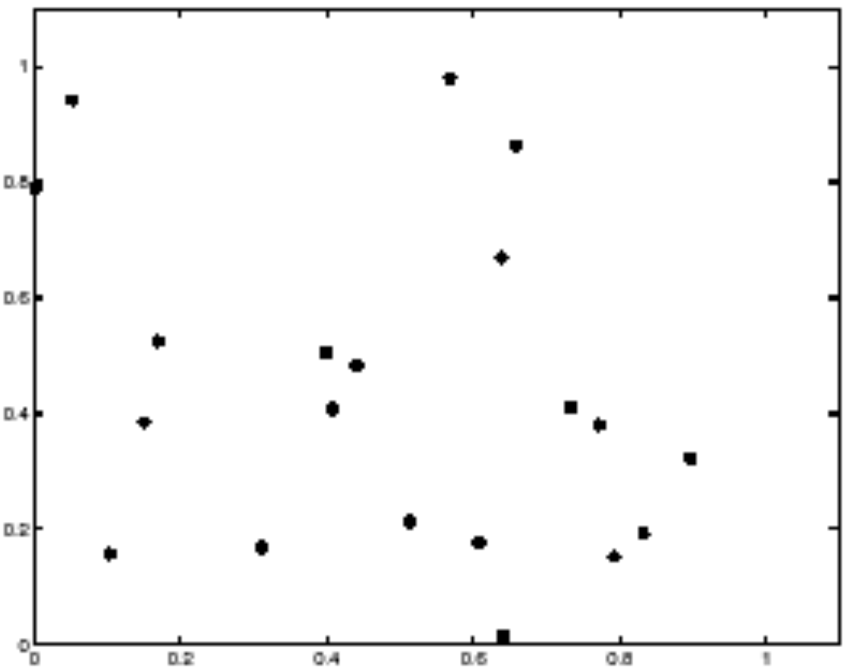
Source: S. Lazebnik

Effect of noise

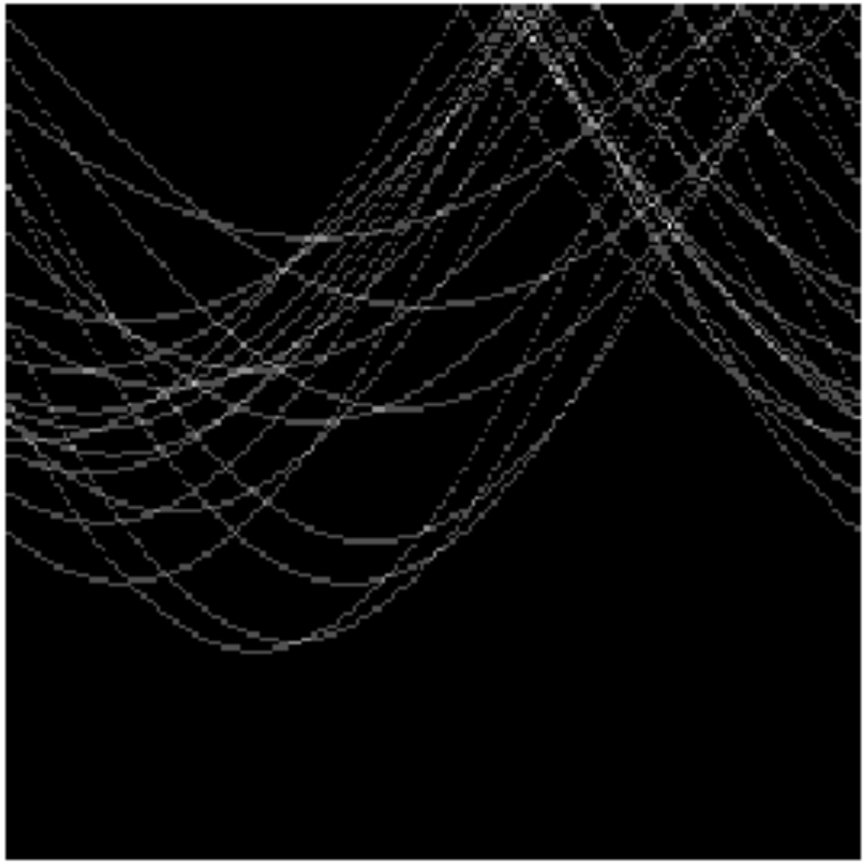
- Number of votes for a line of 20 points with increasing noise:



Random points



features



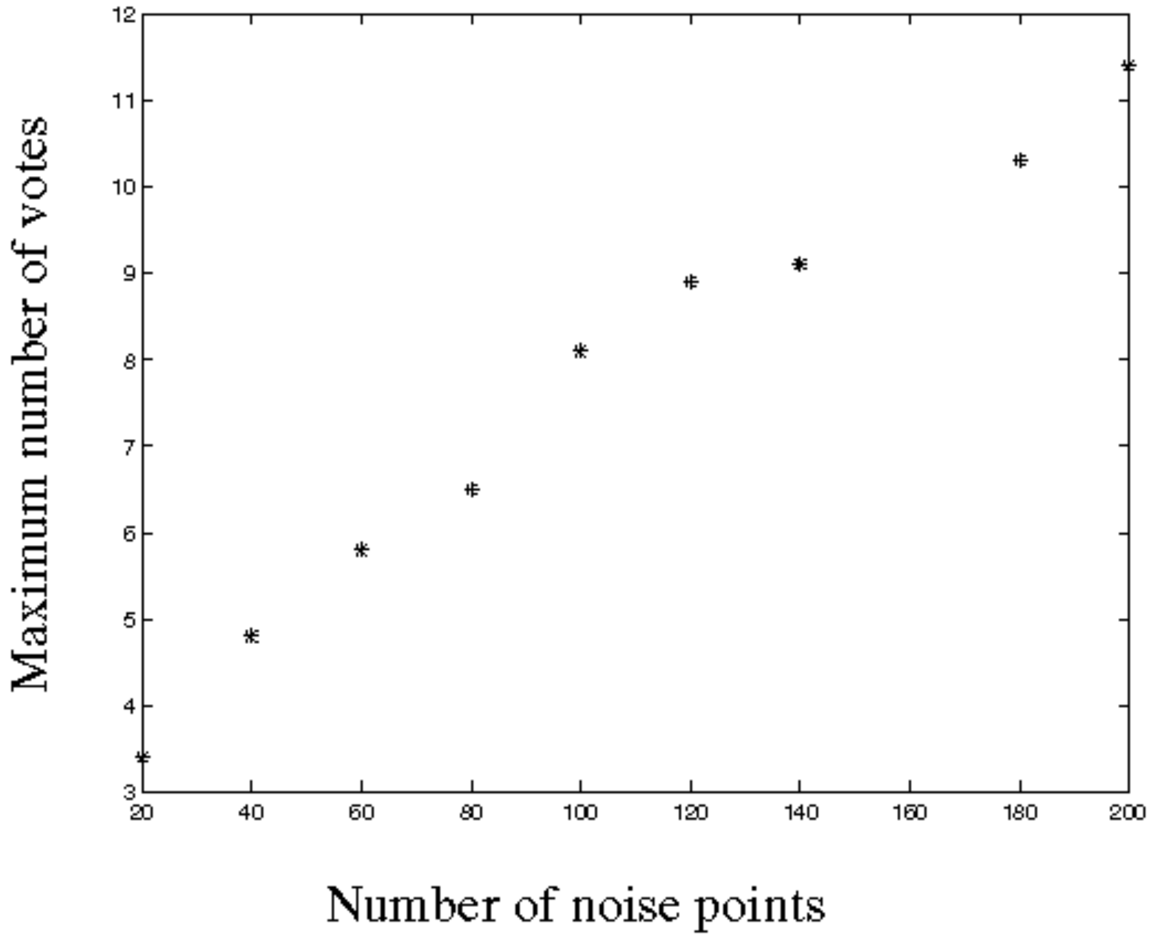
votes

Uniform noise can lead to spurious peaks in the array

Source: S. Lazebnik

Random points

- As the level of uniform noise increases, the maximum number of votes increases too:



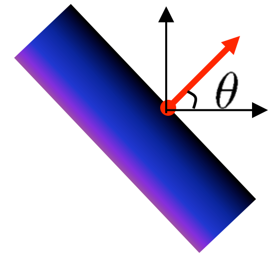
Source: S. Lazebnik

Dealing with noise

- 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

Incorporating image gradients

- Recall: when we detect an edge point, we also know its gradient direction
- But this means that the line is uniquely determined!



