Unit 6
Model Fitting and Image Alignment

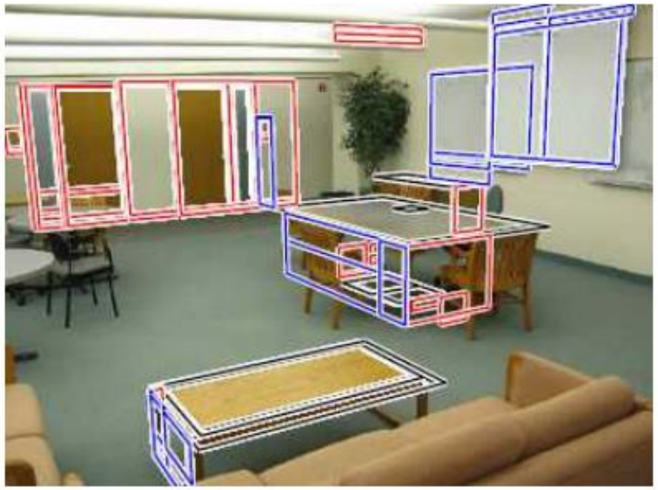
Shang-Hong Lai

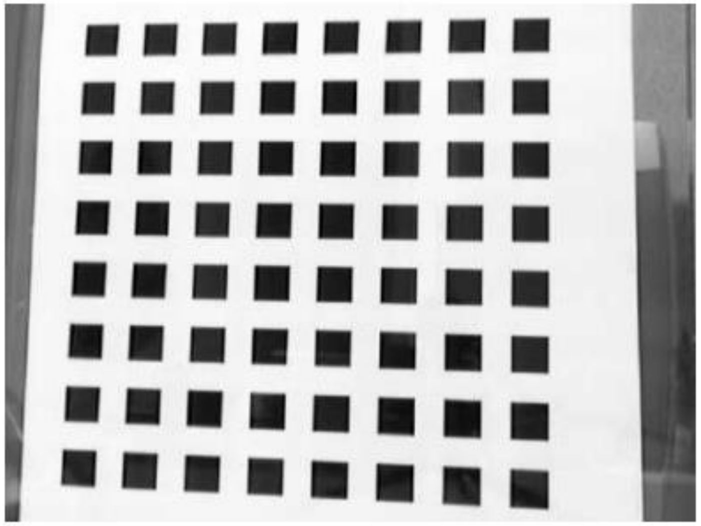
Szeliski (Sec. 7.4, 8.1)

Overview

- Motivation for model fitting
  - Line fitting
  - Estimating fundamental matrix
- Hough transform
  - Fitting lines with the Hough transform
  - Sensitivity to noise & implementation concerns
- Model fitting from data with outliers
  - M-estimation
  - RANSAC

Line Fitting in Real-World Applications



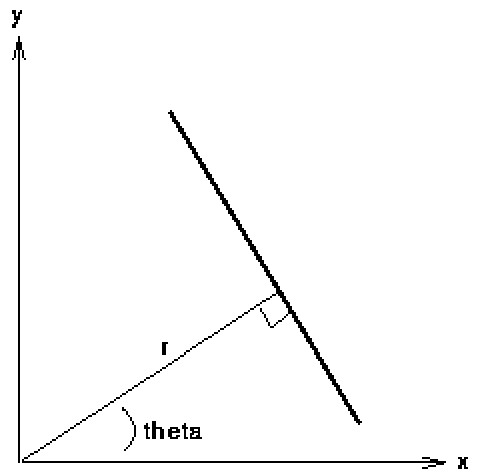


Model Fitting

- Choose a parametric object/some objects to represent a set of tokens
- Most interesting case is when the criterion is not local
  - can't tell whether a set of points lies on a line by looking only at each point and the next.
- Three main questions:
  - what object represents this set of tokens best?
  - which of several objects gets which token?
  - how many objects are there?

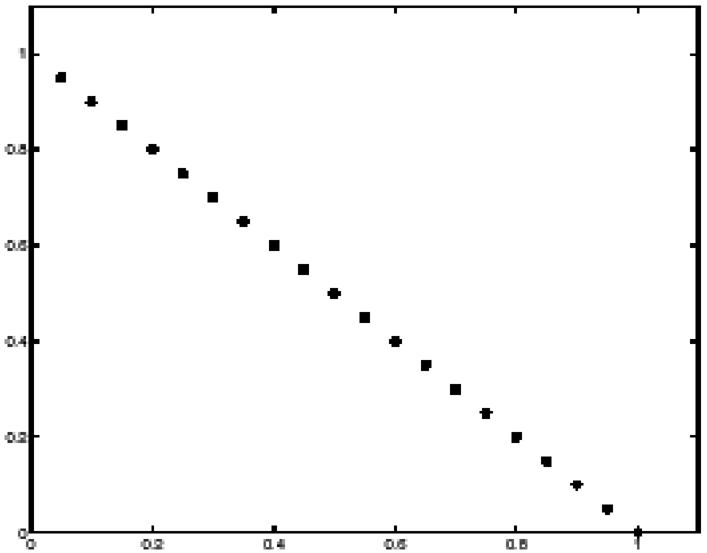
Hough Transform for Line Detection

- Purports to answer all three questions
  - in practice, a good result requires good input.
- We explain for lines first
- One representation: a line is the set of points (x, y) such that:
  $(\cos \theta) X + (\sin \theta) Y + r = 0$

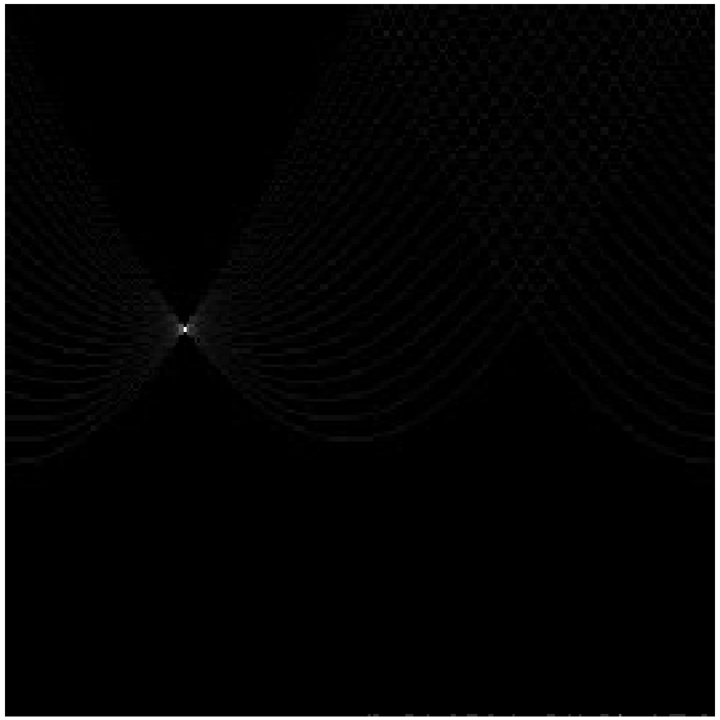


- Different choices of θ, r>0 give different lines
- For any token (x, y) there is a one parameter family of lines through this point, given by $(\cos \theta) X + (\sin \theta) Y + r = 0$
- Each point gets to vote for each line in the family; if there is a line that has lots of votes, that should be the line passing through the points

Tokens



Votes



r: 0 to 1.55

Theta = 45º = 0.785 rad
r = (1√2) / 2 = 0.707

Theta: 0 to 3.14 (rad)

