Computer Vision

CS308

Feng Zheng

SUSTech CS Vision Intelligence and Perception

Week 6

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

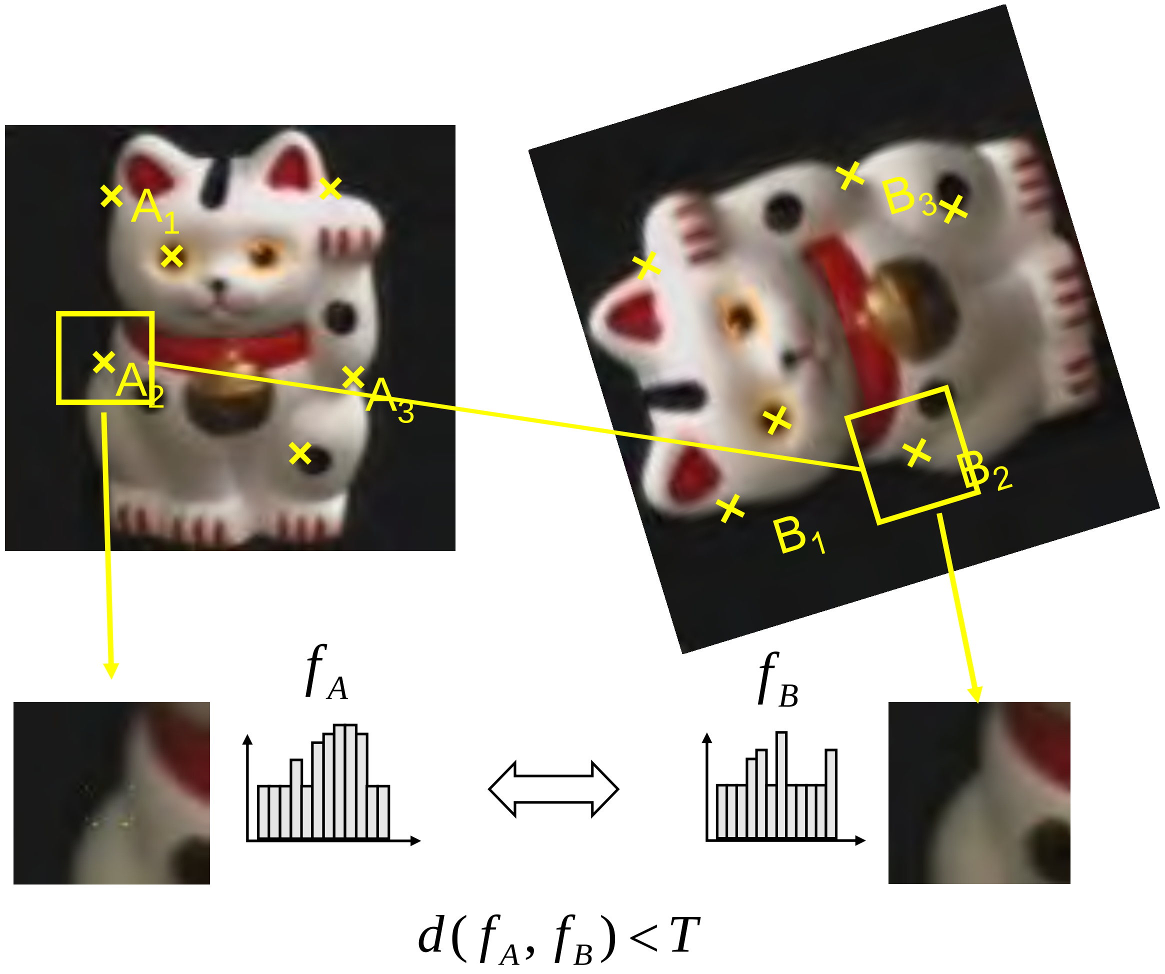
Content

- Brief Review
- Fitting Techniques
  - ➢ Least Squares
  - ➢ Total Least Squares
- Random Sample Consensus (RANSAC)
- Hough Voting
- Image Alignment

Brief Review

Overview of Keypoint Matching

- Steps
  - Find a set of distinctive keypoints
  - Define a region around each keypoint
  - Compute a local descriptor from the region
  - Match local descriptors
- Goals
  - Detect points that are repeatable and distinctive



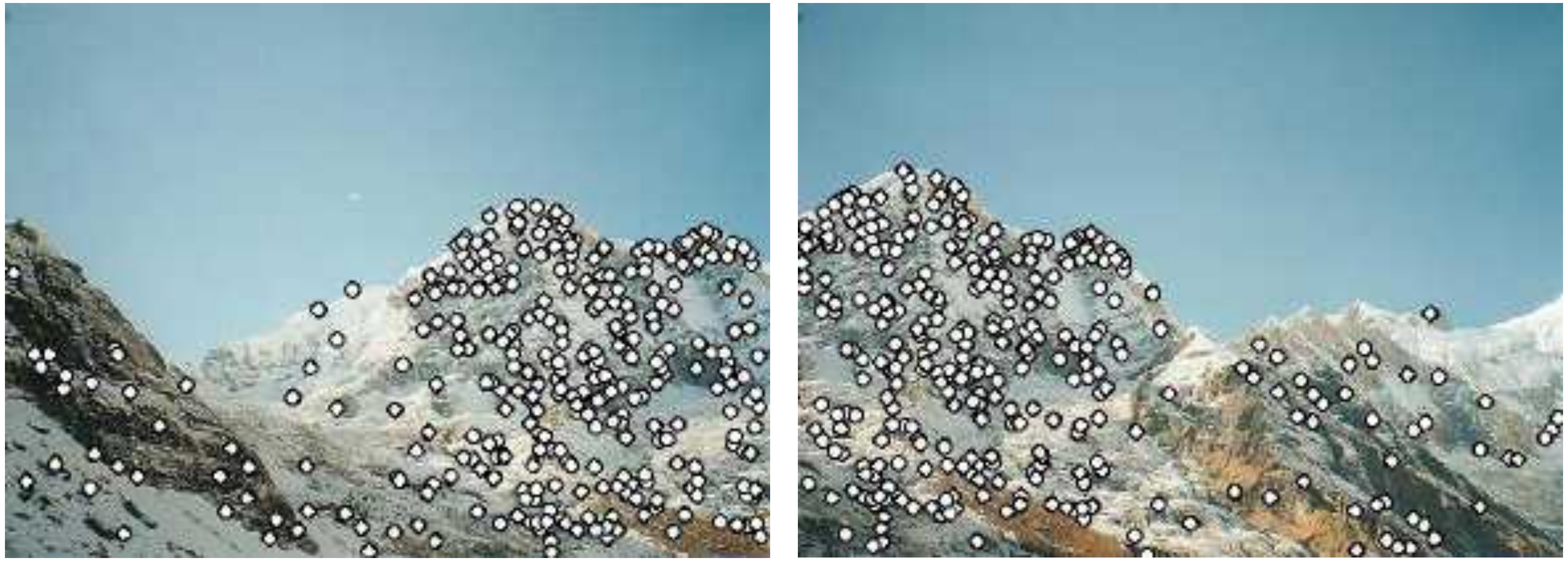
Fitting Techniques

How Do We Build Panorama?

- We need to match (align) images

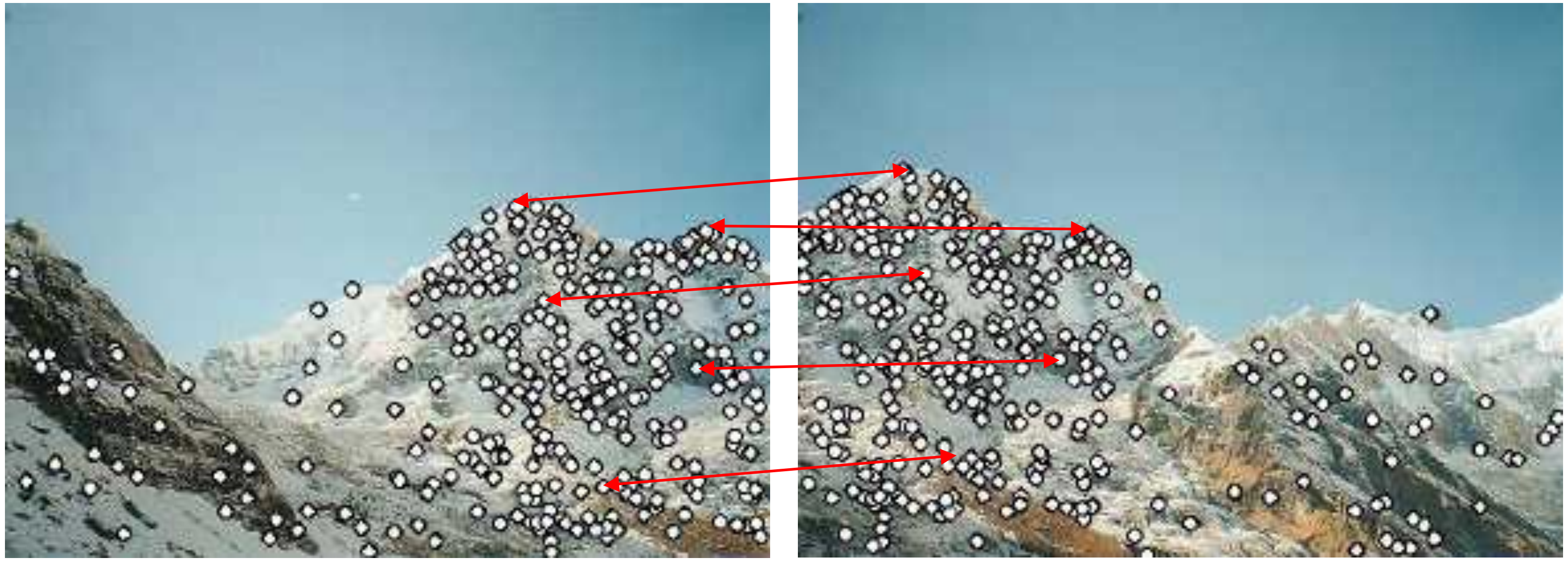
Matching with Features

- Steps
  - Detect feature points in both images



Matching with Features

- Steps
  - Detect feature points in both images
  - Find corresponding pairs



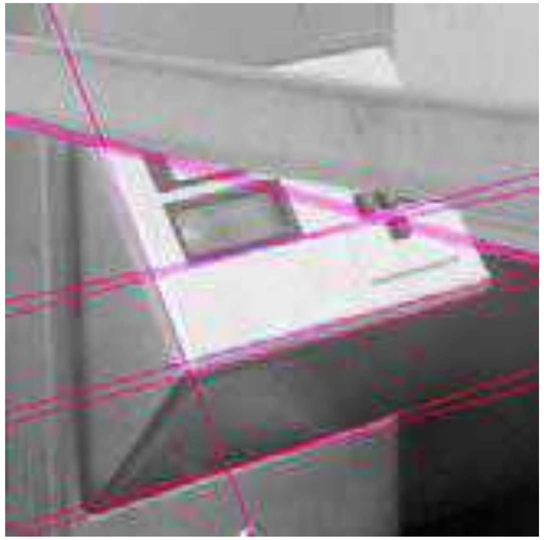
Matching with Features

- Steps
  - Detect feature points in both images
  - Find corresponding pairs
  - Use these pairs to align images

Previous Lecture

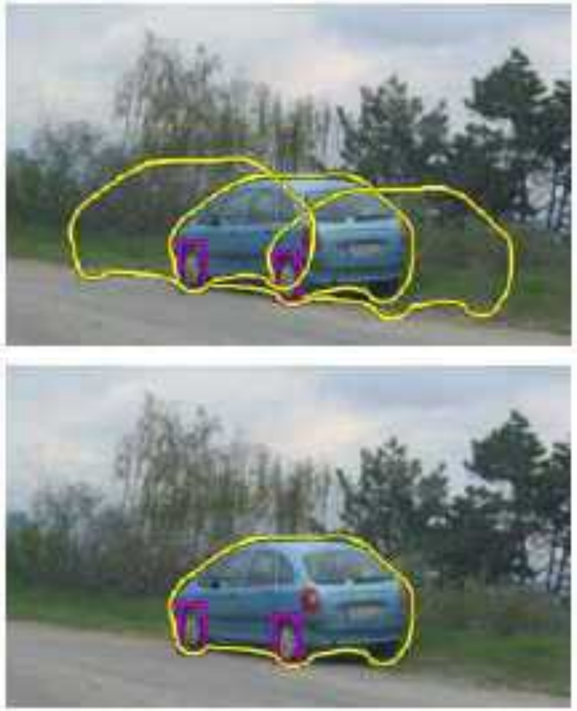
Fitting: Building a Model for a Set of Features

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



Simple model: lines

Simple model: circles



Complicated model: car

Fitting: Issues

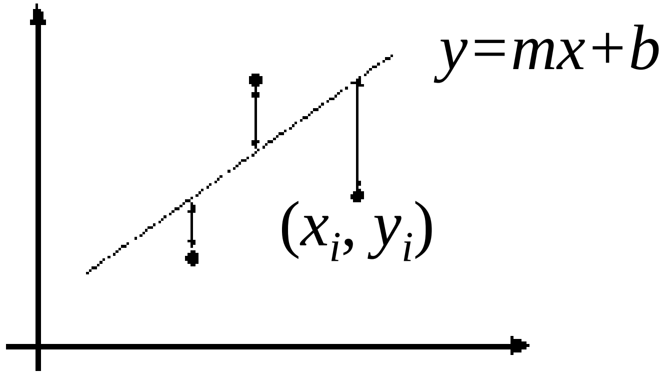
- Case study: Line detection
  - Noise in the measured feature locations
  - Extraneous data: clutter (outliers), multiple lines
  - Missing data: occlusions

Fitting: Issues

- 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

Line Fitting: Ordinary Least Squares

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

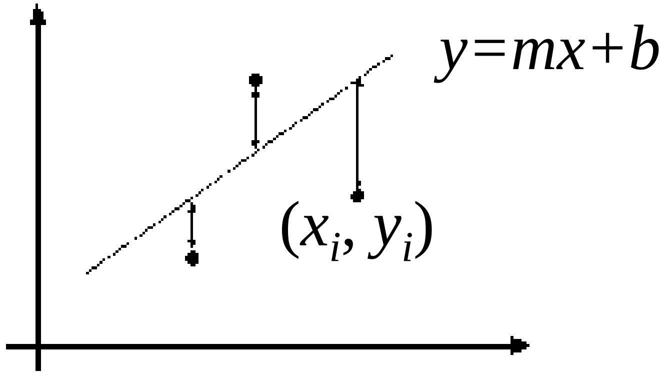


$$E = \sum_{i=1}^{n} (y_i - m x_i - b)^2$$

We know which points belong to the line

Line Fitting: Ordinary Least Squares

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



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

$$E = \sum_{i=1}^{n}\left(y_i - \begin{bmatrix} x_i & 1 \end{bmatrix}\begin{bmatrix} m \\ b \end{bmatrix}\right)^2 = \left\| \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}\begin{bmatrix} m \\ b \end{bmatrix} \right\|^2 = \|Y - XB\|^2$$

$$= (Y - XB)^T (Y - XB) = Y^T Y - 2(XB)^T Y + (XB)^T (XB)$$

Line Fitting: Ordinary Least Squares

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

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

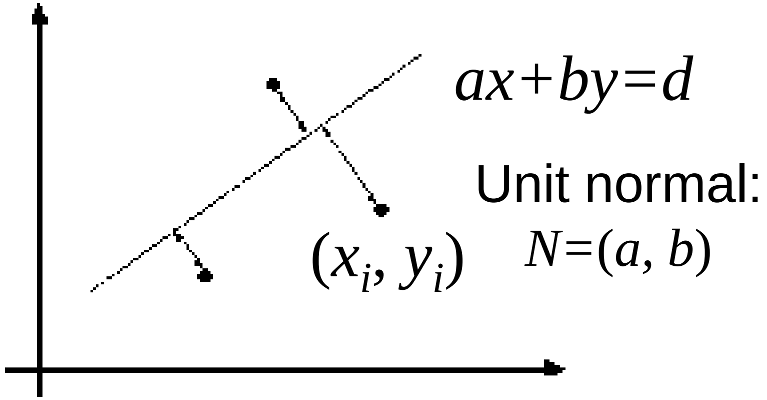
- Problem with "vertical" least squares
  - Not rotation-invariant
  - Fails completely for vertical lines $X^T X$

Total Least Squares

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

$$|ax_i + by_i - d|$$

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



Total Least Squares

$$\frac{\partial E}{\partial d} = \sum_{i=1}^{n} -2(ax_i + by_i - d) = 0 \Rightarrow d = \frac{a}{n}\sum_{i=1}^{n} x_i + \frac{b}{n}\sum_{i=1}^{n} x_i = a\bar{x} + b\bar{y}$$

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

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

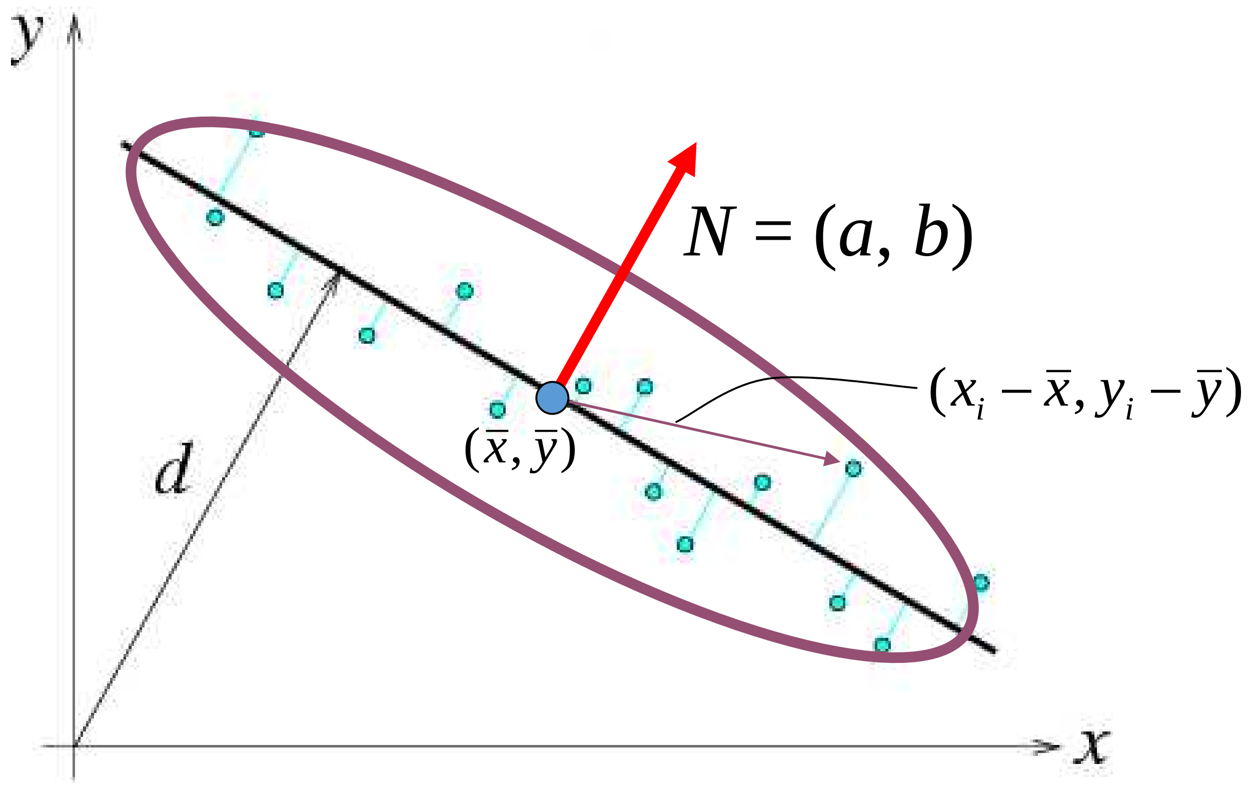
- Solution to $(U^T U)N = 0$, subject to $\|N\|^2 = 1$: eigenvector of $U^T U$ associated with the smallest eigenvalue (least squares solution to *homogeneous linear system* $UN = 0$)