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

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

- Modified Hough transform:

For each edge point (x,y)
  θ = gradient orientation at (x,y)
  ρ = x cos θ + y sin θ
  H(θ, ρ) = H(θ, ρ) + 1
end

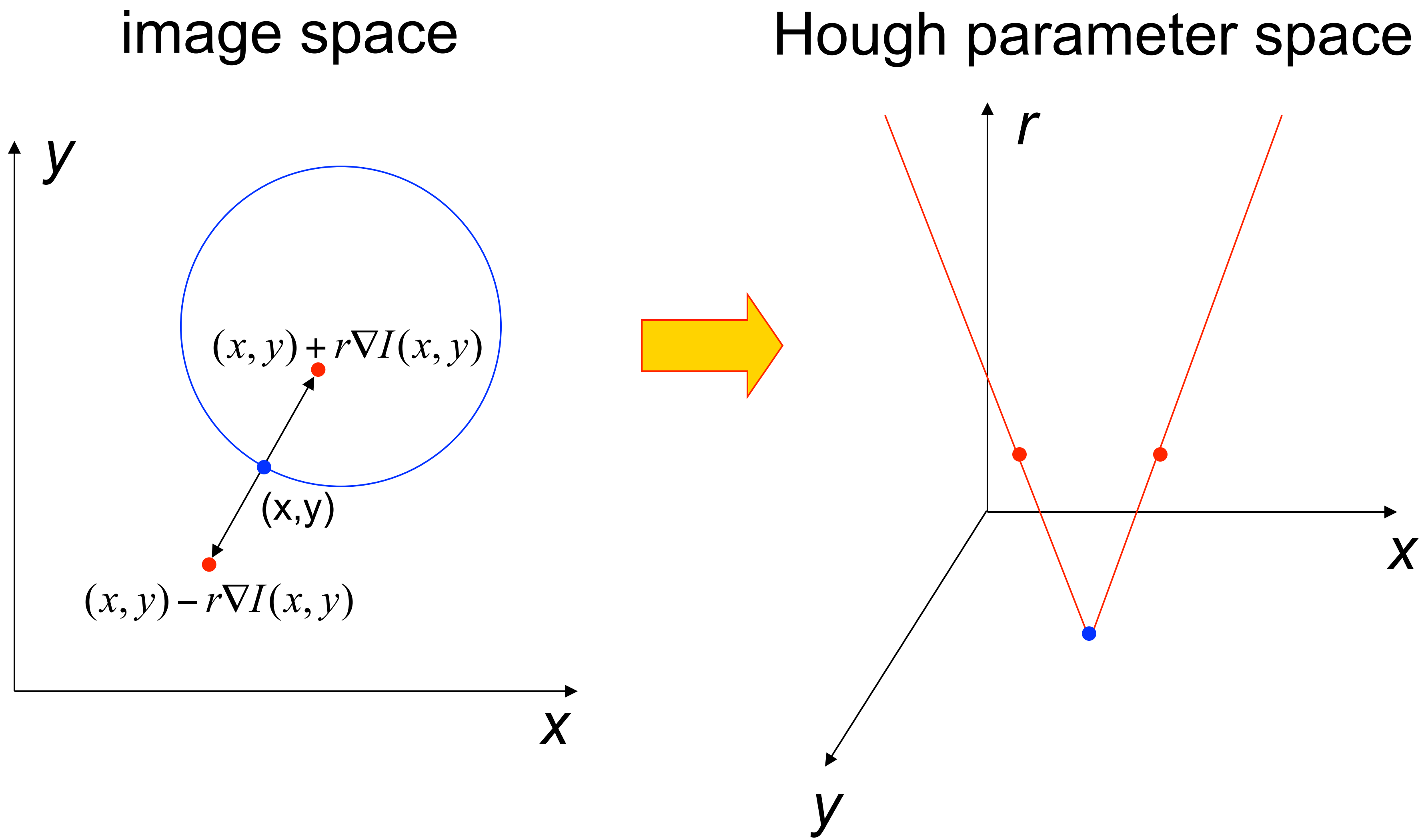
Source: S. Lazebnik

Hough transform for circles

- How many dimensions will the parameter space have?
- Given an unoriented edge point, what are all possible bins that it can vote for?
- What about an *oriented* edge point?

Source: S. Lazebnik

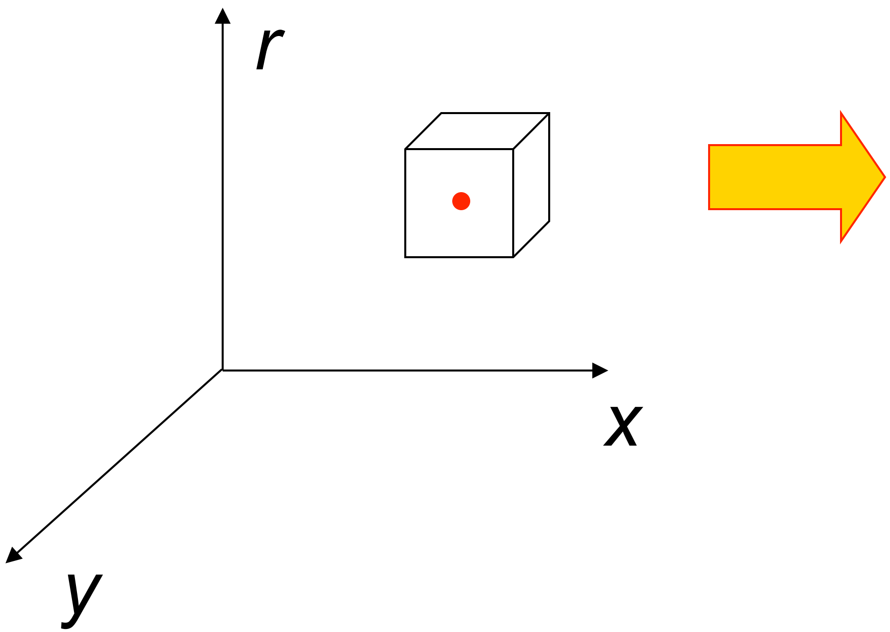
Hough transform for circles

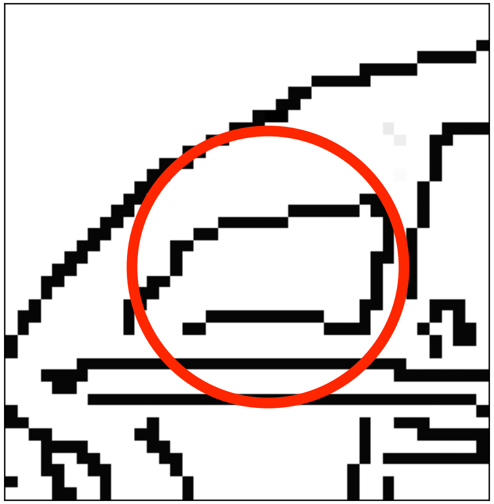


Source: S. Lazebnik

Hough transform for circles

- Conceptually equivalent procedure: for each (x,y,r), draw the corresponding circle in the image and compute its "support"





Is this more or less efficient than voting with features?

Source: S. Lazebnik

Review: Hough transform

- Hough transform for lines
- Hough transform for circles
- Hough transform pros and cons

Source: S. Lazebnik

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
- Cons
  - Complexity of search time increases exponentially with the number of model parameters
  - Non-target shapes can produce spurious peaks in parameter space
  - It’s hard to pick a good grid size