Brightest point = 20 votes

Mechanics of the Hough transform

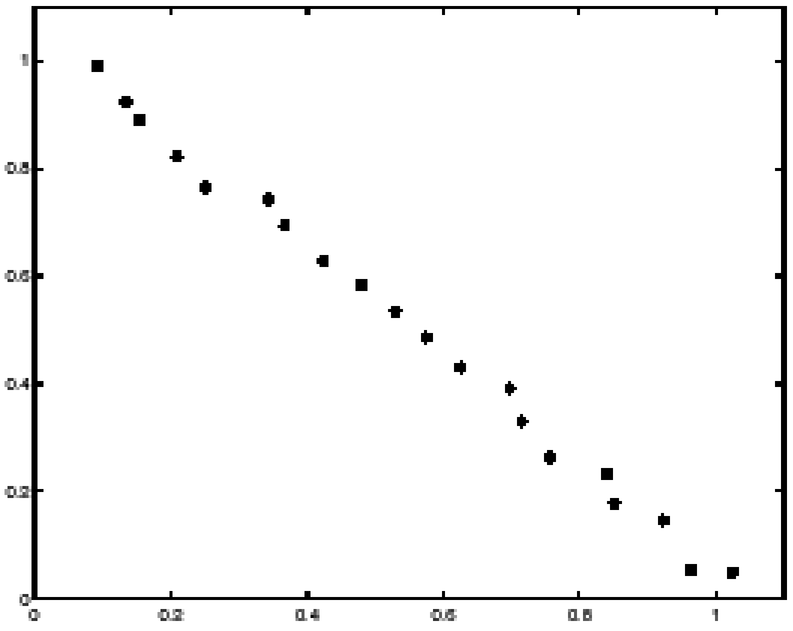
- Construct a voting array representing θ, r
- For each point, render the curve (θ, r) into this array, adding one at each cell
- Difficulties
  - how big should the cells be? (too big, and we cannot distinguish between quite different lines; too small, and noise causes lines to be missed)

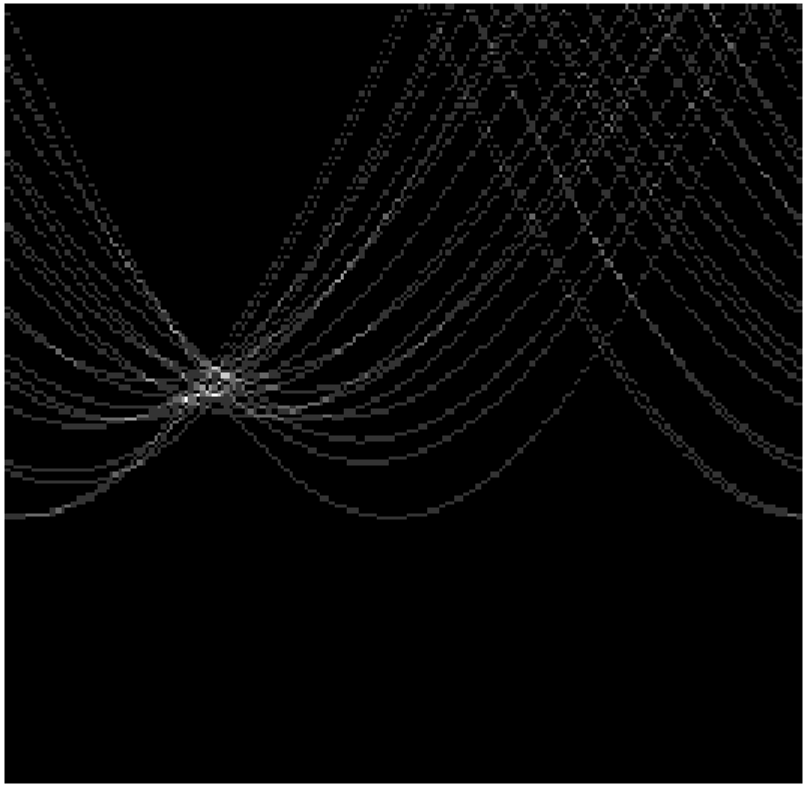
- How many lines?
  - count the peaks in the Hough array
- Who belongs to which line?
  - tag the votes

- Hardly ever satisfactory in practice, because problems with noise and cell size defeat it

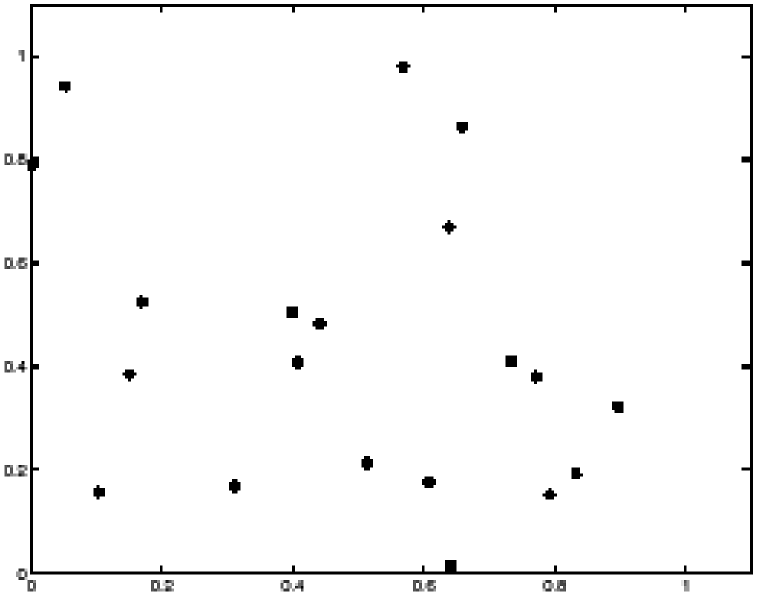
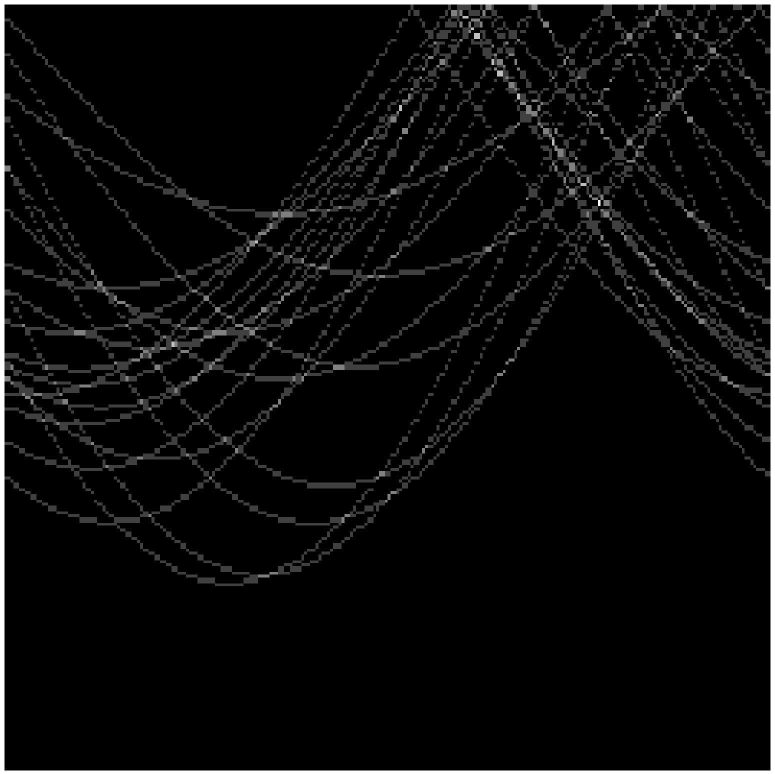
Votes