Total Least Squares

- Second moment matrix

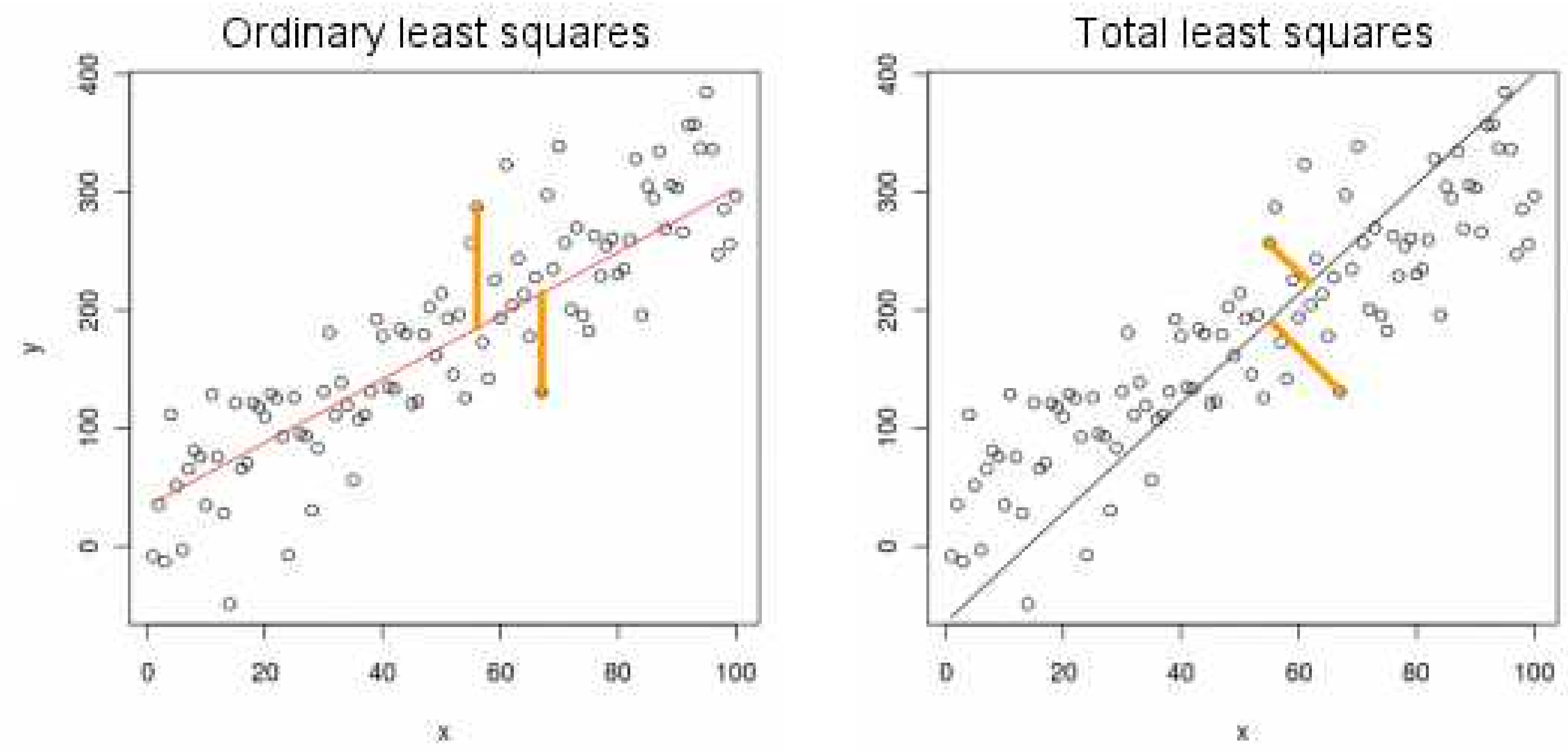
$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}$$

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



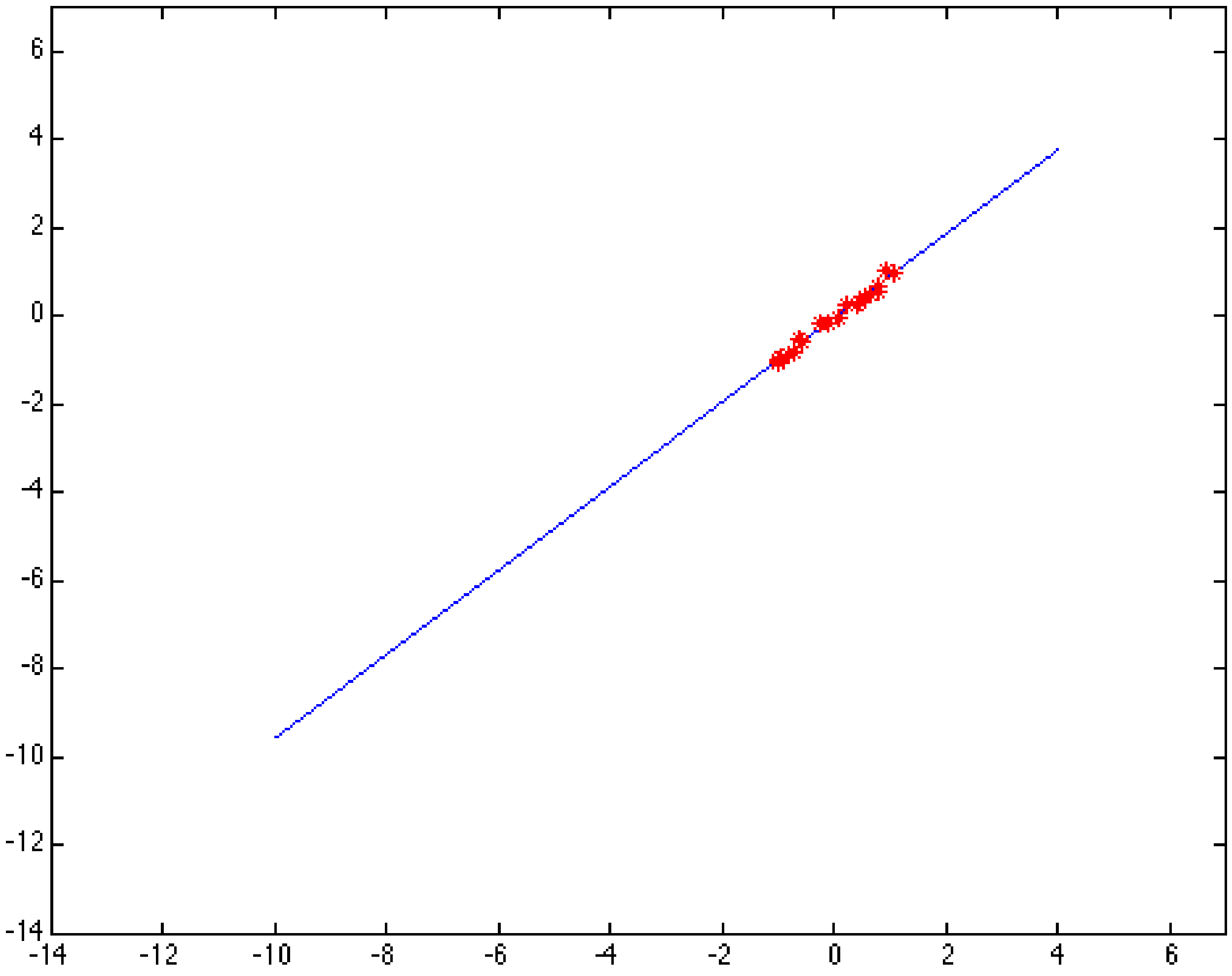
OLS vs. TLS

- The difference between standard OLS regression and "orthogonal" TLS regression



Total Least Squares

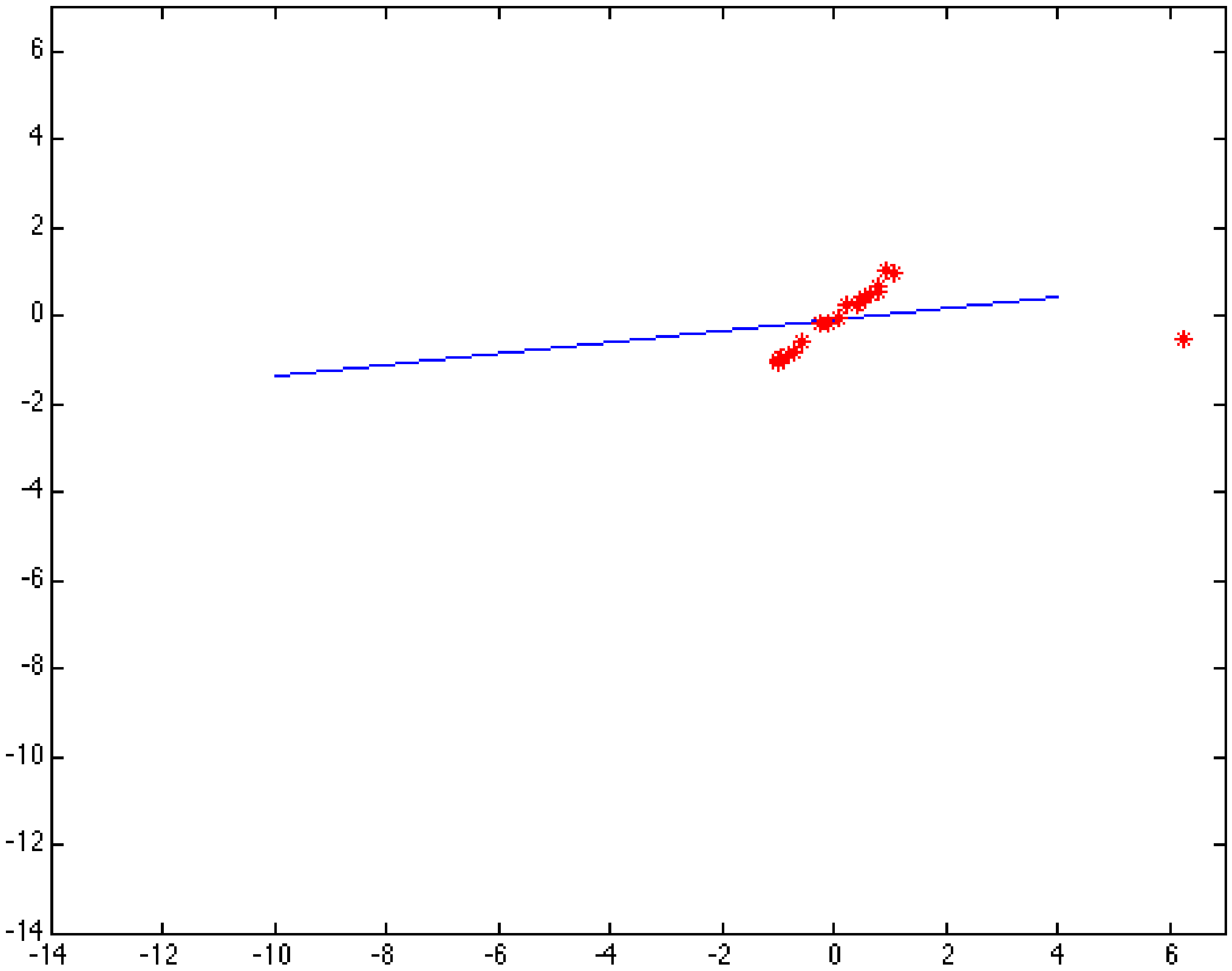
- Robustness to noise: least squares fit to the red points



Total Least Squares

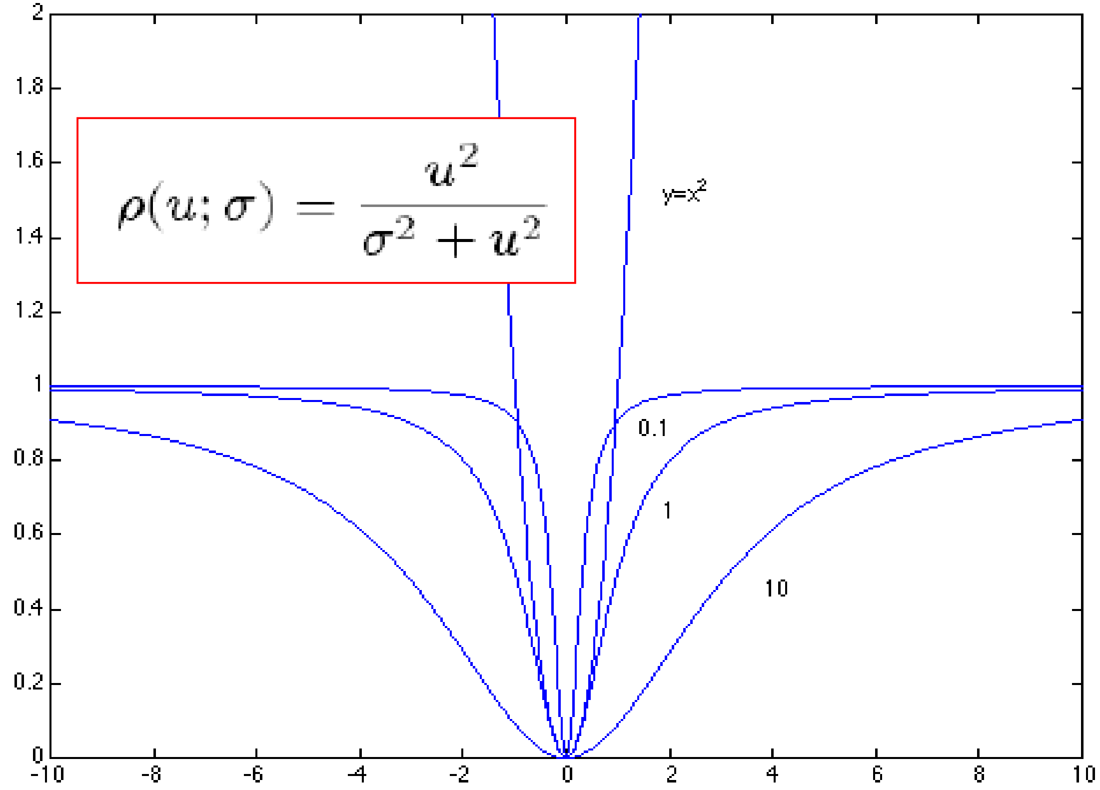
- Robustness to noise: Least squares fit with an outlier

- Problem: squared error heavily penalizes outliers



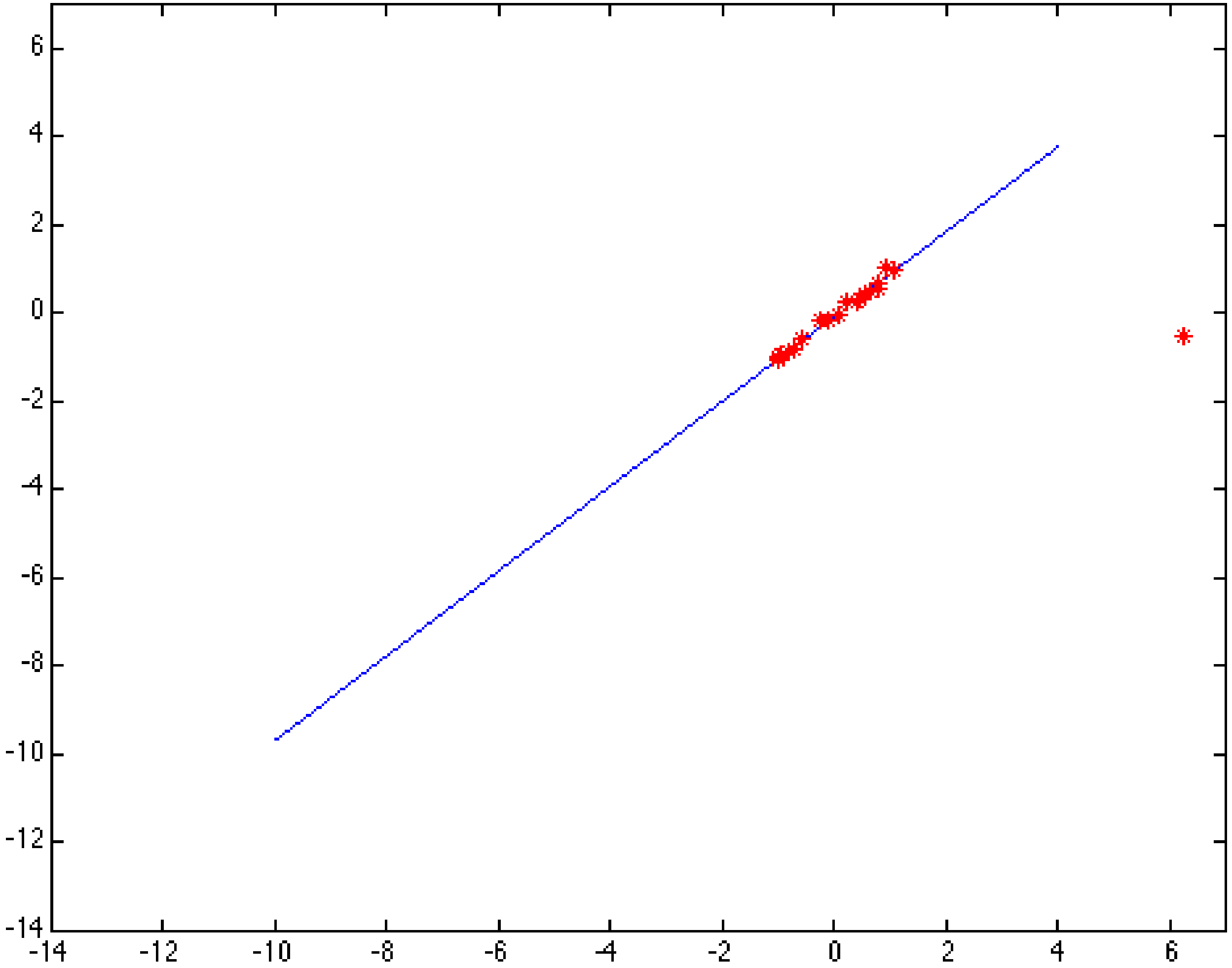
Robust Estimators

- General approach---minimize: $\sum_i \rho(r_i(x_i, \theta); \sigma)$
  $r_i(x_i, \theta)$ – residual of ith point w.r.t. model parameters θ
  ρ – robust function with scale parameter σ
- The robust function ρ behaves like squared distance for small values of the residual u but saturates for larger values of u