Tokens

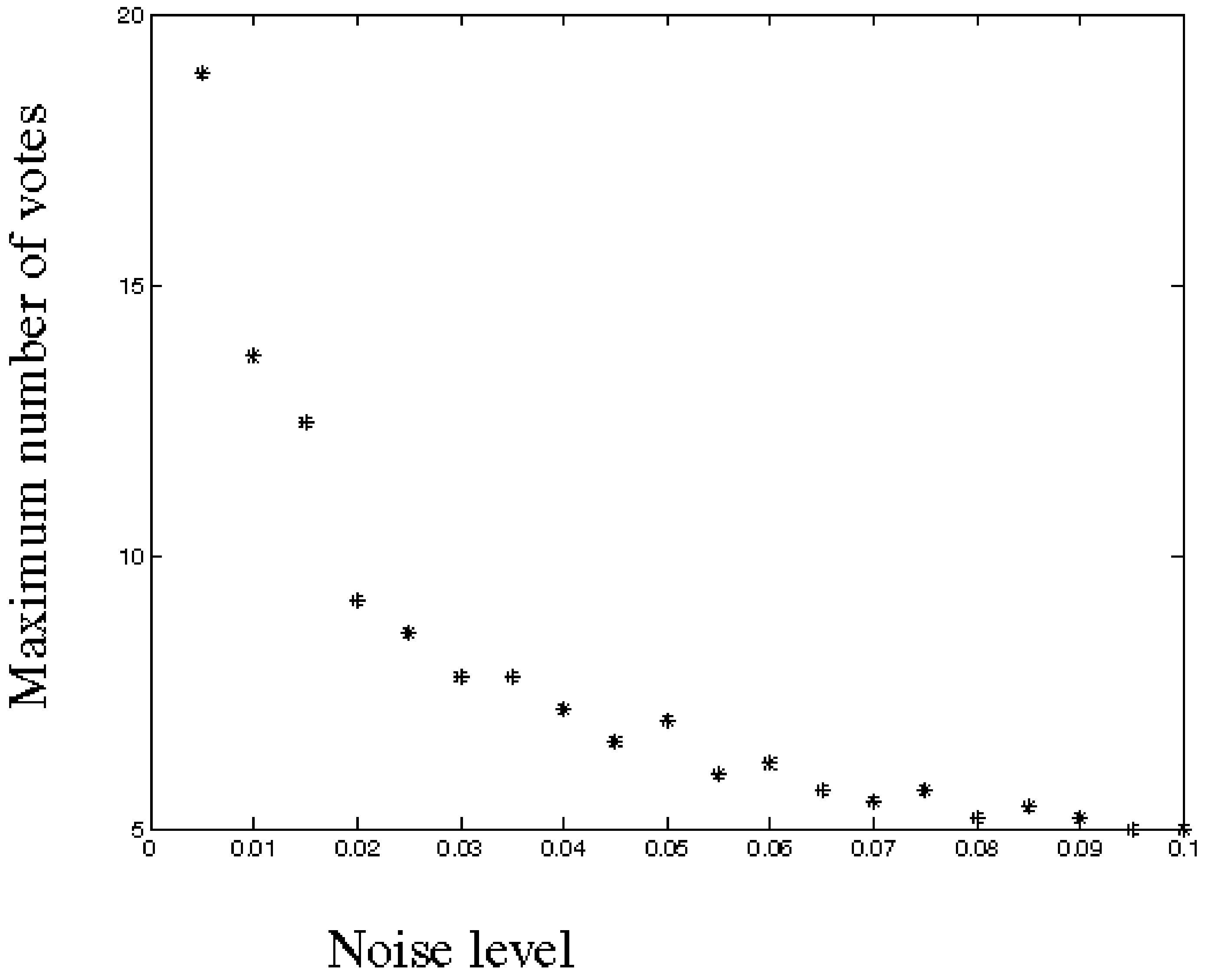




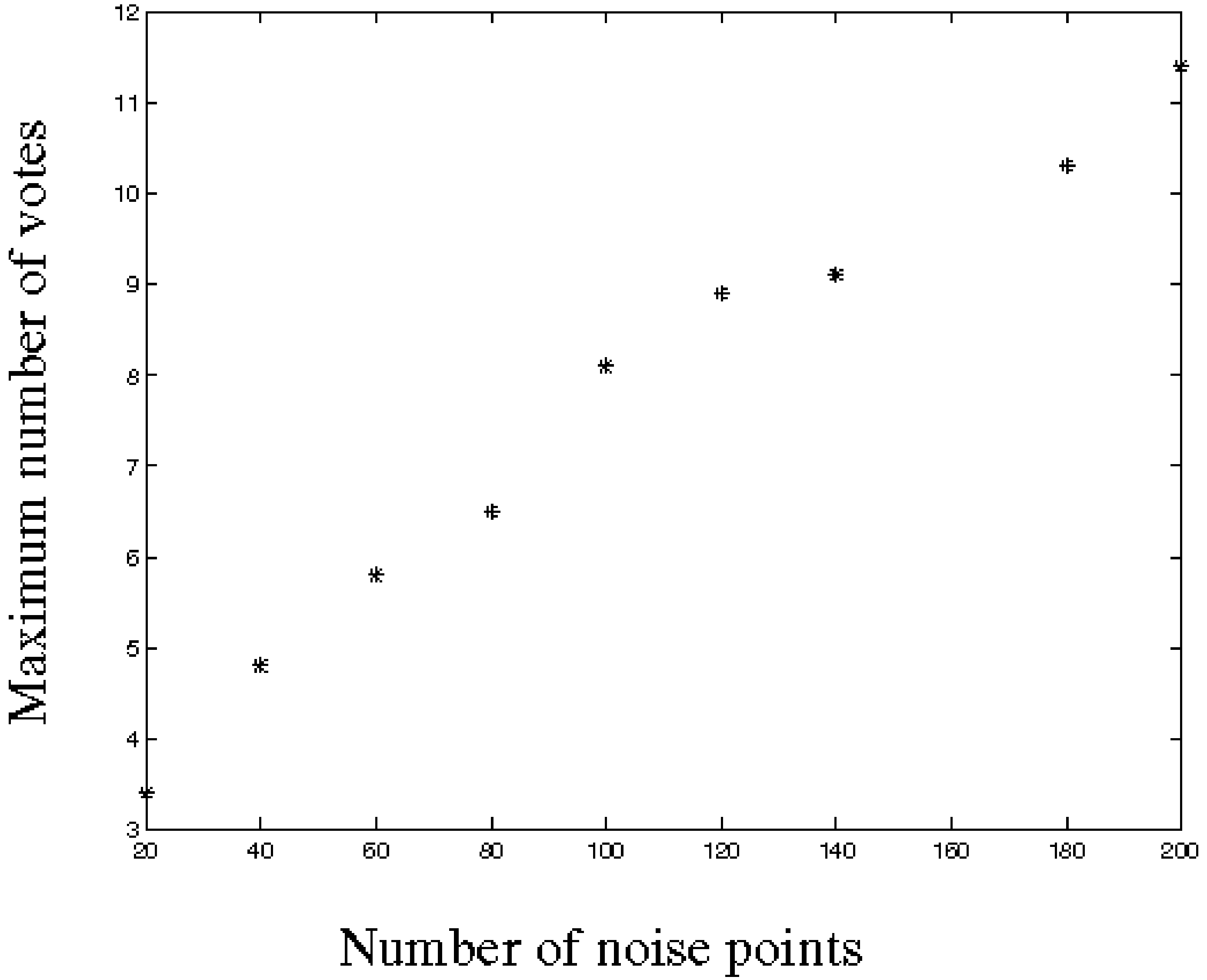
Brightest point = 6 votes


0
0.2
0.4
0.6
0.8
1


Noise Lowers the Peaks



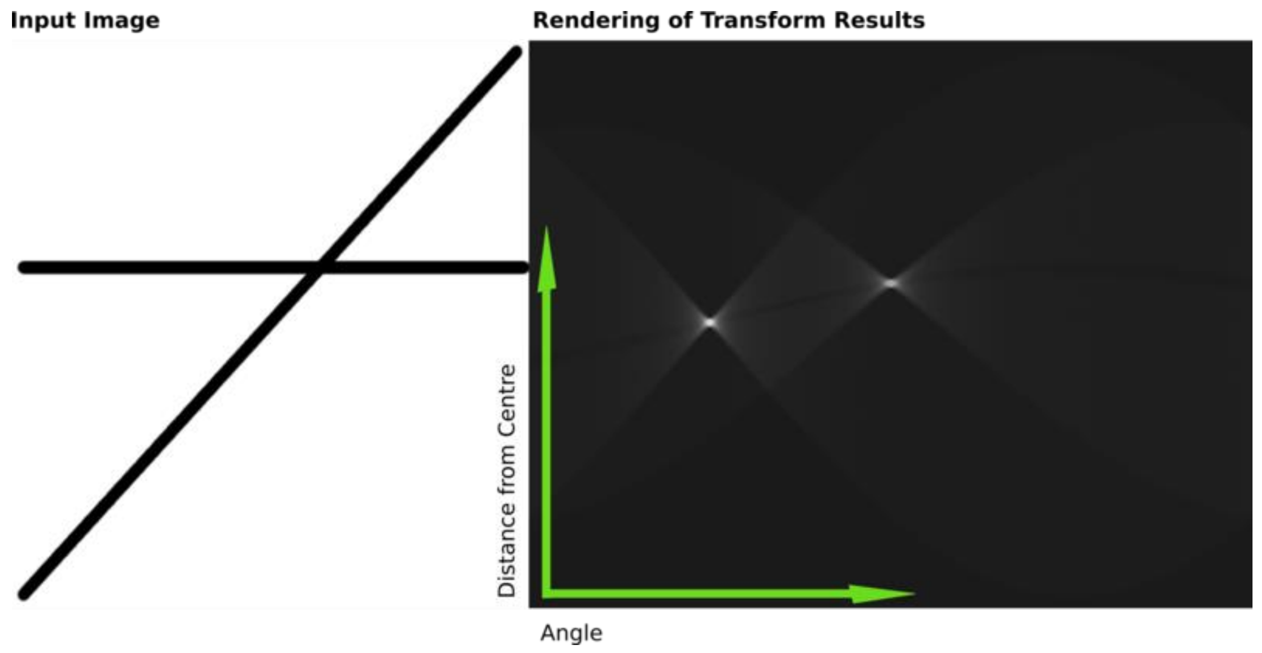
Noise Increases the Votes in Spurious Accumulator Elements



Optimization to the Hough Transform

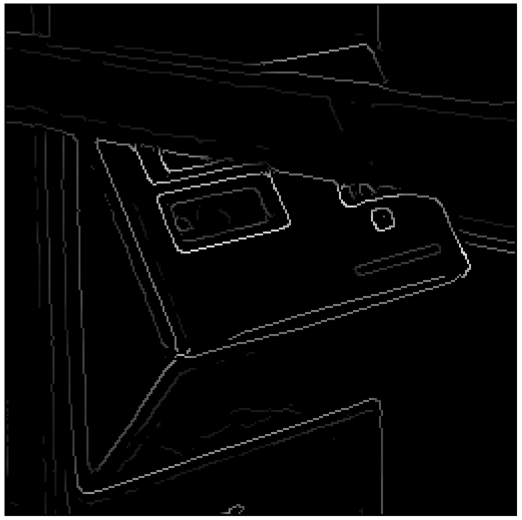
Noise: If the orientation of tokens (pixels) is known, only accumulator elements for lines with that general orientation are voted on. (Most edge detectors give orientation information.)

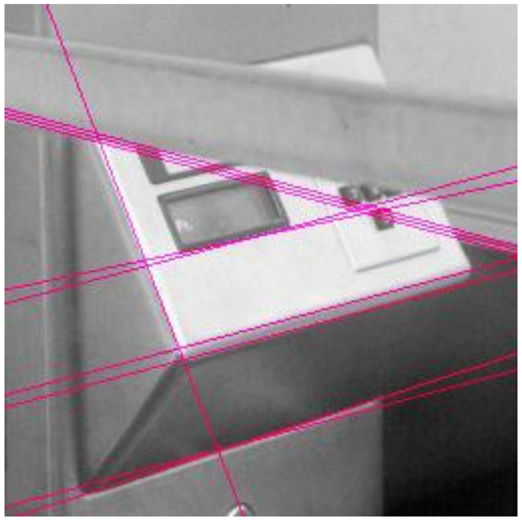
Speed: The accumulator array can be coarse, then repeated in areas of interest at a finer scale.

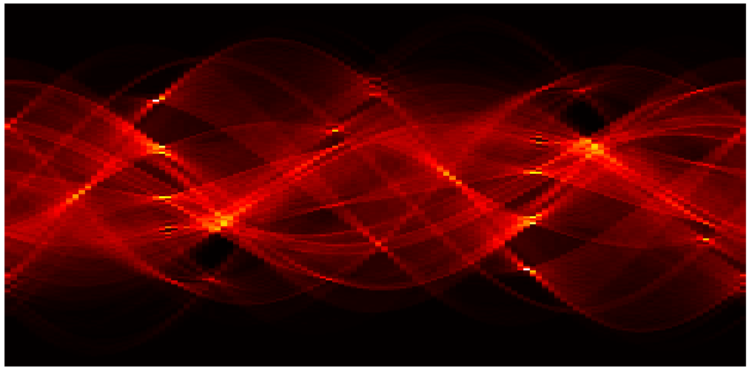


Real World Example

Original


Edge Detection


Found Lines


Hough Space

Least-Square Line Fitting

we want to choose the line that minimizes

$$\sum_i (y_i - ax_i - b)^2.$$

By differentiation, the line is given by the solution to the problem

$$\begin{pmatrix} \overline{y^2} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} \overline{x^2} & \overline{x} \\ \overline{x} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

To minimize the sum of perpendicular distances between points and lines, we need to minimize

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

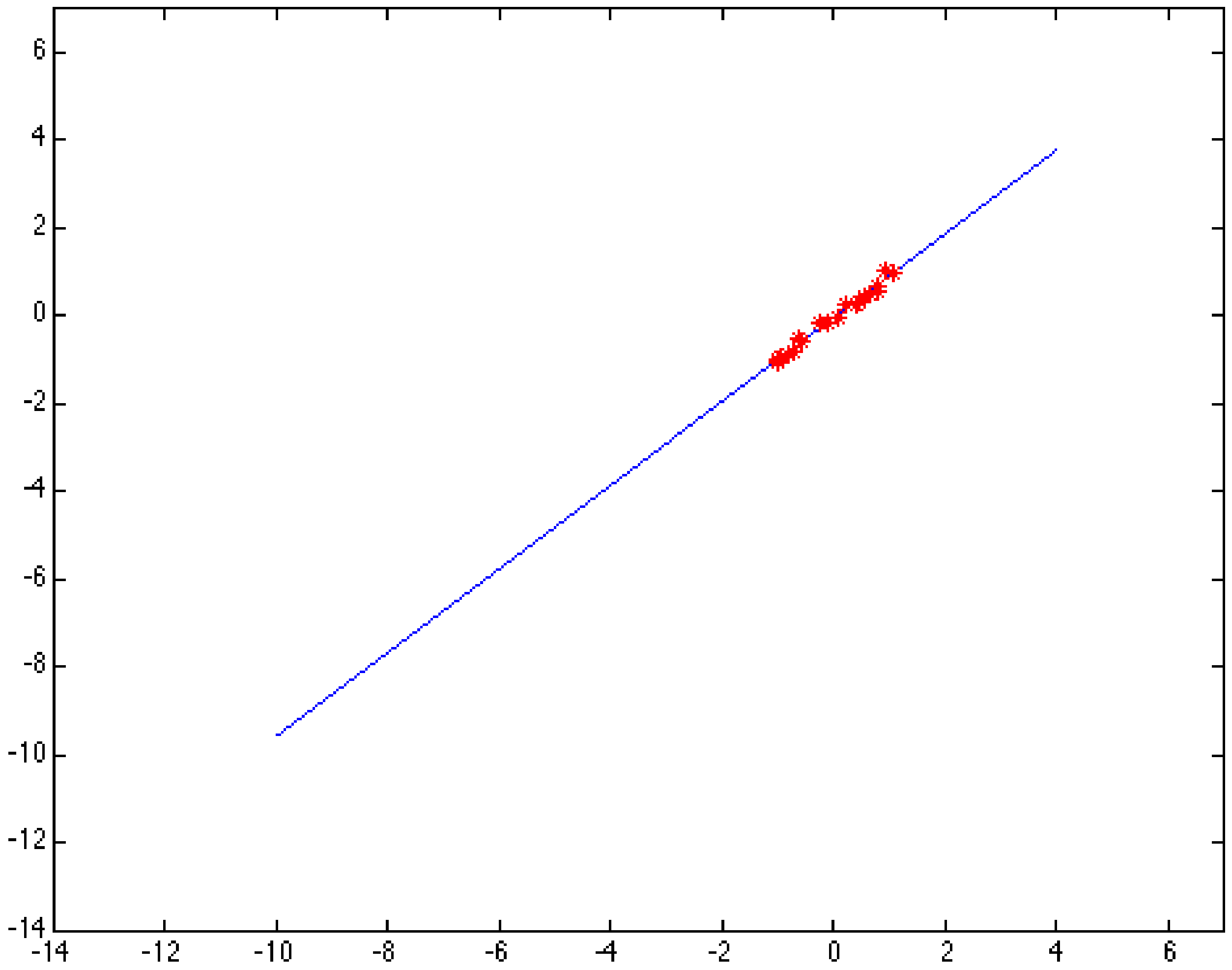
subject to $a^2 + b^2 = 1$.