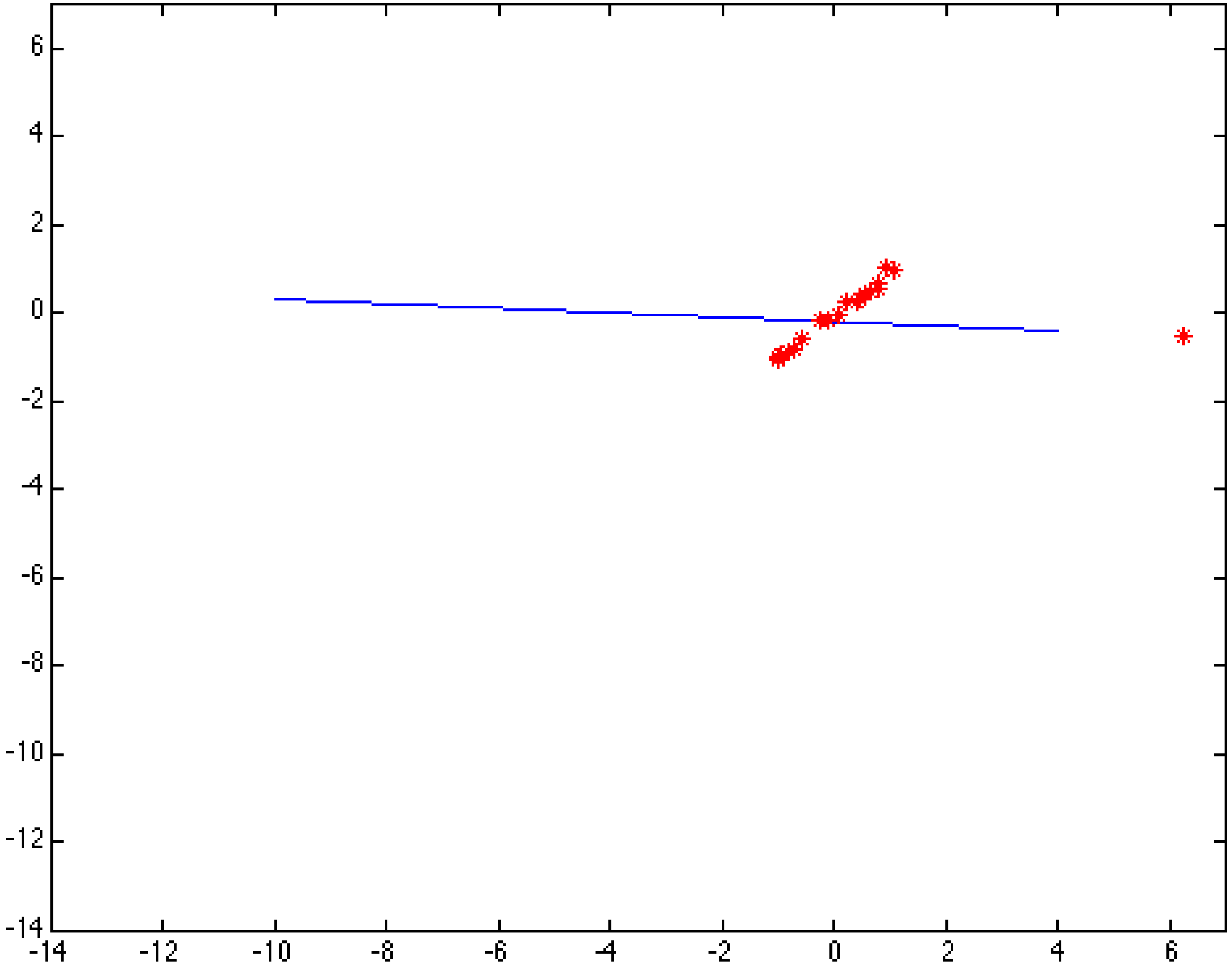
Choosing the Scale: Just Right

- The effect of the outlier is minimized, when choosing a just right scale



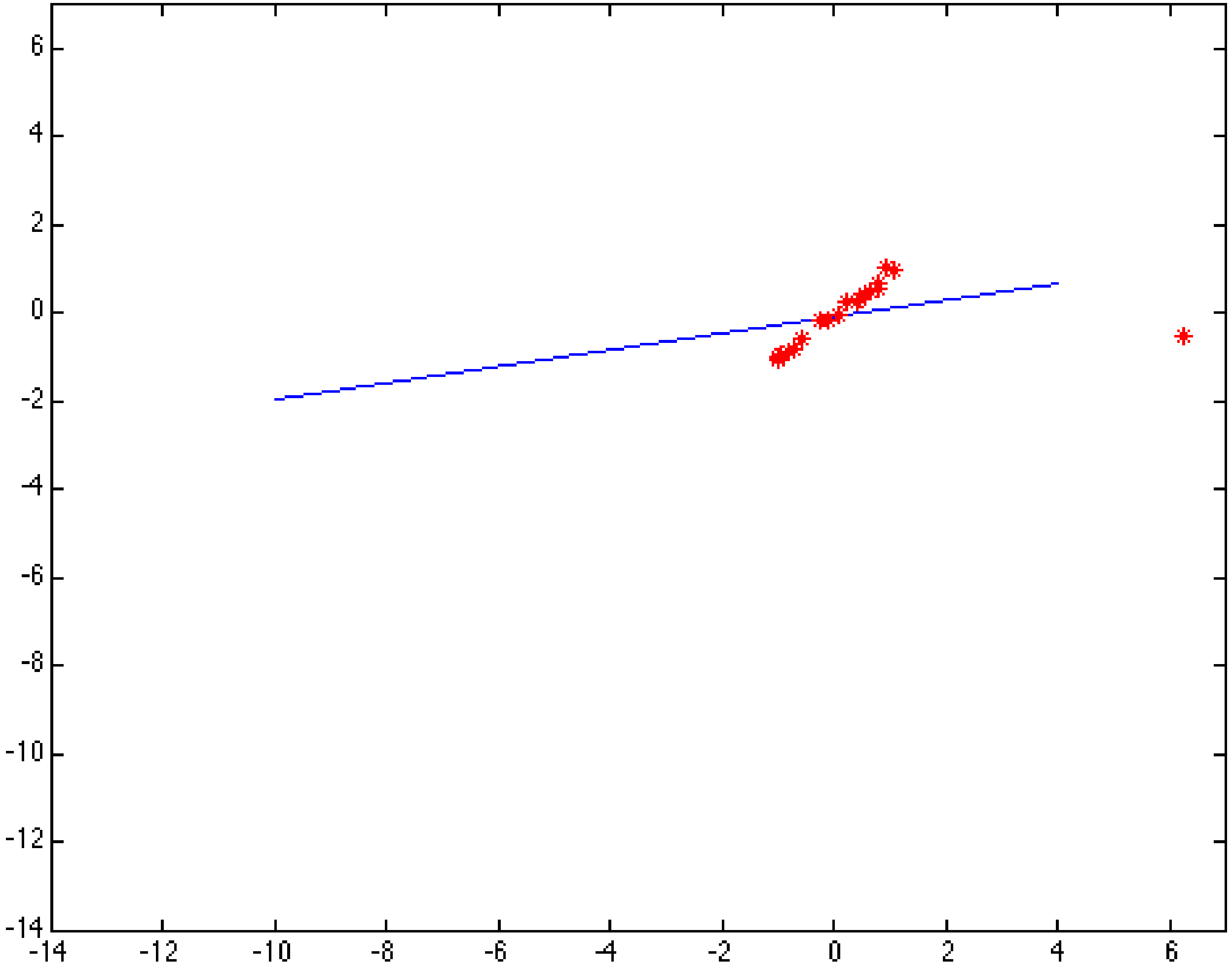
Choosing the Scale: Too Small

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



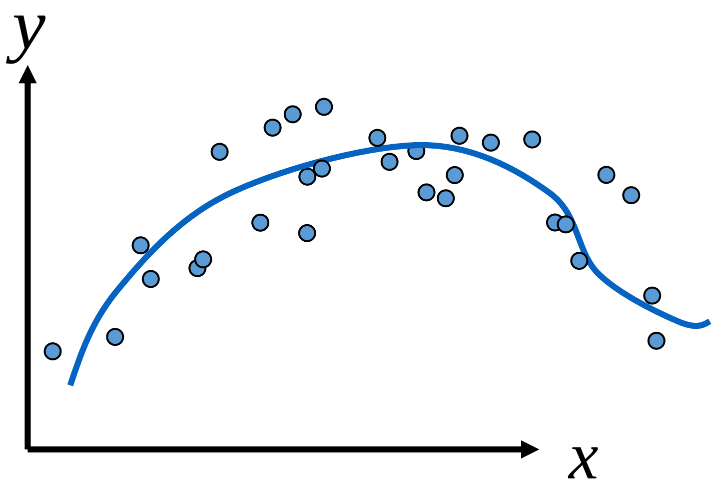
Choosing the Scale: Too Large

- Behaves much the same as least squares



Curve Fitting

- Find Polynomial: $y = f(x) = ax^3 + bx^2 + cx + d$
  - ➢ That best fits the given points (x_i, y_i)



- Minimize: $\frac{1}{N}\sum_i [y_i - (ax_i^3 + bx_i^2 + cx_i + d)]^2$

- Using: $\frac{\partial E}{\partial a} = 0 \;, \quad \frac{\partial E}{\partial b} = 0 \;, \quad \frac{\partial E}{\partial c} = 0 \;, \quad \frac{\partial E}{\partial d} = 0$

- Note: $f(x)$ is LINEAR in the parameters (a, b, c, d)

Random Sample Consensus

RANSAC

- Robust fitting (TLS) 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

- Algorithm
- Repeat N times:
  - Draw s points uniformly at random
  - Fit line to these s points (TLS)
  - 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
- End
- Four parameters: s, t, d and N

Choosing the Parameters

- Initial number of points s
  - ➢ Minimum number needed to fit the model
    - ✓ 2 points

- Distance threshold t
  - ➢ (1) Choose t so probability for inlier is p (e.g. 0.95)
  - ➢ (2) Zero-mean Gaussian noise with standard deviation σ: $t^2 = 3.84\sigma^2$

Choosing the Parameters

- Number of times N
  - Choose N so that, with probability p, at least one random sample is free from outliers (e.g. $p=0.99$)

Desired success rate after N times: p

Outlier ratio (Unknown): e

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

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

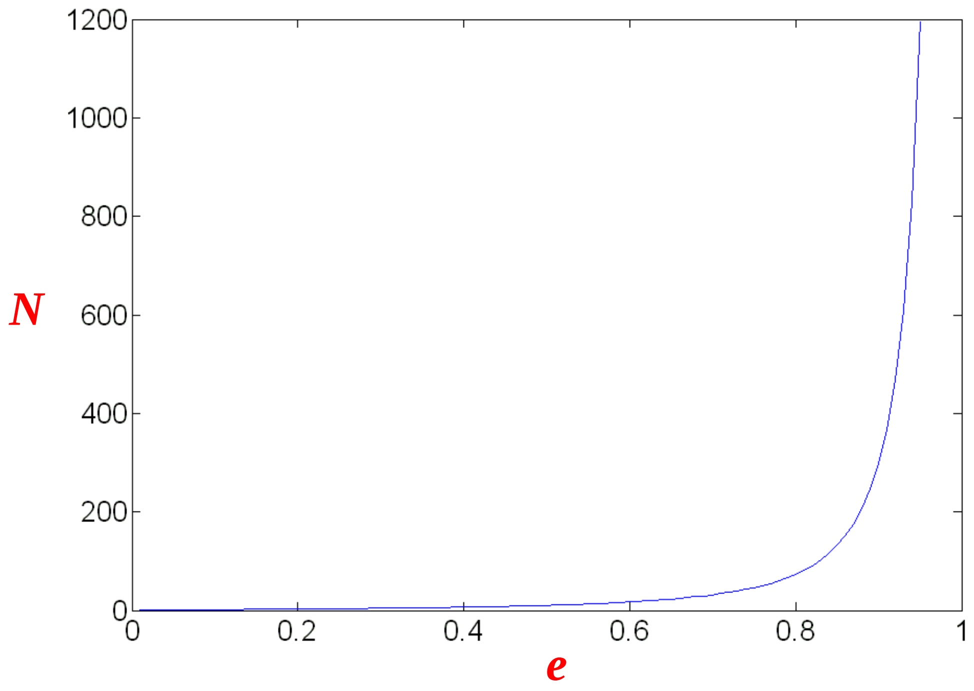
N	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Choosing the Parameters

- Consensus set size d (number of inliers)
  - ➢ Should match expected inlier ratio

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

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



Adaptively determining the number of samples

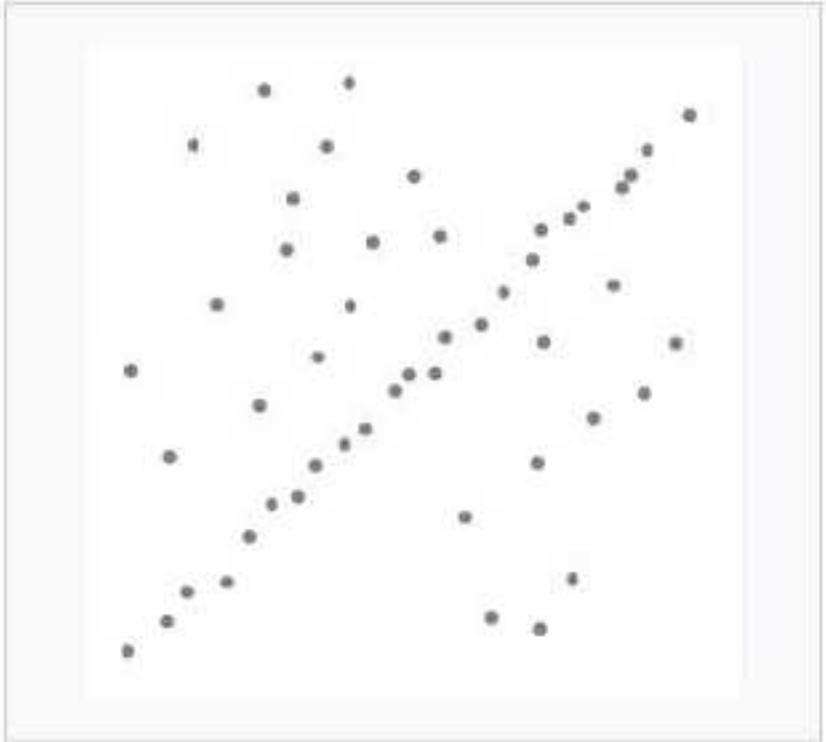
- Inlier ratio *e* is often unknown a priori, so pick worst case, e.g. 50%, and adapt if more inliers are found, e.g. 80% would yield $e=0.2$
- Adaptive procedure:
  - $N=\infty$, $sample\_count=0$
  - While $N > sample\_count$
    - ✓ Choose a sample (fitting) and count the number of inliers
    - ✓ Set e = 1 – (number of inliers)/(total number of points)
    - ✓ Recompute N from e:

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

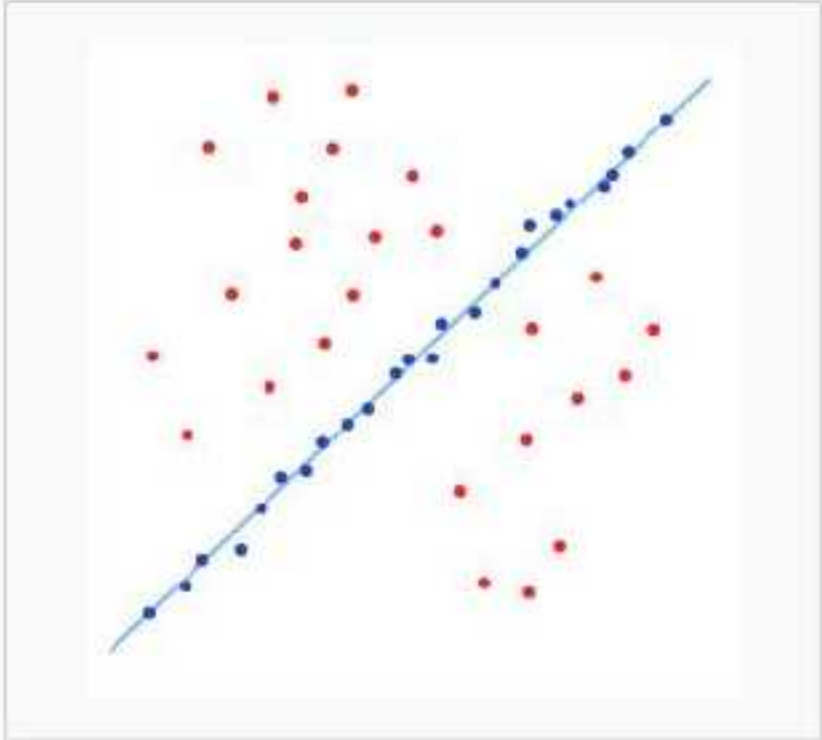
    - ✓ Increment the *sample_count* by 1

RANSAC

- An example



A data set with many outliers for which a line has to be fitted.



Fitted line with RANSAC; outliers have no influence on the result.

RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Can't always get a good initialization of the model based on the minimum number of samples
  - Sometimes too many iterations are required
  - Can fail for extremely low inlier ratios
  - We can often do better than brute-force sampling

Hough transform

Voting Schemes

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

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

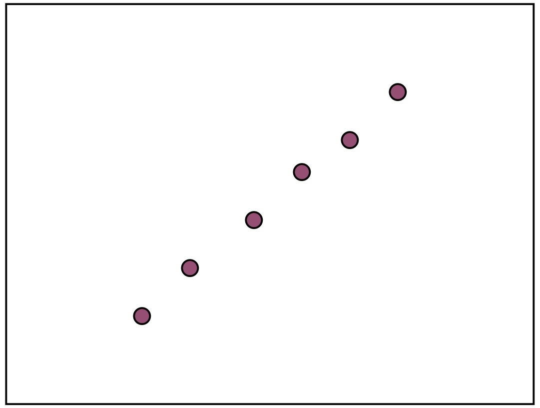
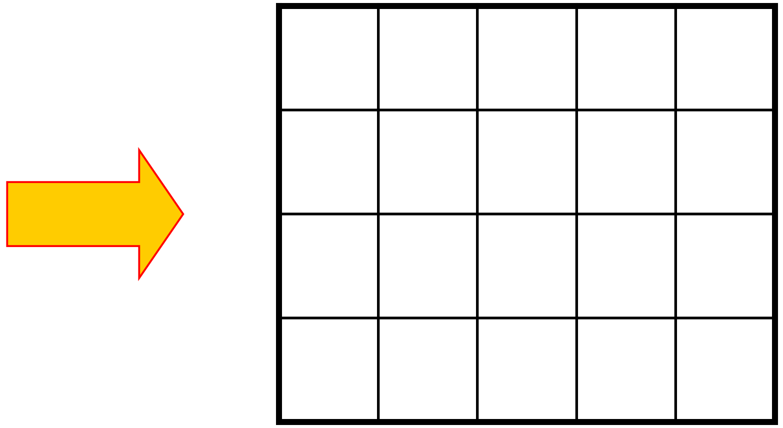


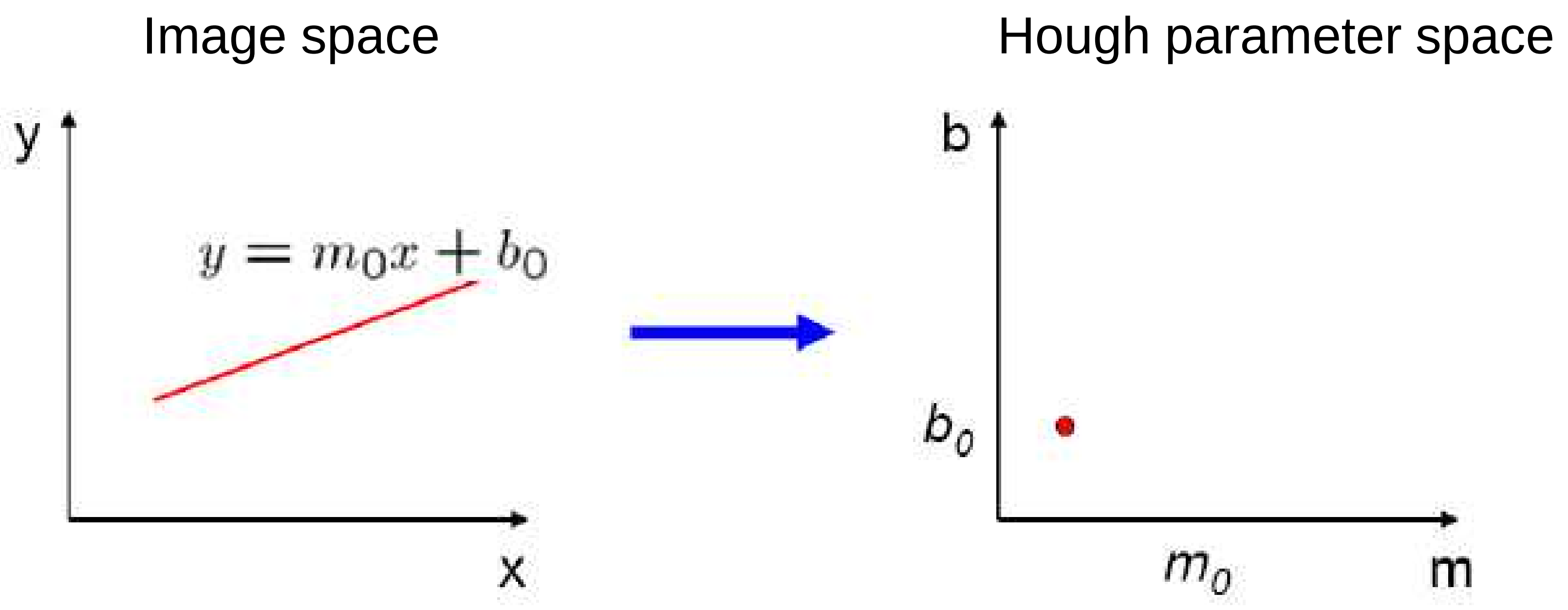
Image space



Hough parameter space

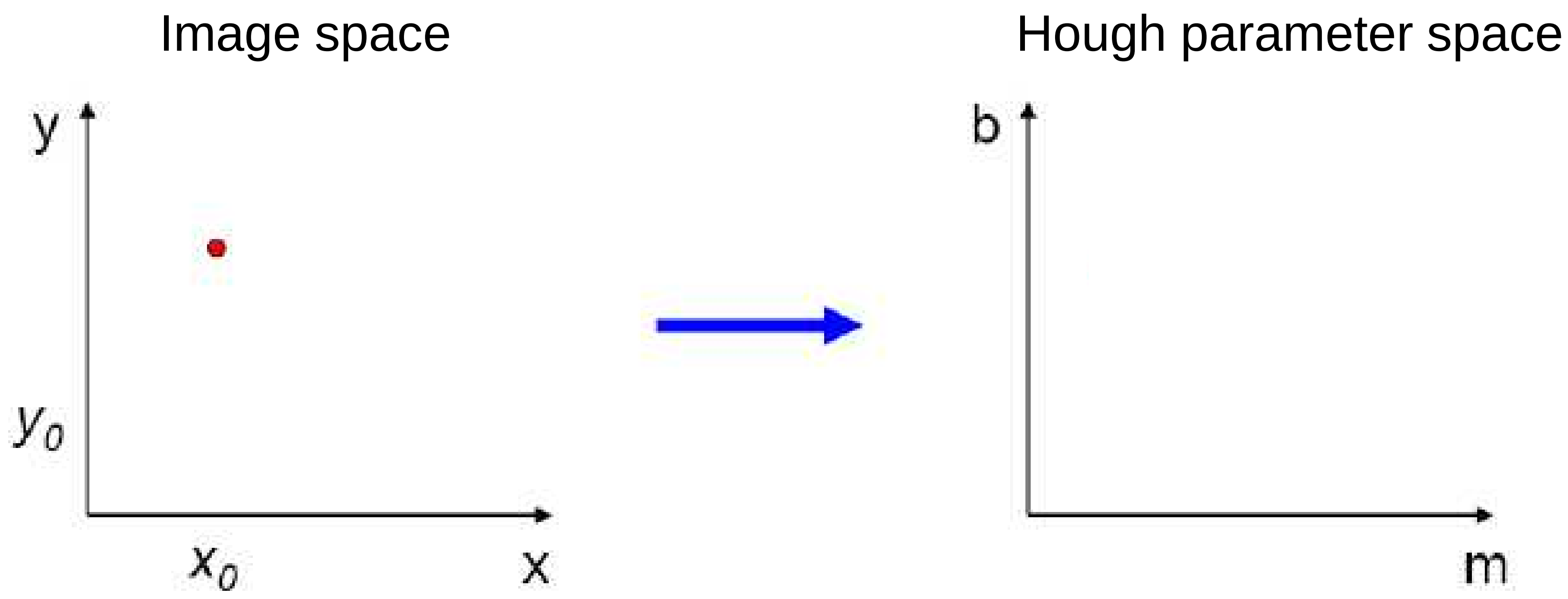
Parameter Space Representation

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



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

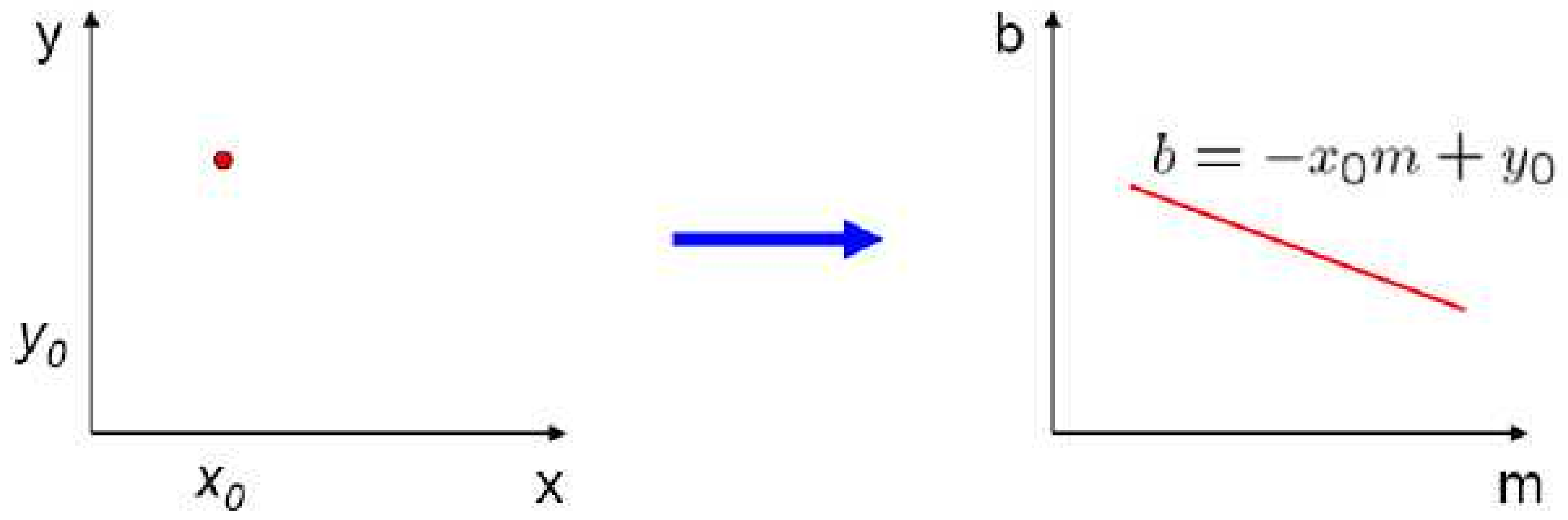
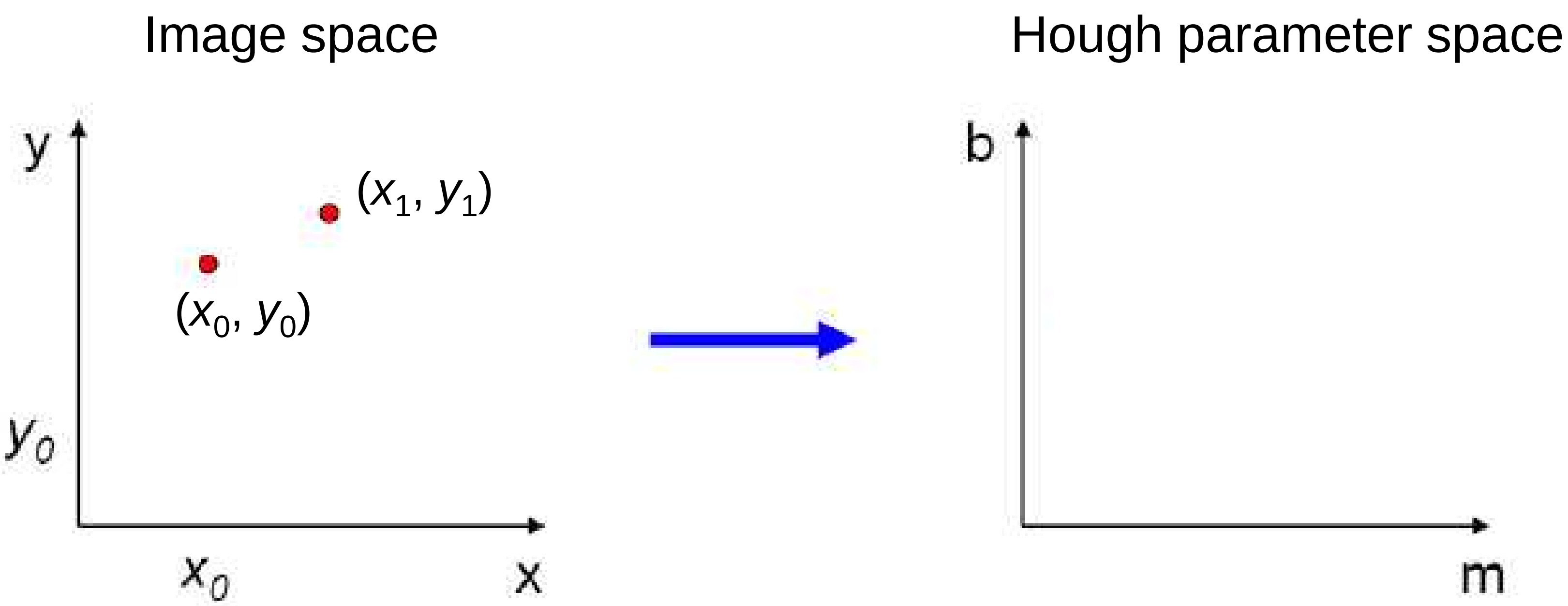


Image space

Hough parameter space

Parameter Space Representation

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

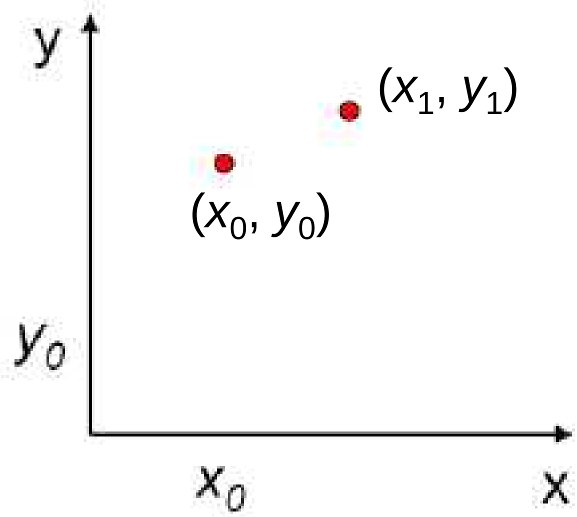


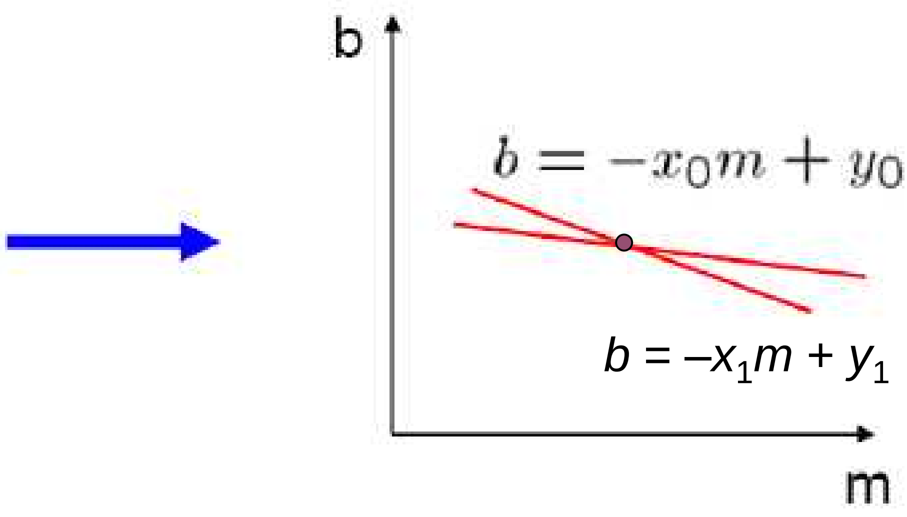
Parameter Space Representation

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

Image space

Hough parameter space

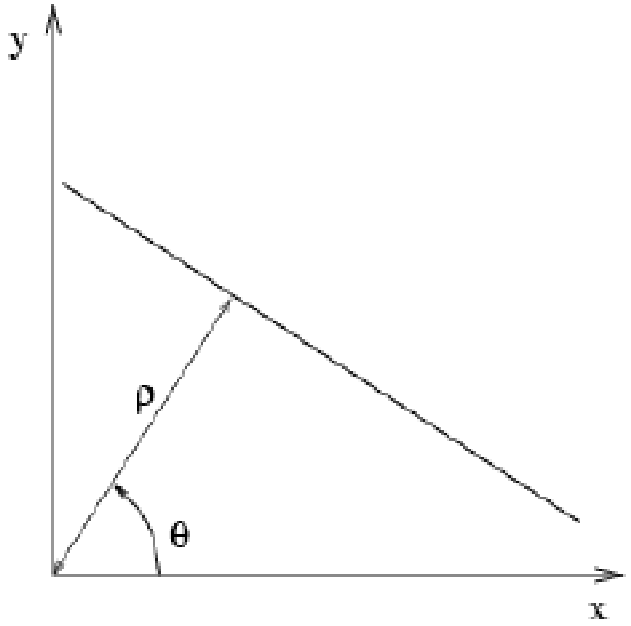




Parameter Space Representation

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

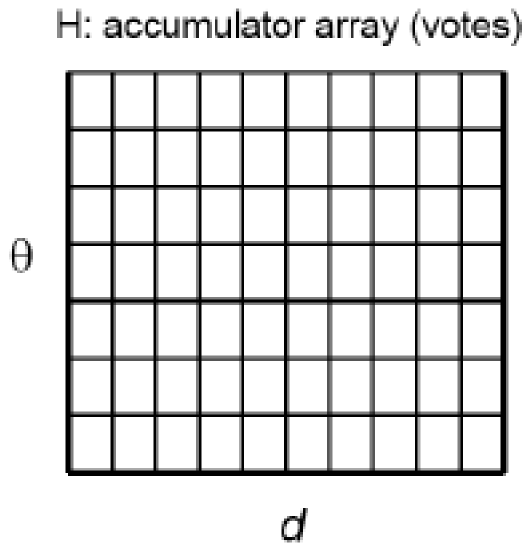
$$x\cos\theta + y\sin\theta = \rho$$



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

Algorithm Outline

- Initialize accumulator H to all zeros
- For **each** edge point (x,y) in the image
  For $\theta = 0$ 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
    $\rho = x \cos\theta + y \sin\theta$



Basic Illustration

- A line

Horizontal axis is θ
Vertical is rho.

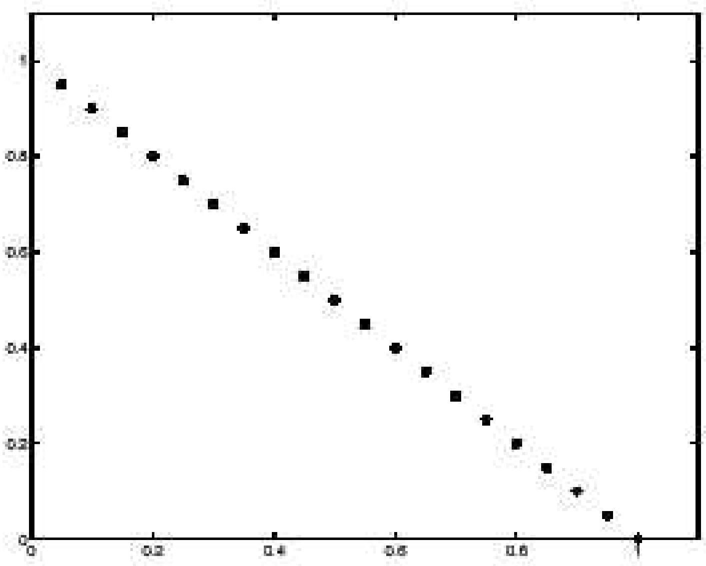
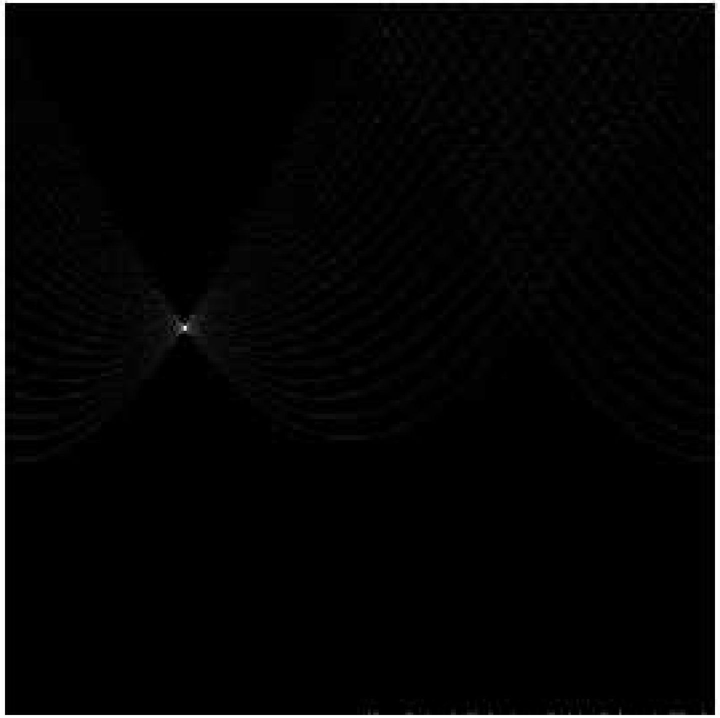