Algorithm 15.1: Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large

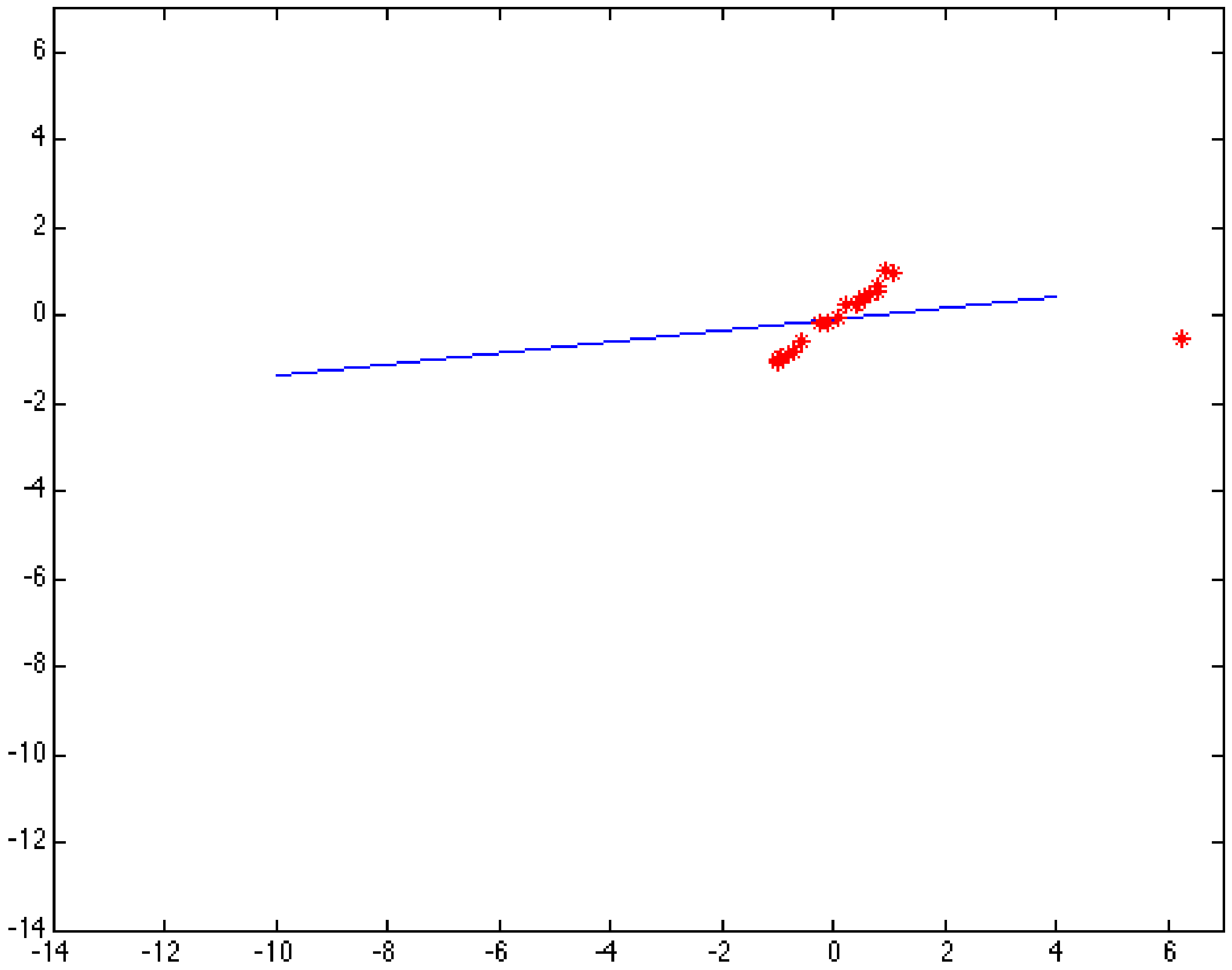
```
Put all points on curve list, in order along the curve
Empty the line point list
Empty the line list
Until there are too few points on the curve
    Transfer first few points on the curve to the line point list
    Fit line to line point list
    While fitted line is good enough
        Transfer the next point on the curve
            to the line point list and refit the line
    end
    Transfer last point(s) back to curve
    Refit line
    Attach line to line list
end
```