Image space



Votes

Basic Illustration

- A line with noise

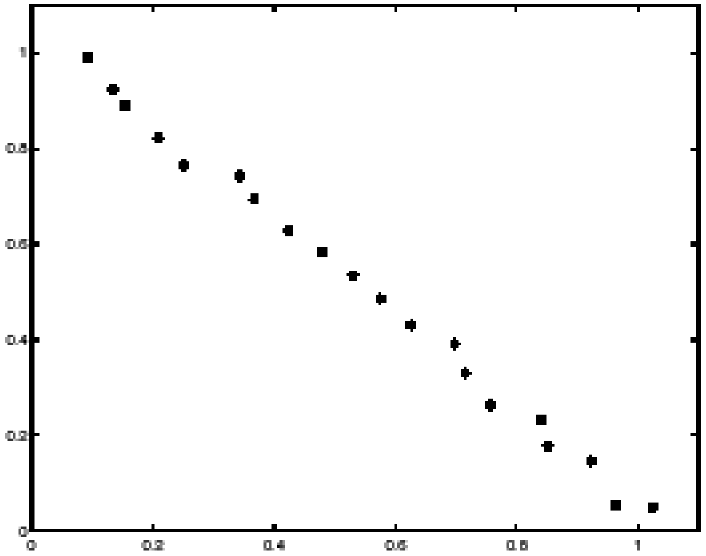
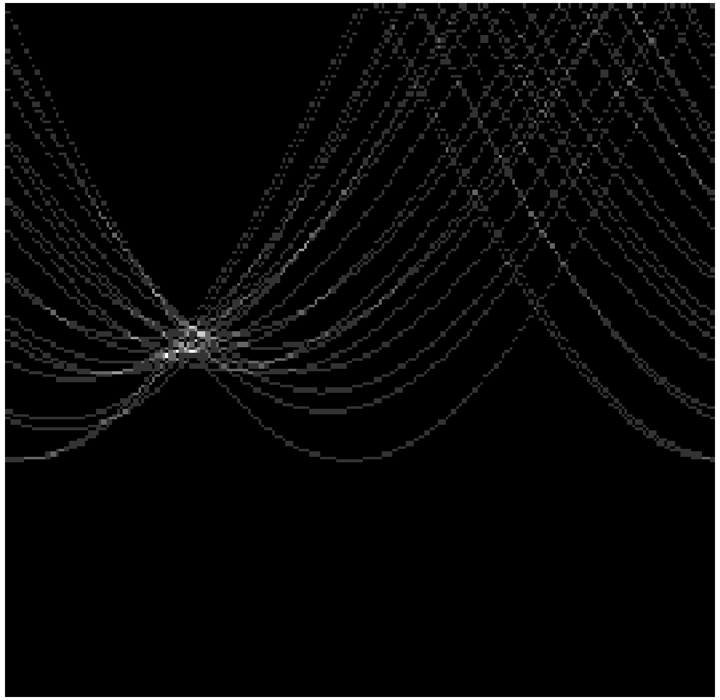


Image space



Votes

Basic Illustration

- Scattered points

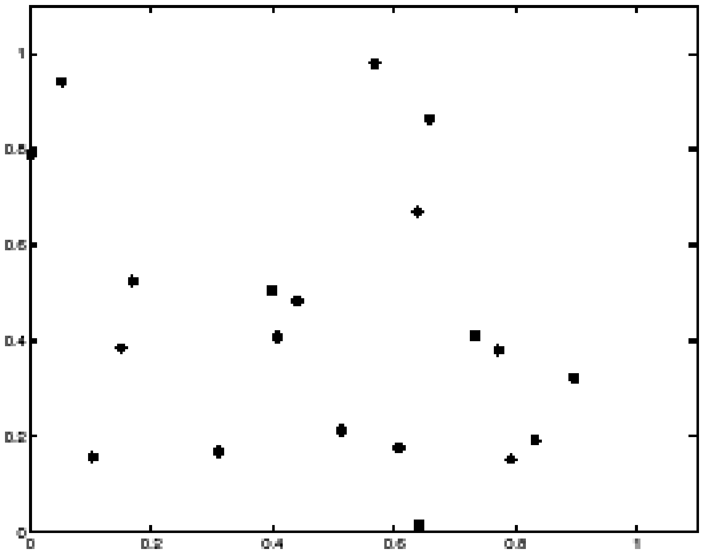
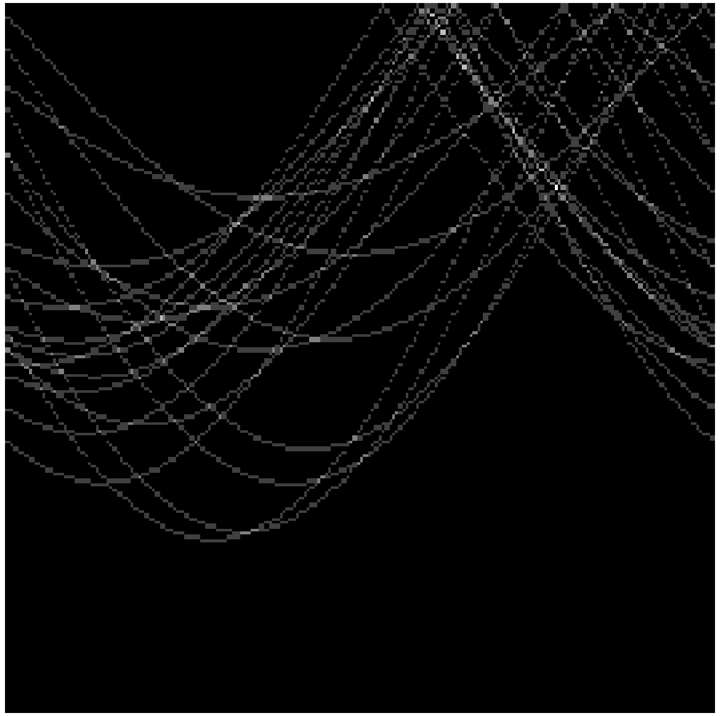


Image space



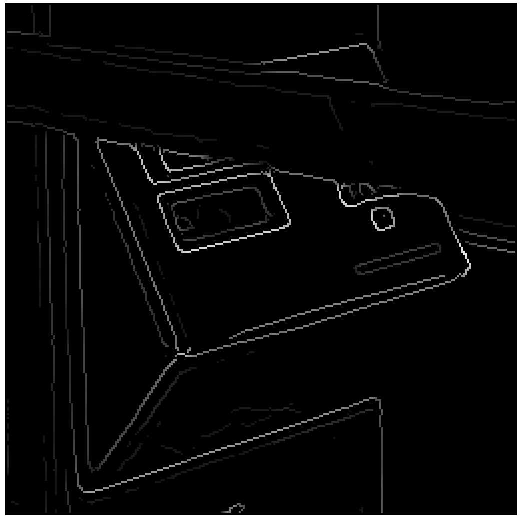
Votes

Mechanics of the Hough transform

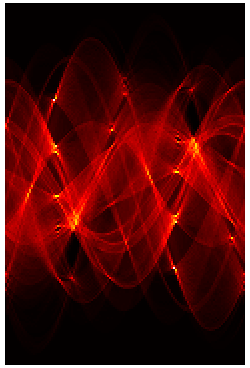
- Difficulties
  - How big should the cells be? (too big, and we merge quite different lines; too small, and noise causes lines to be missed)
- How many lines?
  - Count the peaks in the Hough array
  - Treat adjacent peaks as a single peak
- Which points belong to each line?
  - Search for points close to the line
  - Solve again for line and iterate

Real World Example

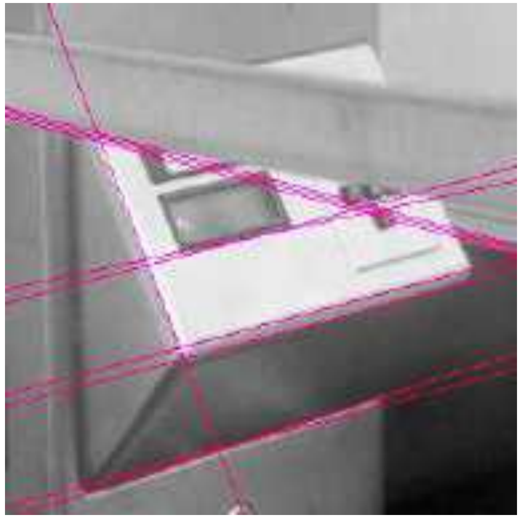
Original



Edge Detection



Parameter Space



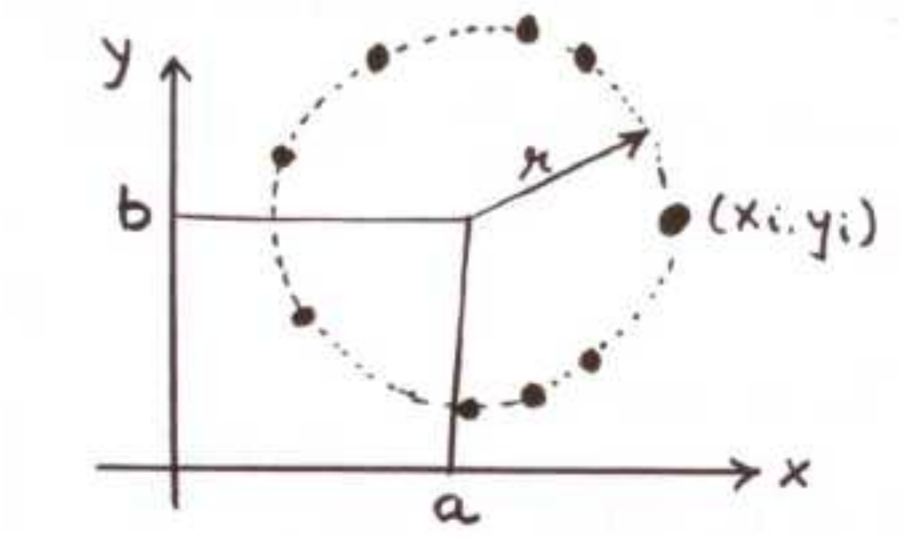
Found Lines

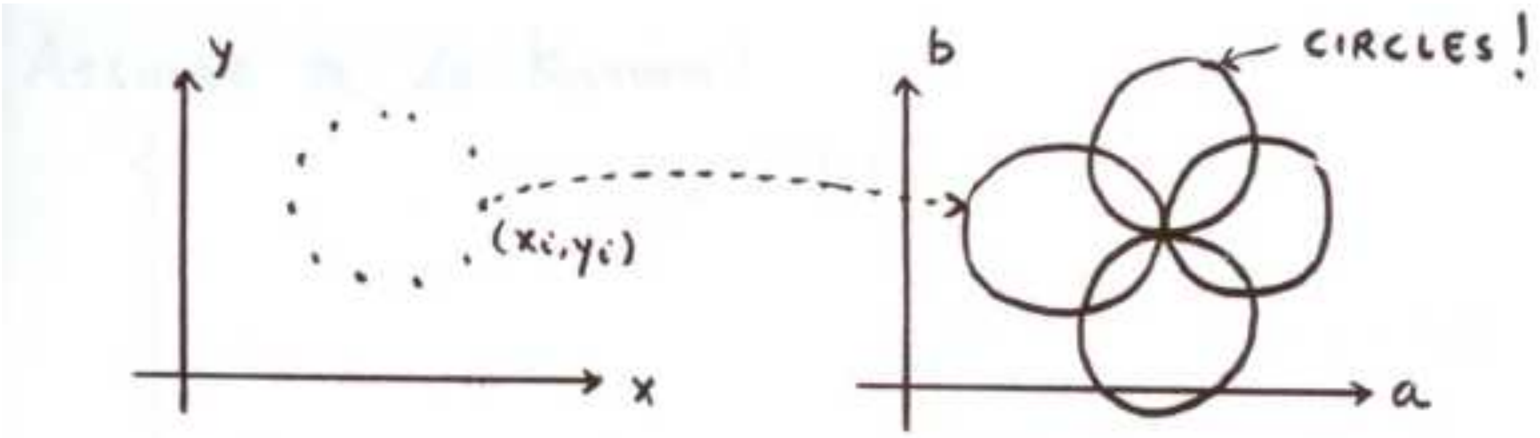
Finding Circles by Hough Transform

- Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- If radius is known:
  - ➢ 2D Hough Space
- Accumulator Array: $A(a,b)$



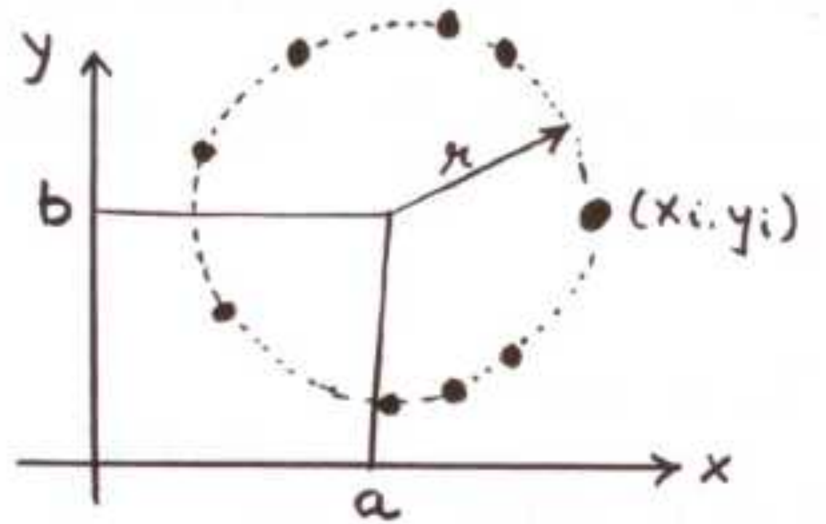


Finding Circles by Hough Transform

- Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- If radius is not known:
    - ➢ 3D Hough space!
- Use Accumulator array: $A(a,b,r)$



- What is the surface in the Hough space?

Finding Circles by Hough Transform

- Hough transform for circles

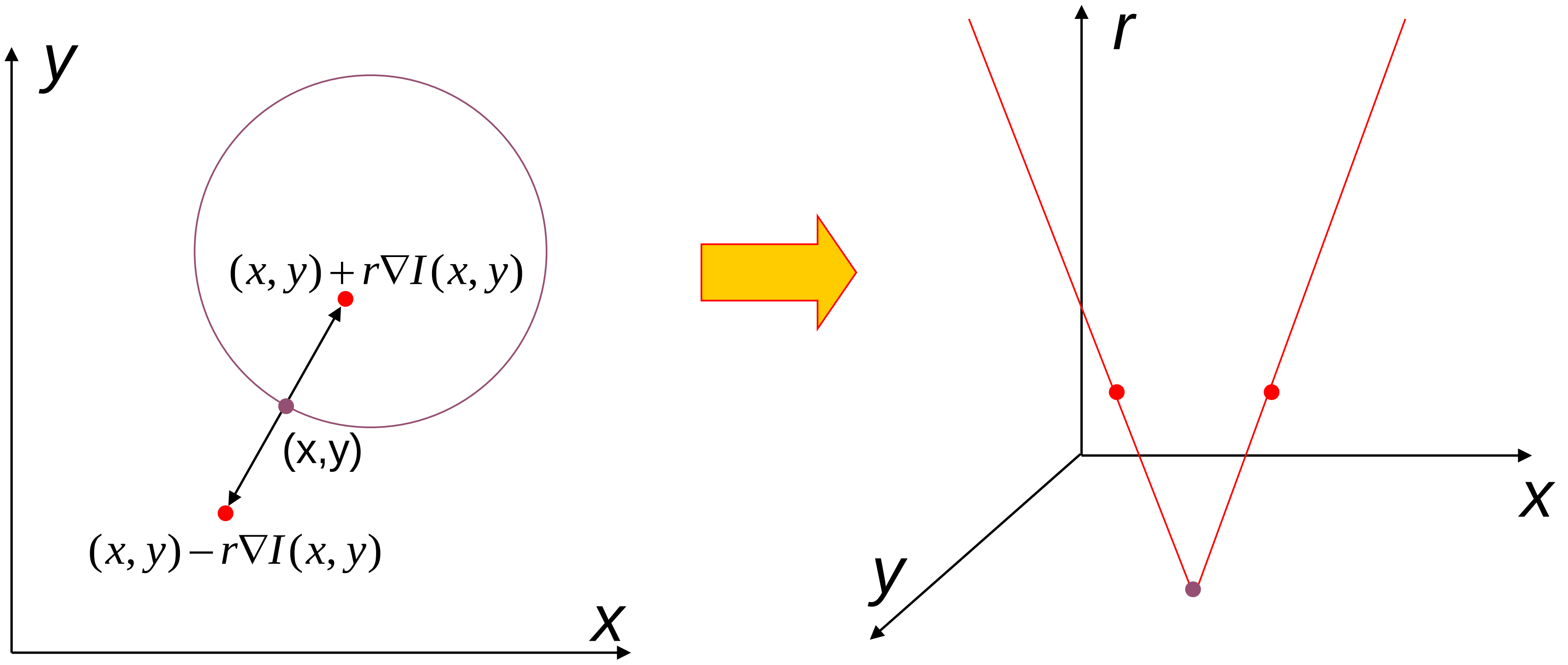


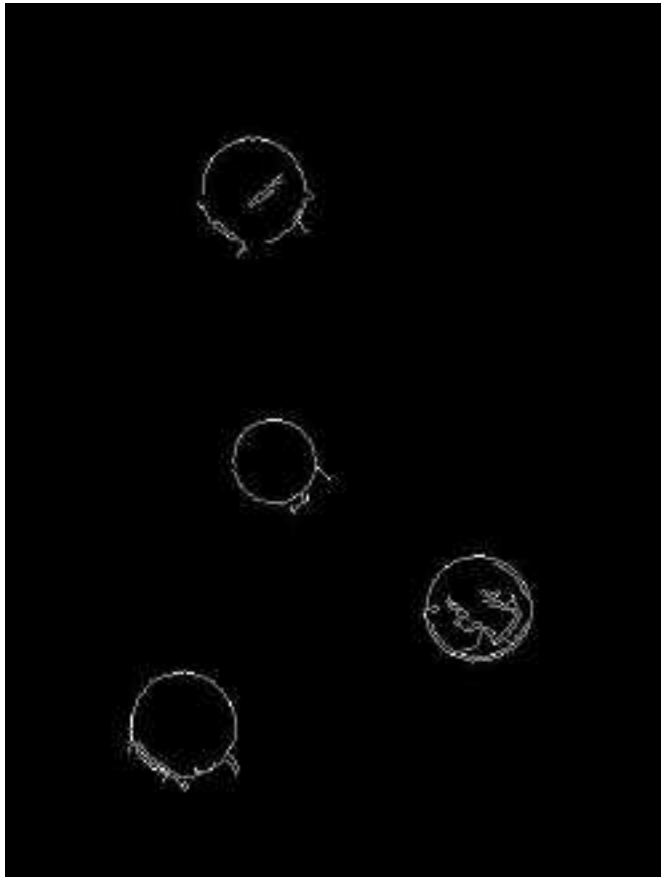
Image space

Hough parameter space

Finding Coins

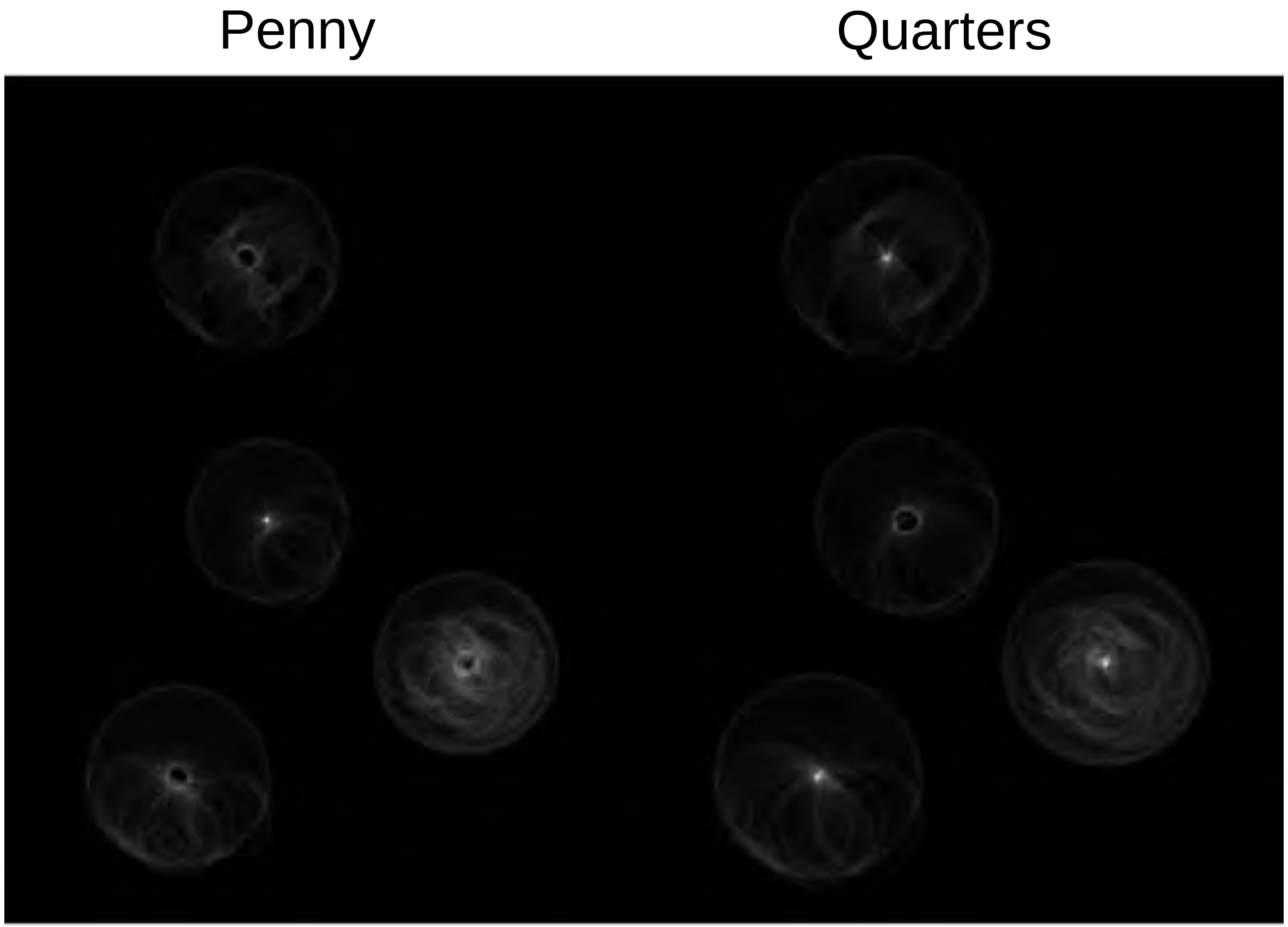
Original

Edges (note noise)