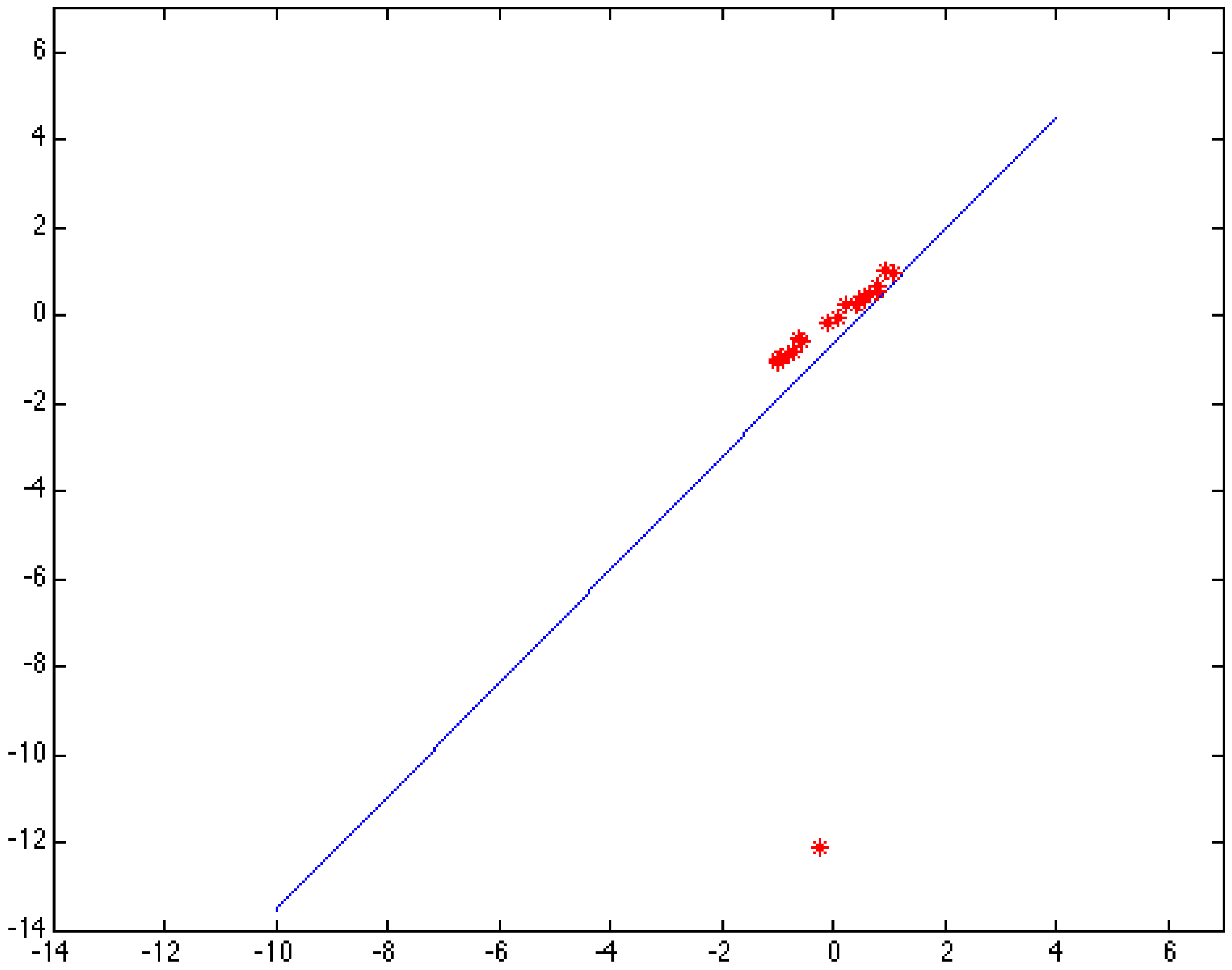
Robust Model Fitting

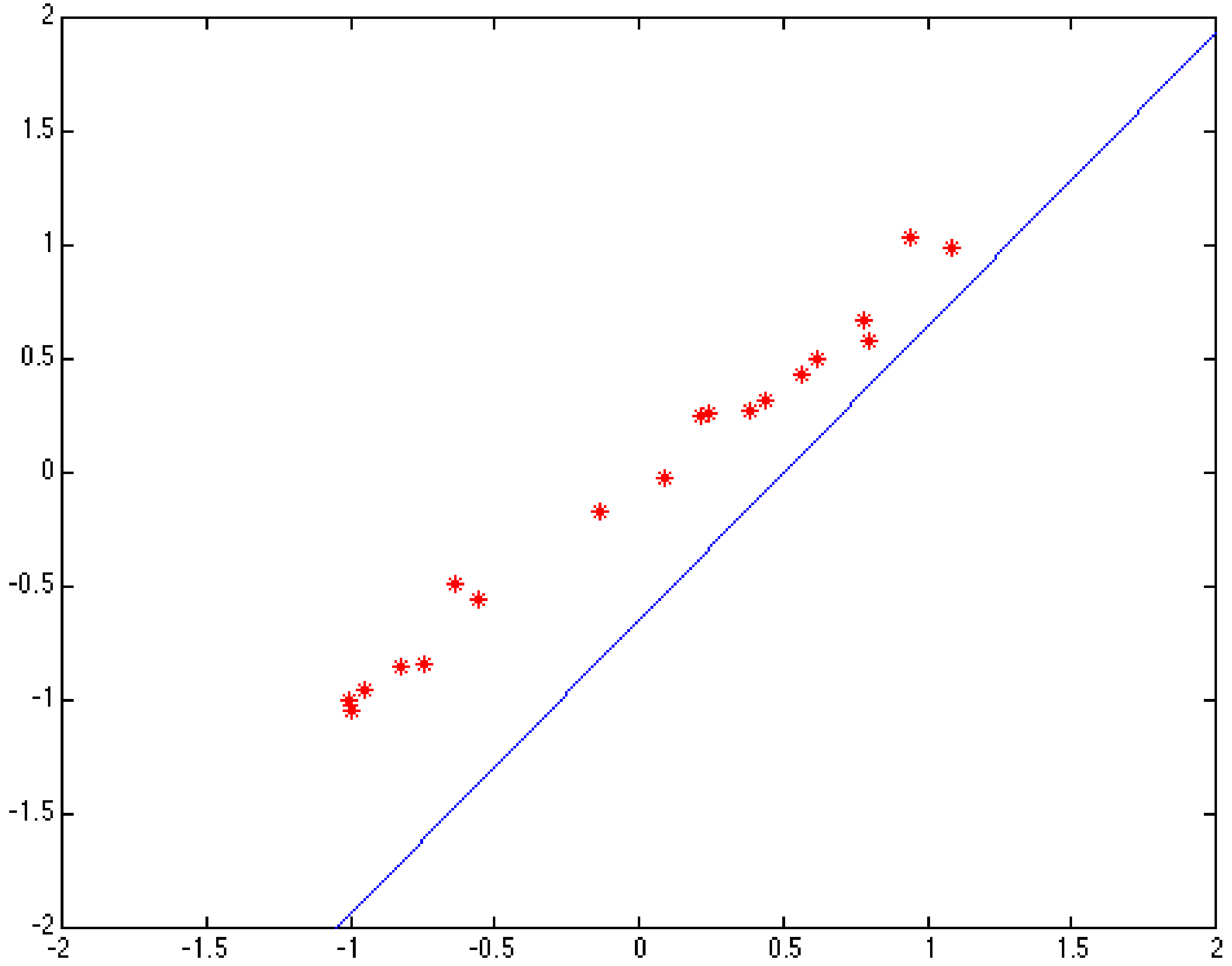
- Least square estimation can be very sensitive to the presence of high-level noises or outliers
- Robust estimation
  - M-estimator
    - trimmed least square estimation
      - Square nearby, threshold far away
  - RANSAC
    - Search for good points