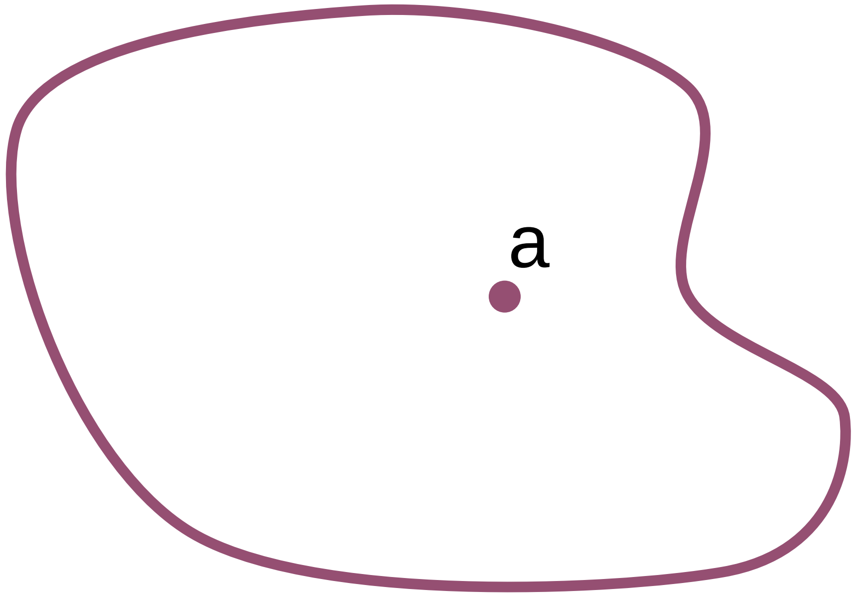
Finding Coins

- Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size



Generalized Hough Transform

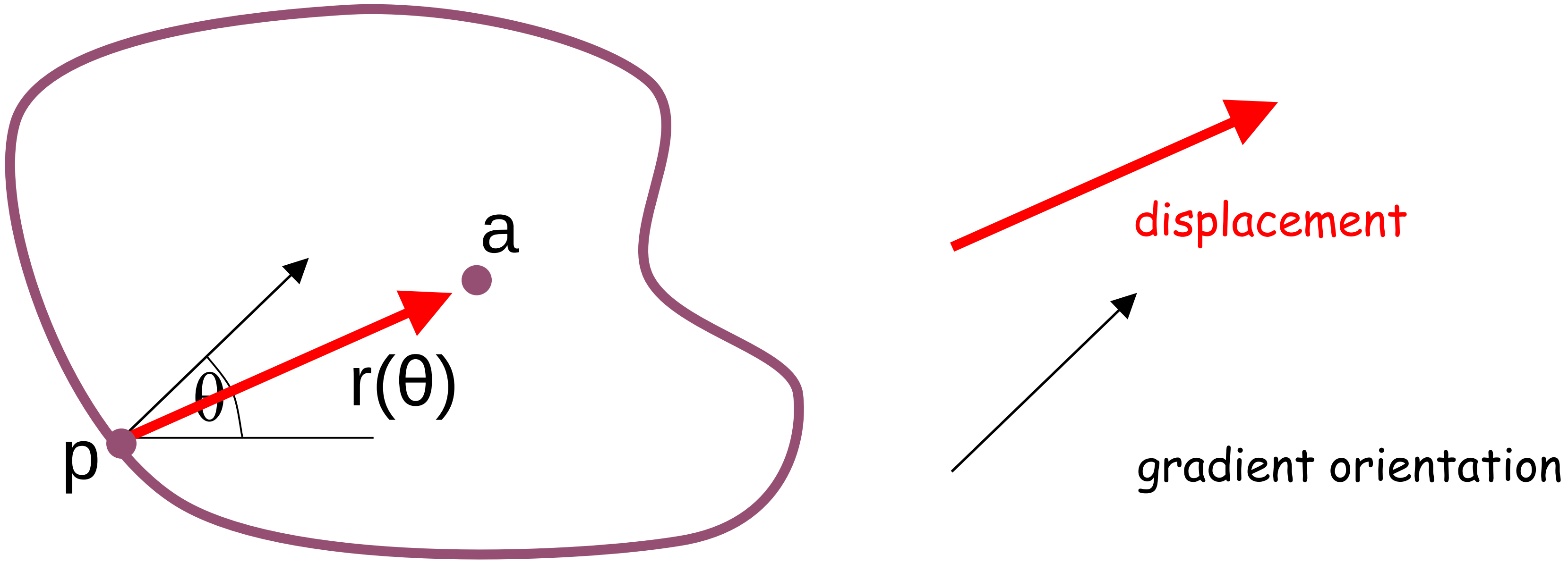
- We want to find a fixed shape **(known)** defined by its boundary points and a reference point



D. Ballard, [Generalizing the Hough Transform to Detect Arbitrary Shapes], PR 13(2), 1981, pp. 111-122.

Generalized Hough Transform

- We want to find a fixed shape **(known)** defined by its boundary points and a reference point
- For every boundary point p, we can compute the displacement vector $r = a - p$ as a function of gradient orientation θ

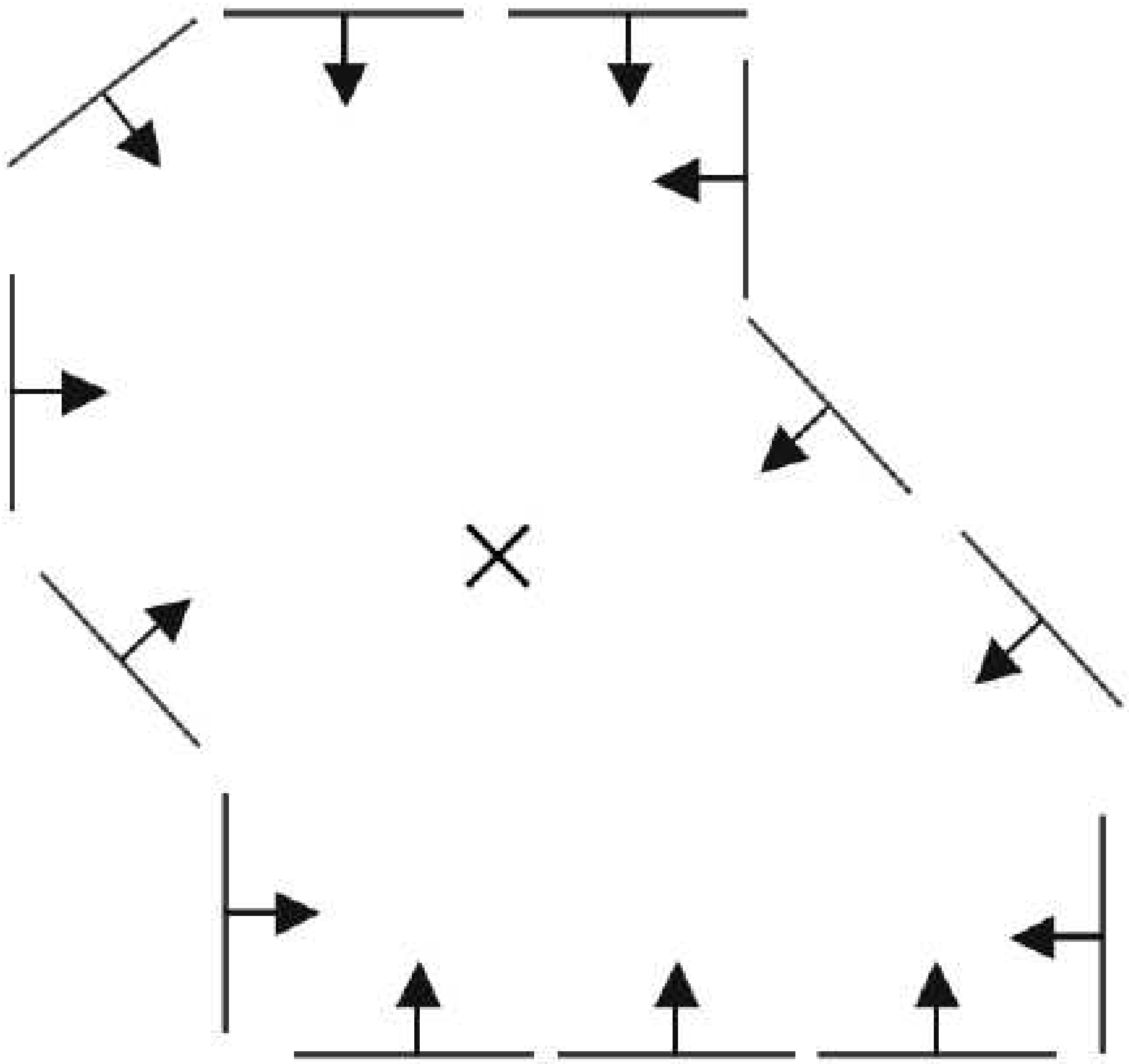


Generalized Hough Transform

- Construct a model for a shape:
  - Construct a table indexed by θ storing displacement vectors r as function of gradient direction
- Detect using the model
  - For each edge point *p* with gradient orientation θ:
    - ✓ Retrieve all r indexed with θ
    - ✓ For each $r(\theta)$, put a vote in the Hough space at $p + r(\theta)$
  - Peak in this Hough space is reference point with most supporting edges
- Assumption: translation is the only transformation here, i.e., orientation and scale are fixed

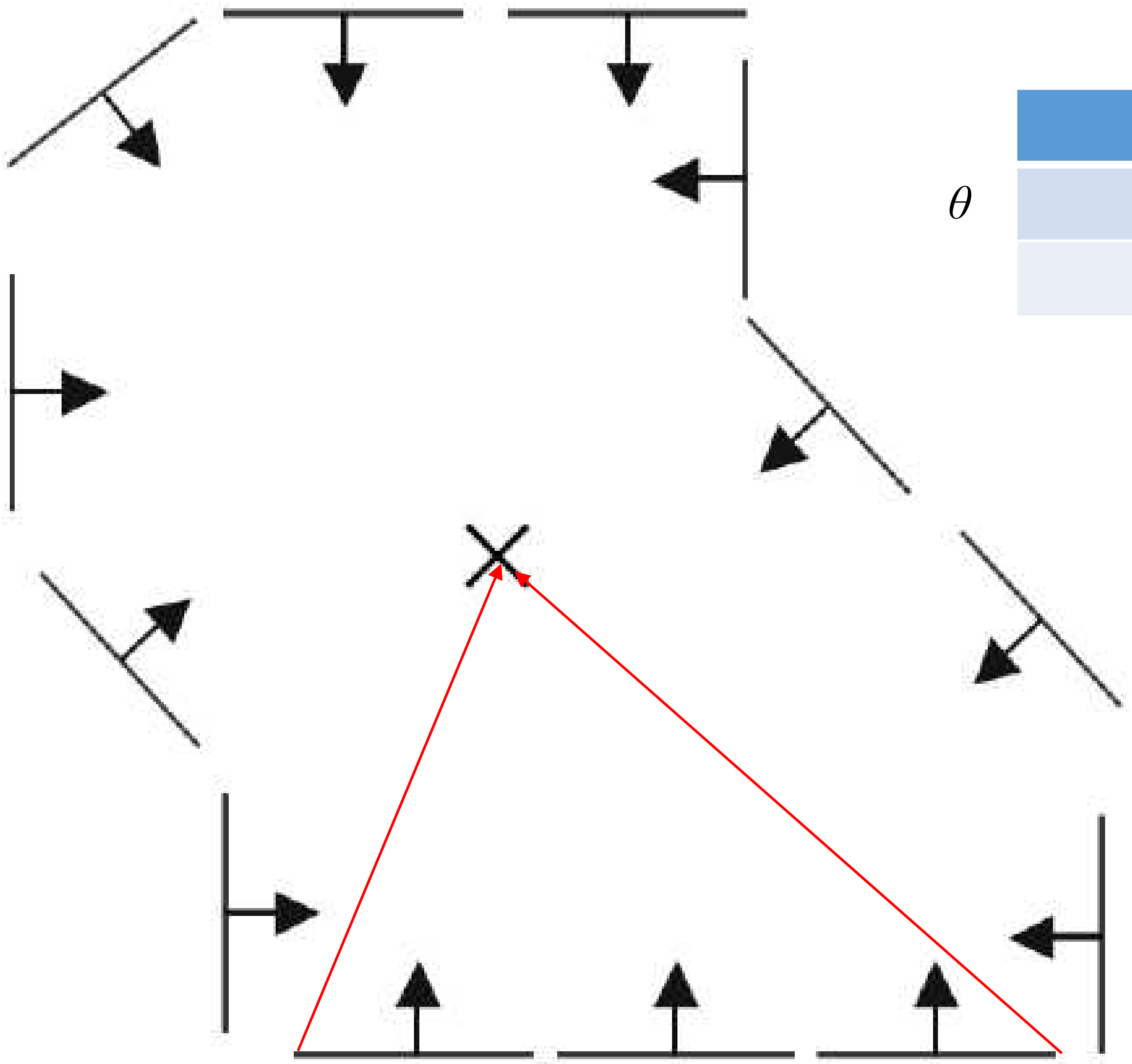
Example: a Known and Fixed Shape

- Model shape
  - Gradient orientation
  - No rotation



Example: Building a Table

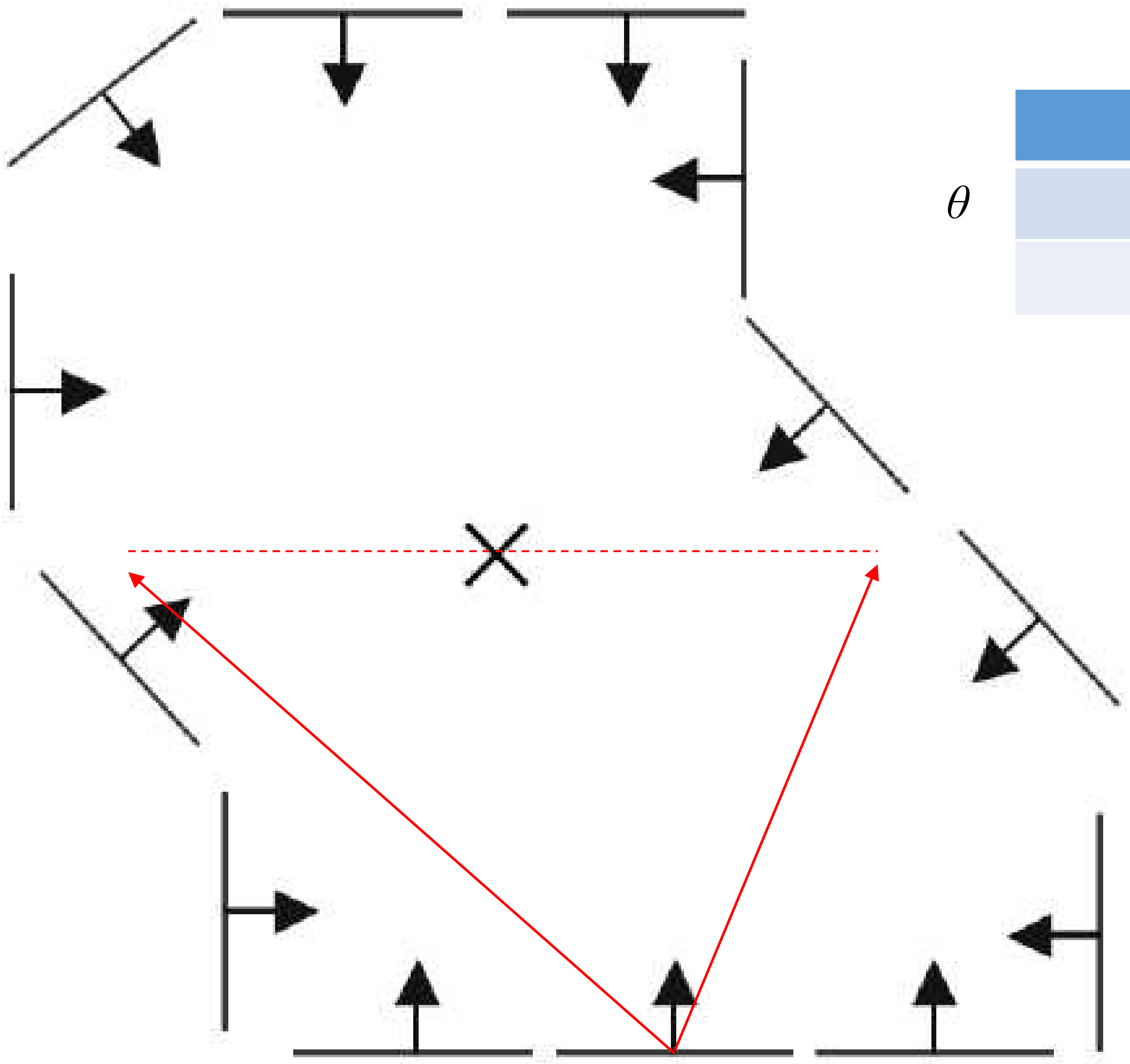
- Displacement vectors for model points

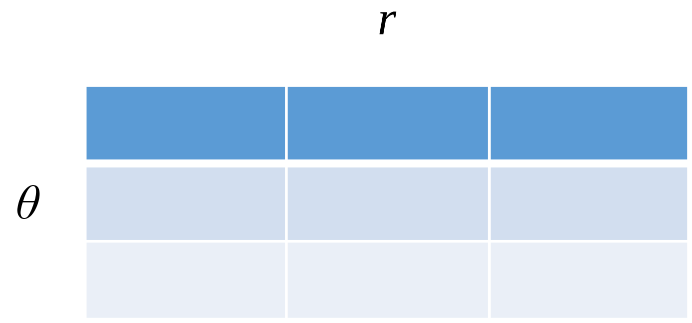


	r		
θ			

Example: Detection

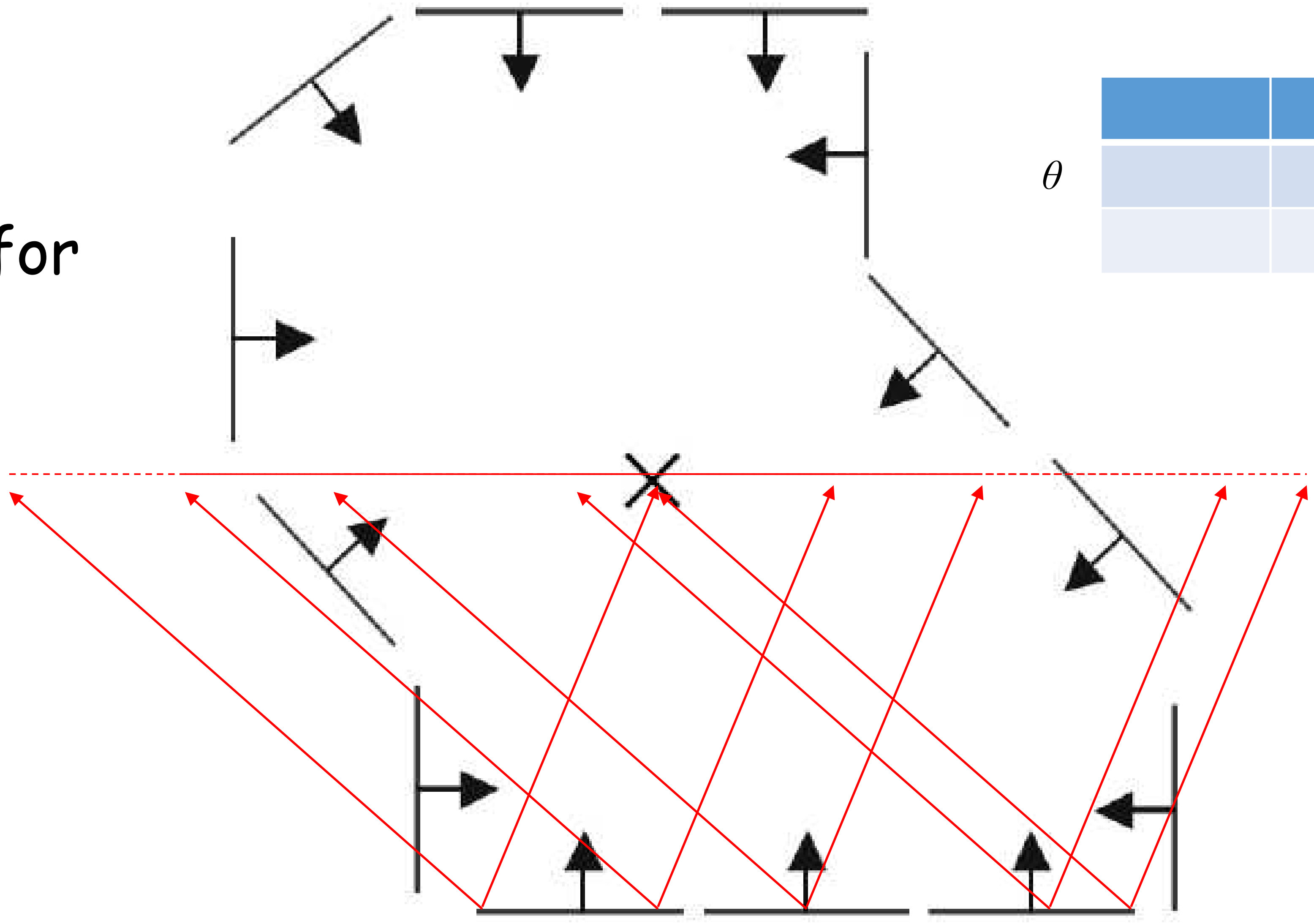
- Range of voting locations for **test** point

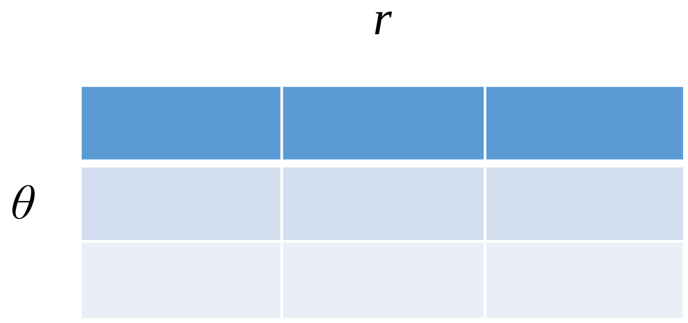




Example: Detection

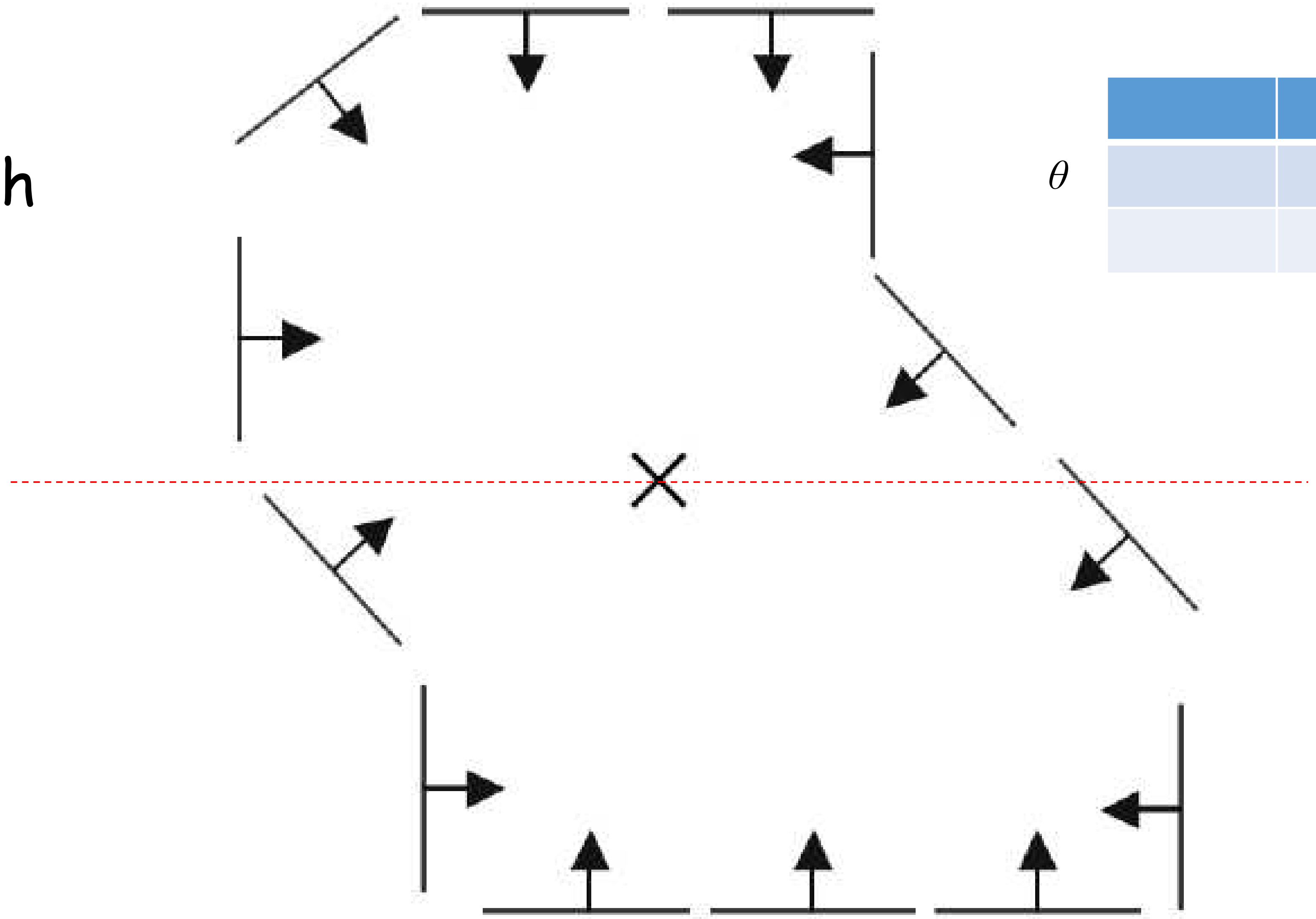
- Range of voting locations for test point





Example: Detection

- Votes for points with
  $\theta = \uparrow$

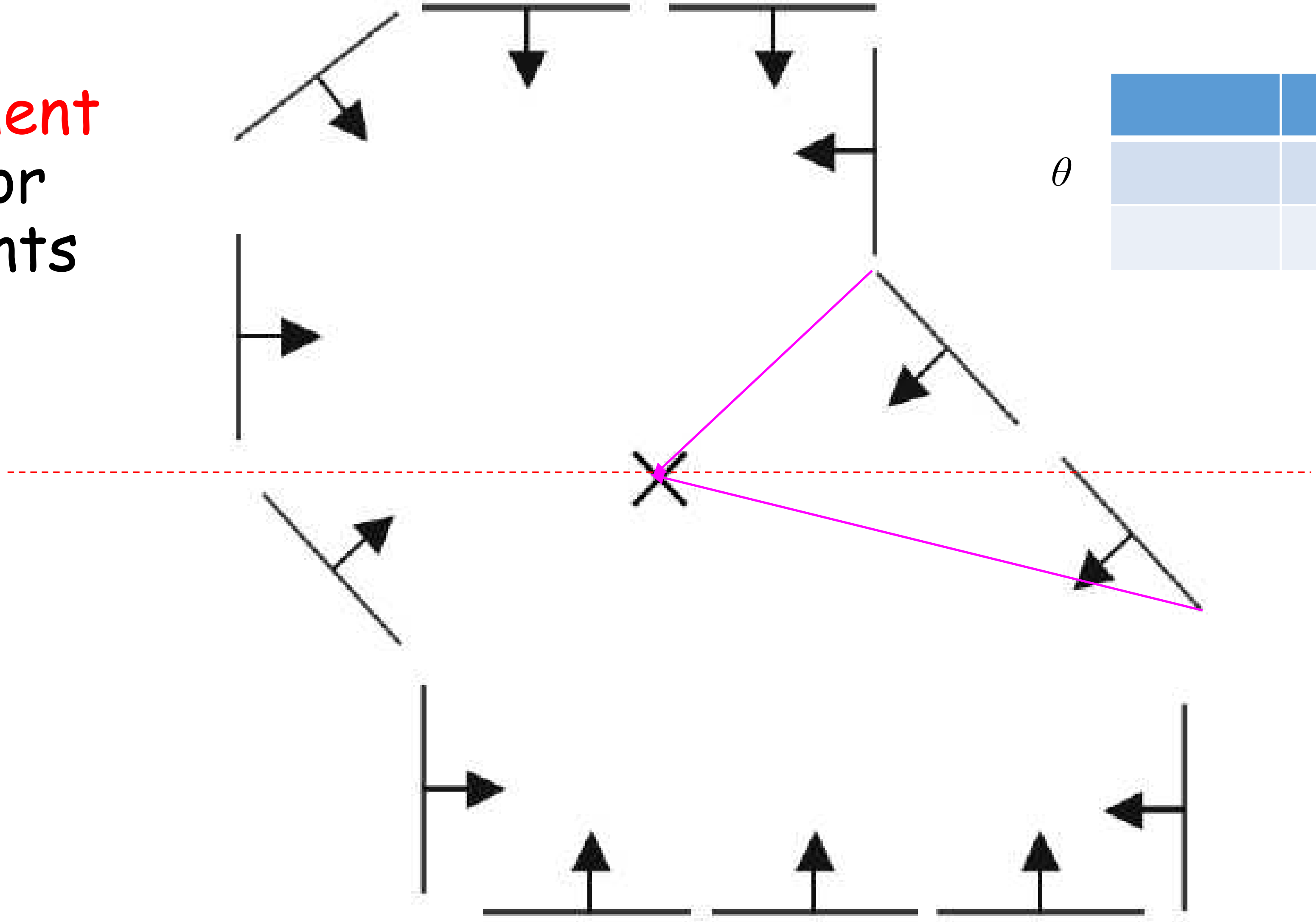


r

θ

Example: Building a Table

- Displacement vectors for model points

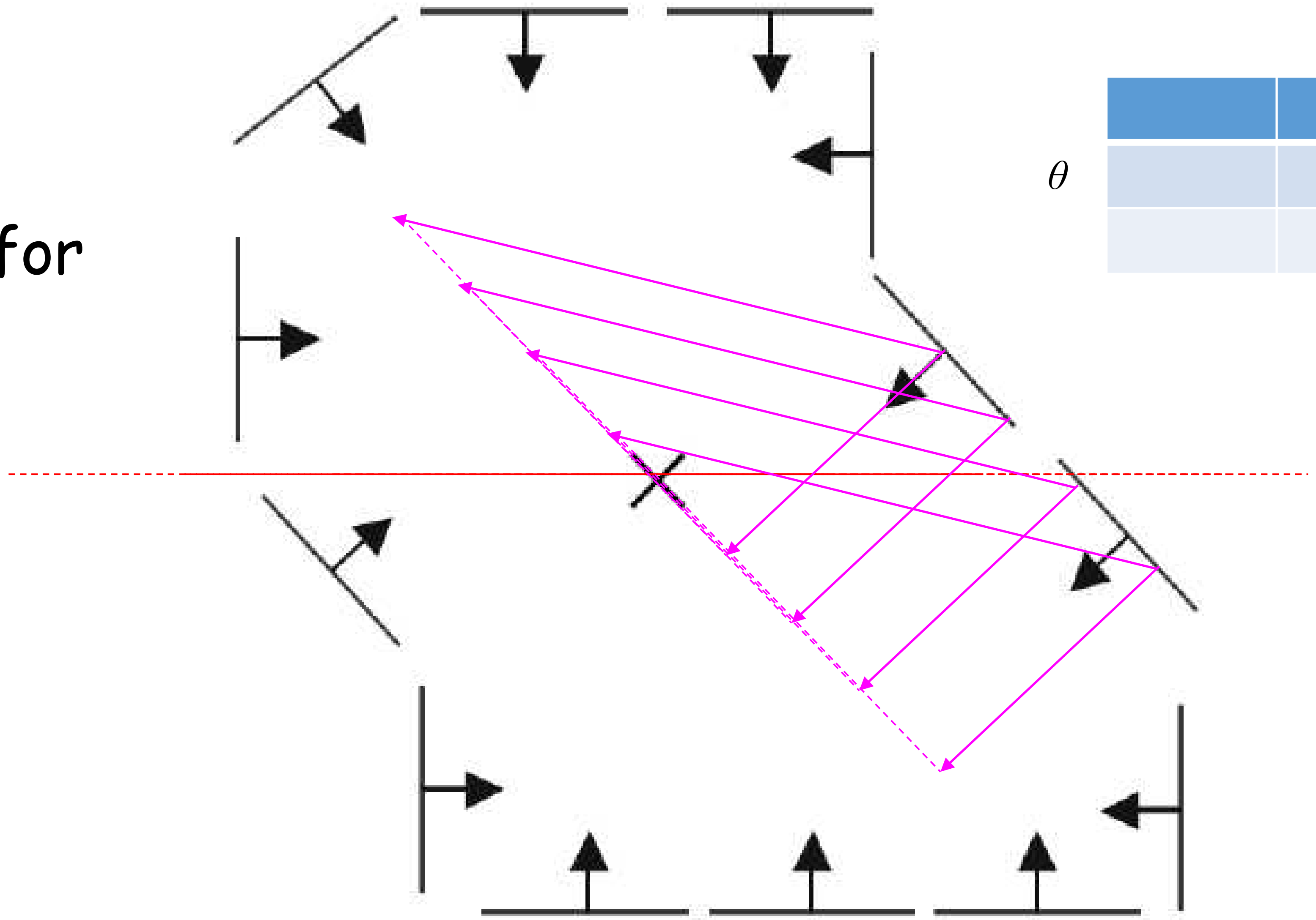


r

θ			

Example: Detection

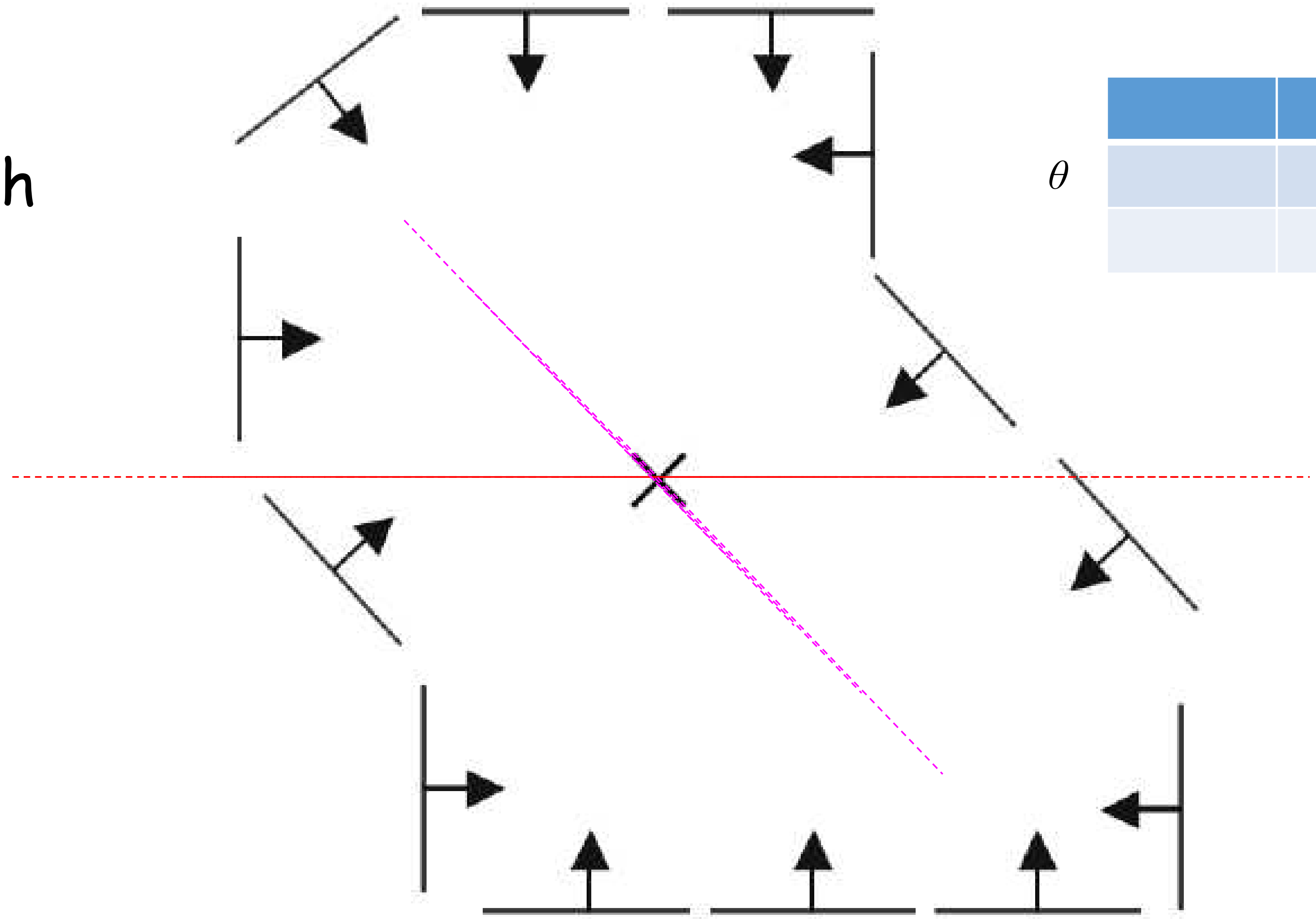
- Range of voting locations for test point

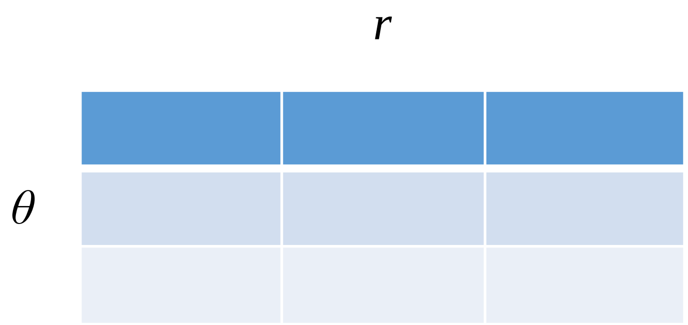


	r		
θ			

Example: Detection

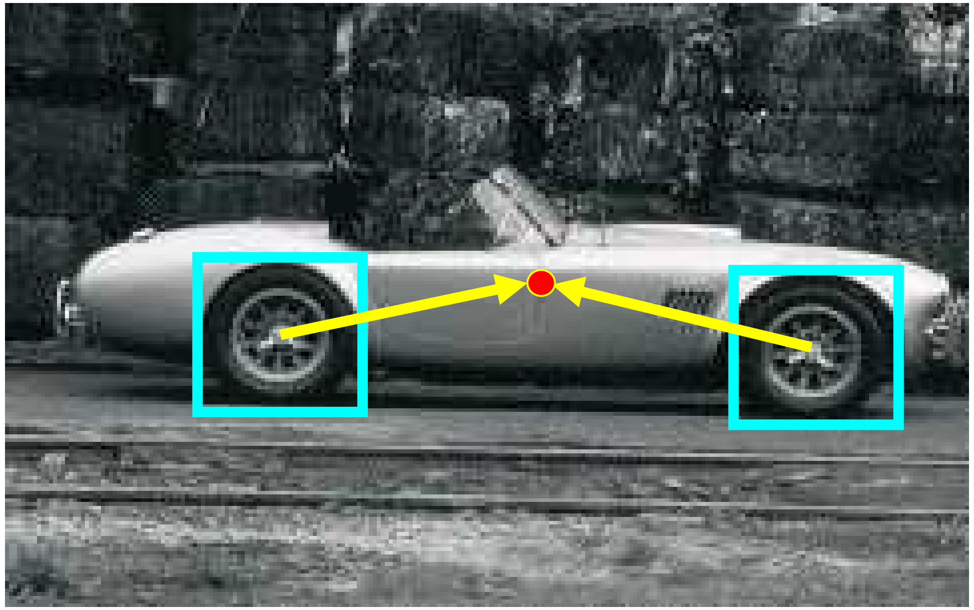
- Votes for points with $\theta = \swarrow$





Application in Recognition

- Instead of indexing displacements by gradient orientation, index by "visual codeword"



training image

What is the codeword?

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

- Instead of indexing displacements by gradient orientation, index by "visual codeword"

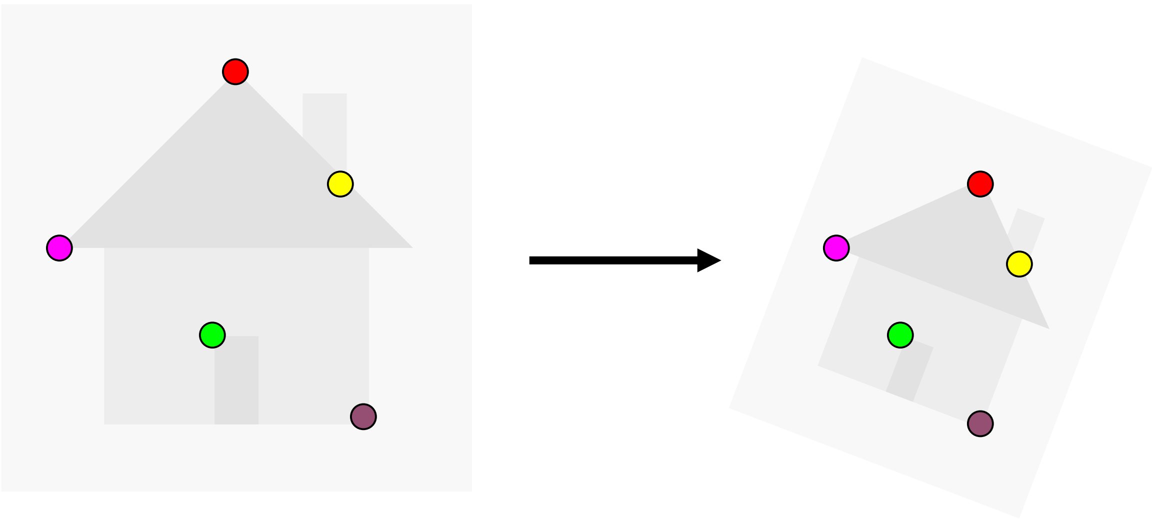
test image

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

Image Alignment

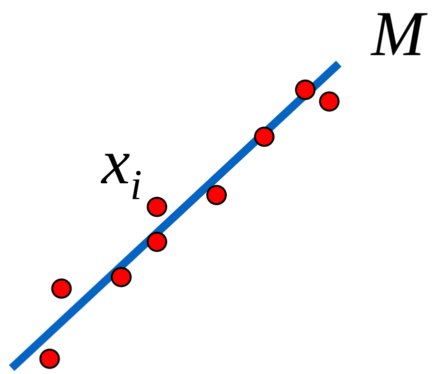
Image Alignment

- Two broad approaches:
  - ➢ Direct (pixel-based) alignment
    - ✓ Search for alignment where most pixels agree
  - ➢ Feature-based alignment
    - ✓ Search for alignment where *extracted features* agree
    - ✓ Can be verified using pixel-based alignment



Alignment as Fitting

- Previously: fitting a model to features in one image

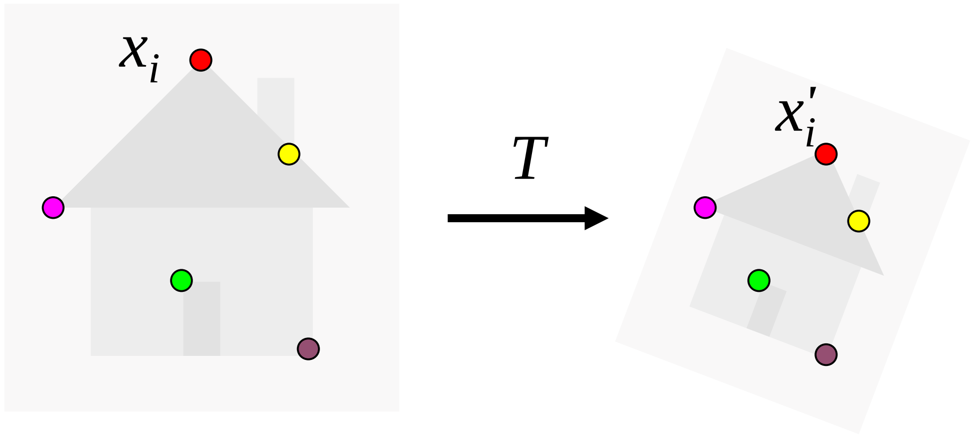


Find model M that minimizes

$$\sum_i \text{residual}(x_i, M)$$

Alignment as Fitting

- Alignment: fitting a model to a transformation between pairs of features (*matches*) in two images

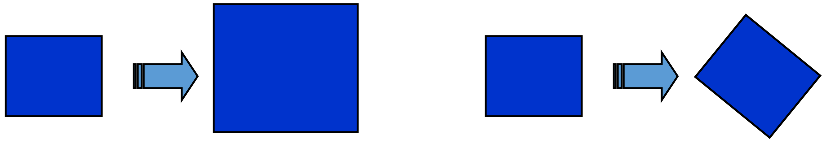


Find transformation T that minimizes

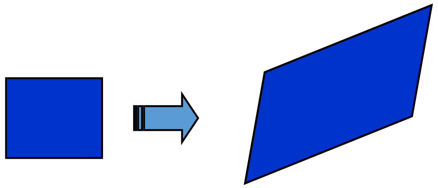
$$\sum_i \text{residual}(T(x_i), x_i')$$

2D Transformation Models

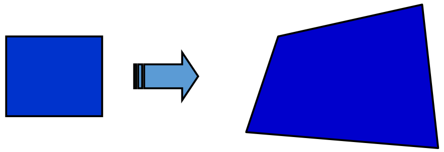
- Similarity (translation, scale, rotation)



- Affine

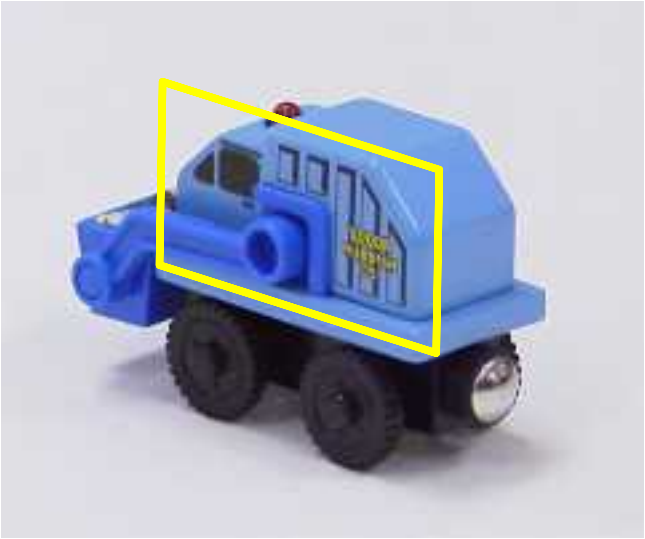


- Projective (homography)



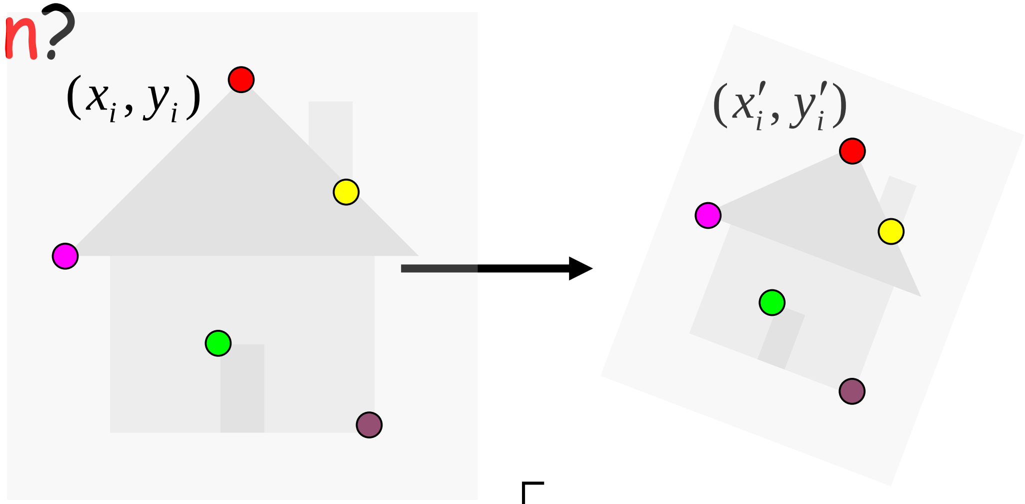
Affine Transformations

- Simple fitting procedure (linear least squares)
- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models



Affine Transformations

- Assume we know the correspondences (???), how do we get the transformation?



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \Rightarrow \begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

Affine Transformations

- Linear system with six unknowns
- Each match gives us two linearly independent equations: need at least three to solve for the transformation parameters

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

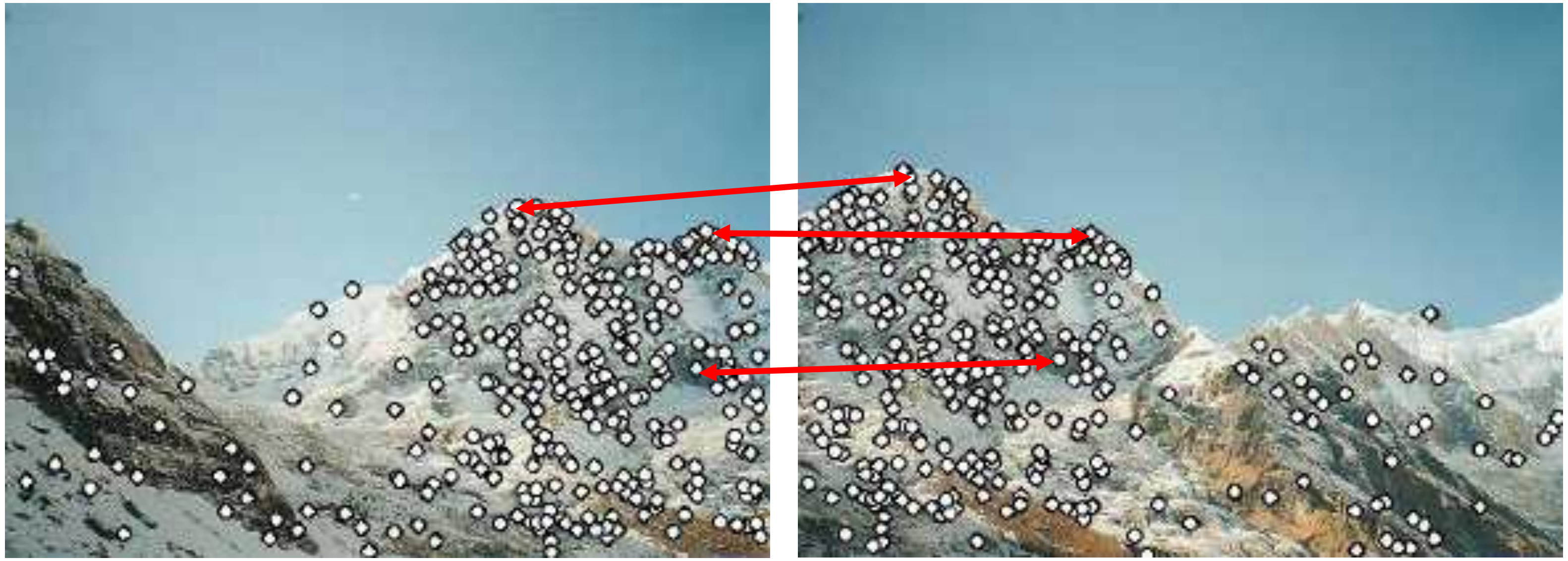
Feature-Based Alignment Outline

- Extract features

Feature-Based Alignment Outline

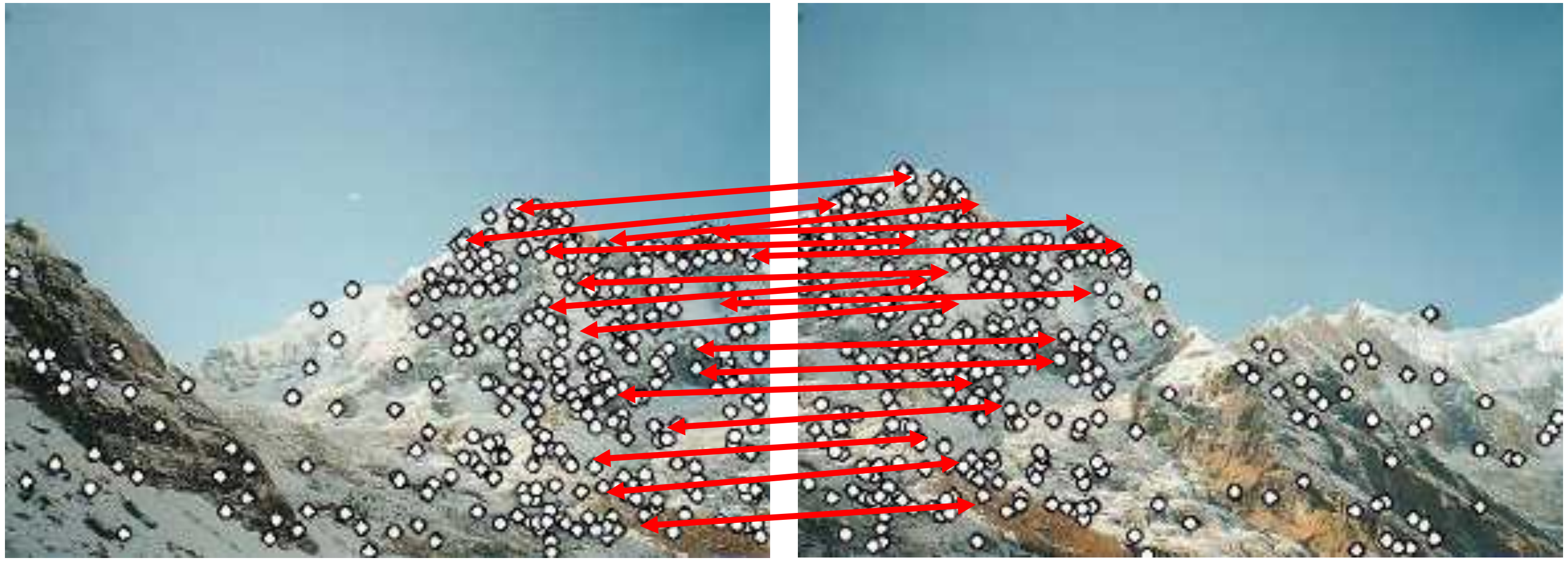
- Extract features
- Compute *putative matches*

Feature-Based Alignment Outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T*

Feature-Based Alignment Outline



- Extract features
- Compute *putative* matches
- Loop:
  - ➢ *Hypothesize* transformation *T*
  - ➢ *Verify* transformation (search for other matches consistent with *T*)

Feature-Based Alignment Outline

- Extract features
- Compute *putative matches*
- Loop:
  - ➢ *Hypothesize* transformation *T*
  - ➢ *Verify* transformation (search for other matches consistent with *T*)

Conclusions

Conclusion

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting
- Alignment as a fitting problem

Thanks

zhengf@sustc.edu.cn