6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

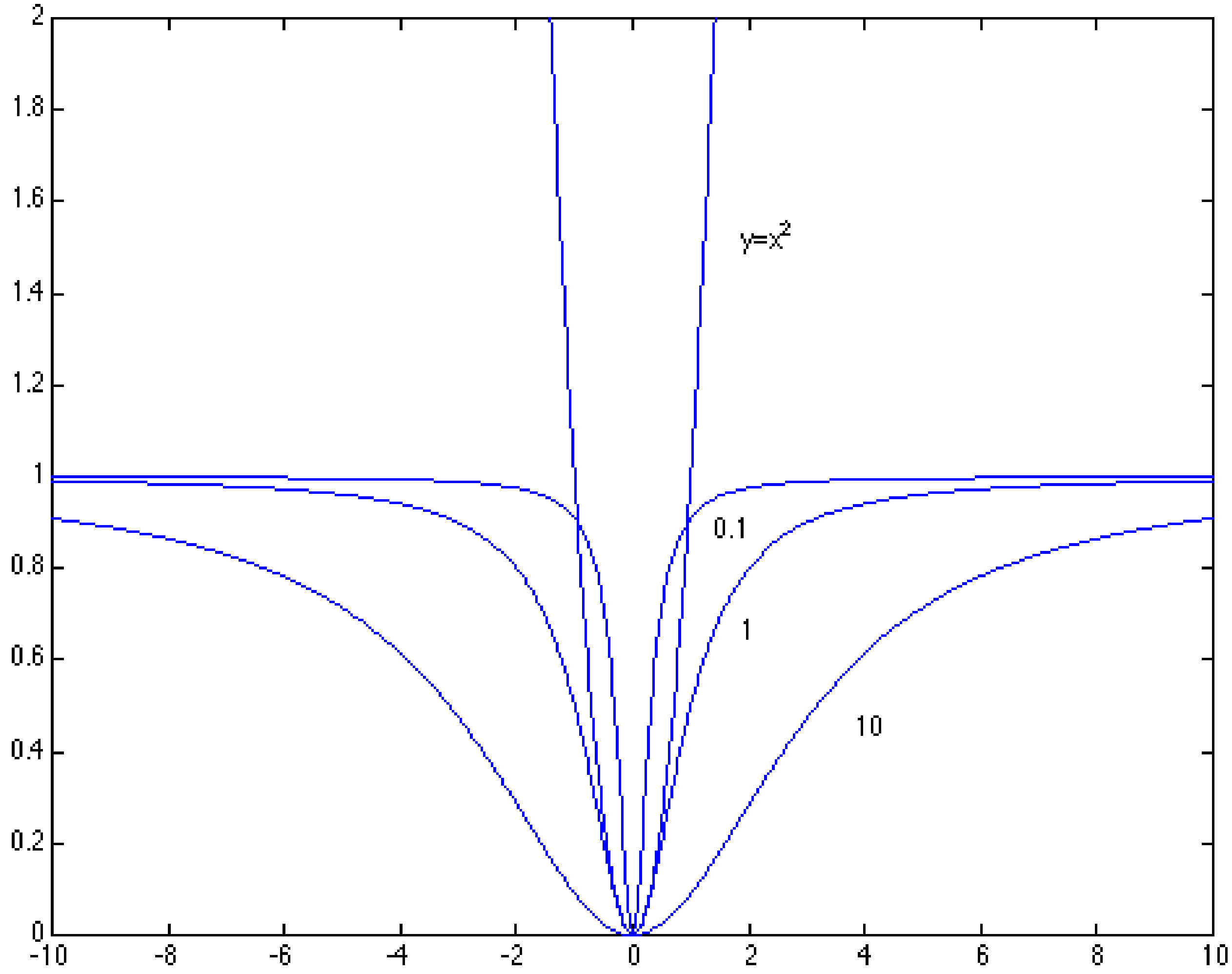

6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

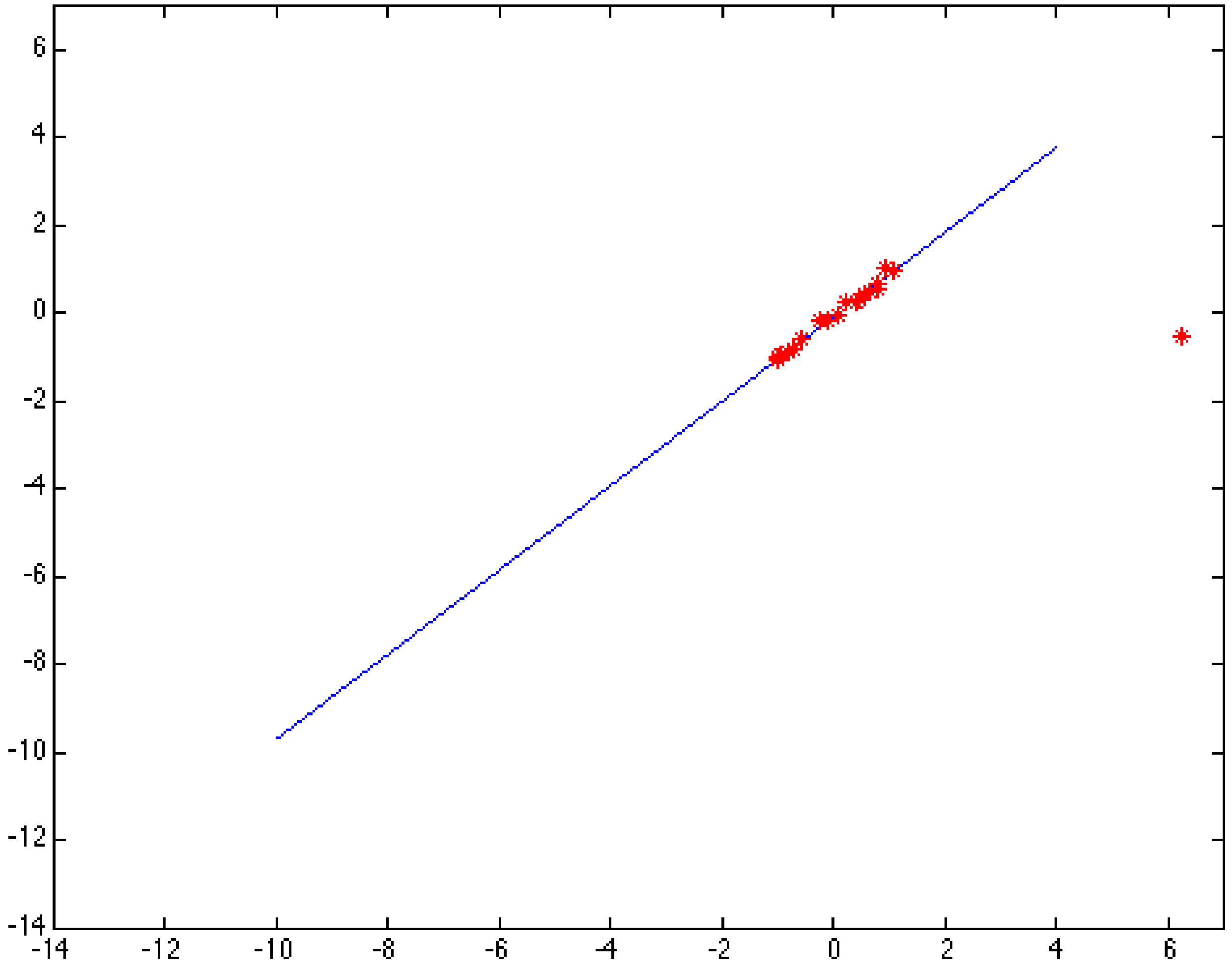

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

LS Error Function vs. Robust Error Function

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

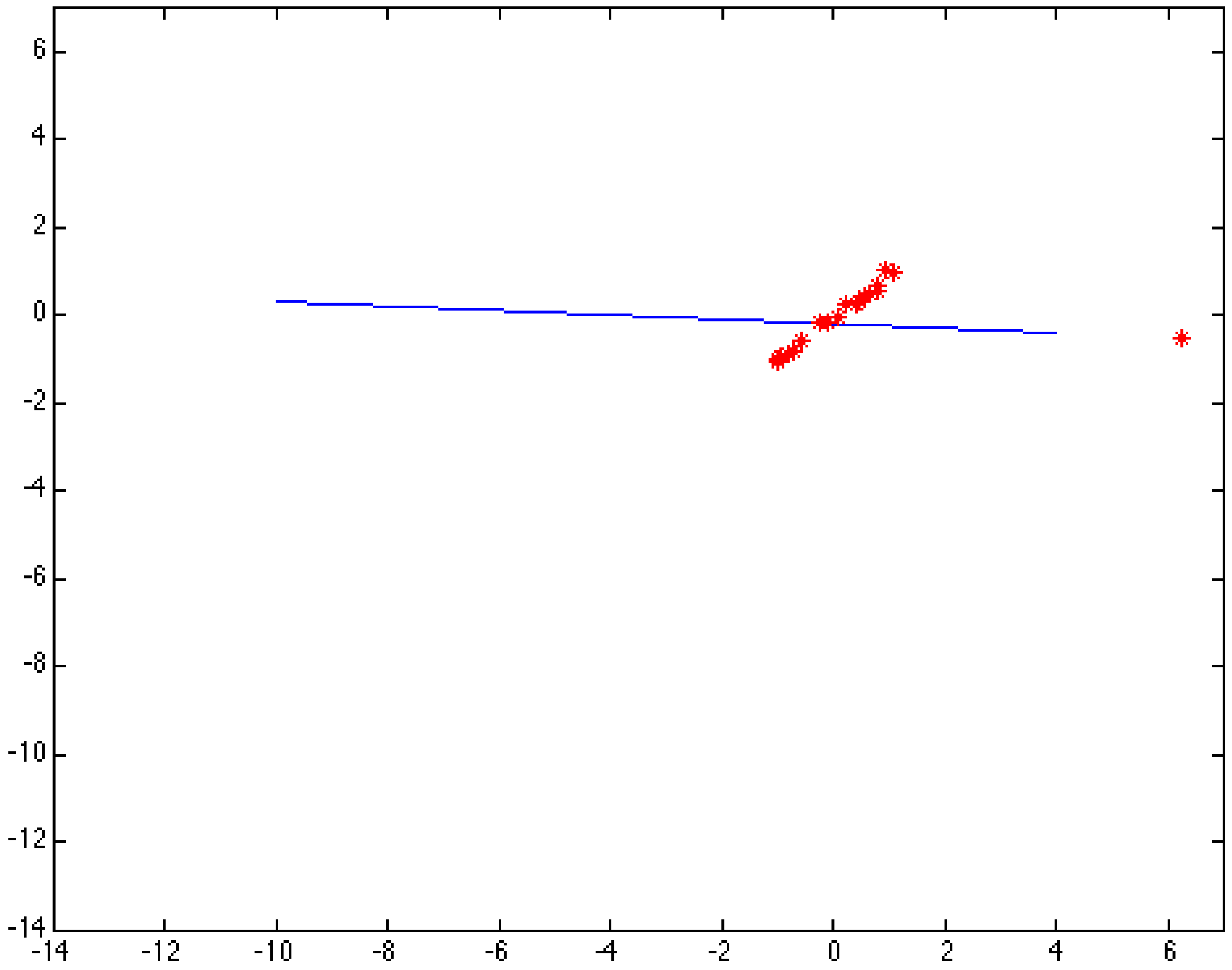


Line Fitting with Robust Error Function



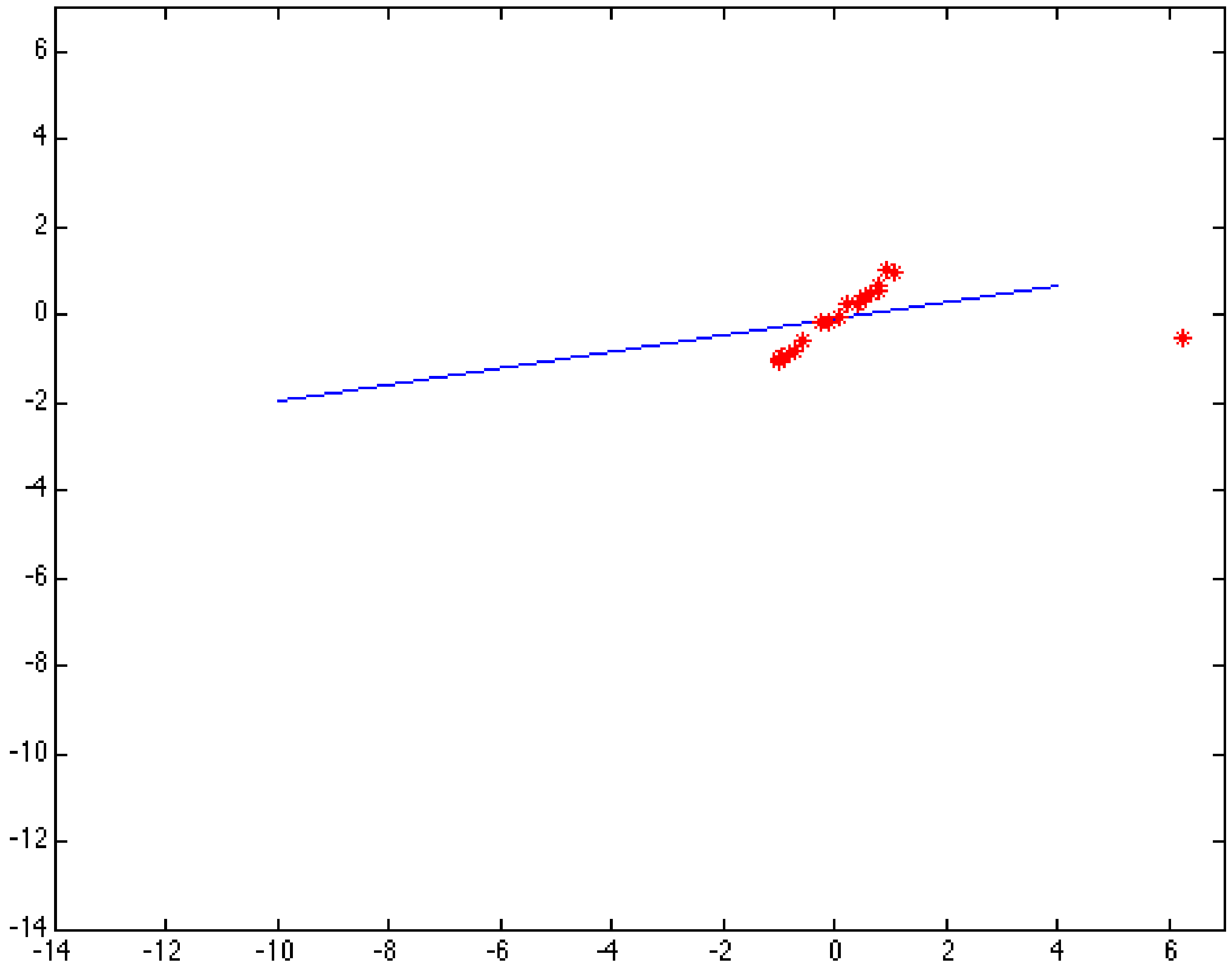
Line Fitting with Robust Error Function

Too small scale parameter σ



Line Fitting with Robust Error Function

Too large scale parameter σ



M-Estimator

An **M-estimator** estimates parameters by minimizing an expression of the form

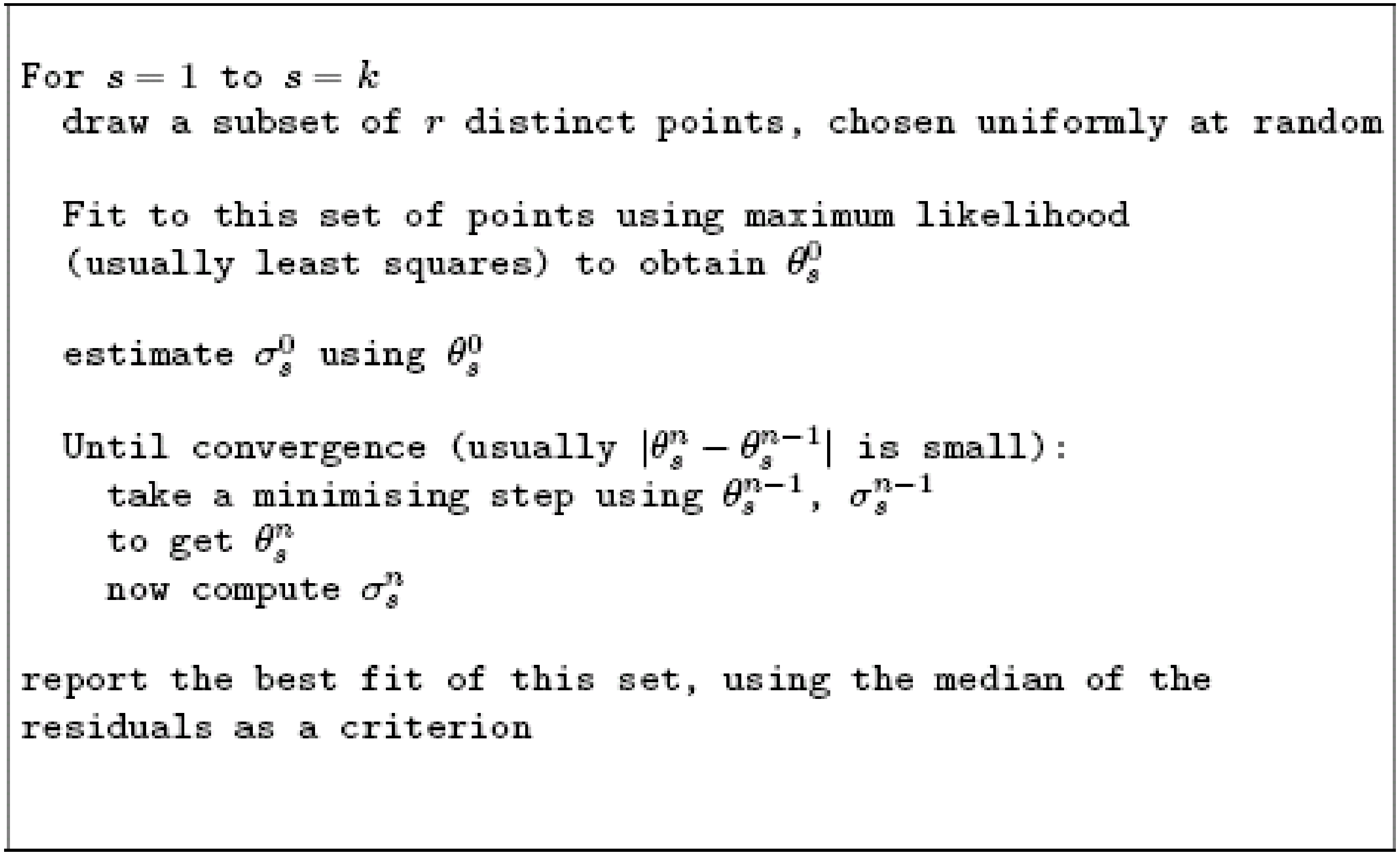
$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma)$$

where θ are the parameters of the model being fitted and $r_i(\boldsymbol{x}_i, \theta)$ is the residual

$$\sum_i \rho(r_i(\boldsymbol{x}_i, \theta); \sigma) \frac{\partial \rho}{\partial \theta} = 0$$

Solve the equation by **iteratively re-weighted least squares** estimation.

M-Estimator



```
For $s = 1$ to $s = k$
  draw a subset of $r$ distinct points, chosen uniformly at random

  Fit to this set of points using maximum likelihood
  (usually least squares) to obtain $\theta_s^0$

  estimate $\sigma_s^0$ using $\theta_s^0$

  Until convergence (usually $|\theta_s^n - \theta_s^{n-1}|$ is small):
    take a minimising step using $\theta_s^{n-1}$, $\sigma_s^{n-1}$
    to get $\theta_s^n$
    now compute $\sigma_s^n$

report the best fit of this set, using the median of the
residuals as a criterion
```

Algorithm 17.3: *Using an M-estimator to fit a probabilistic model*

RANSAC

Random Sample Consensus

- Choose a small subset uniformly at random
- Fit to that
- Anything that is close to result is signal; all others are noise
- Refit
- Do this many times and choose the best

- Issues
  - How many times?
    - Often enough that we are likely to have a good line
  - How big a subset?
    - Smallest possible
  - What does close mean?
    - Depends on the problem
  - What is a good line?
    - One with enough number of data points close to the model

RANSAC

- RANdom Sample Consensus
- Approach: we want to avoid the impact of outliers, so let’s look for “inliers”, and use those only.
- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won’t have much support from the rest of the points.

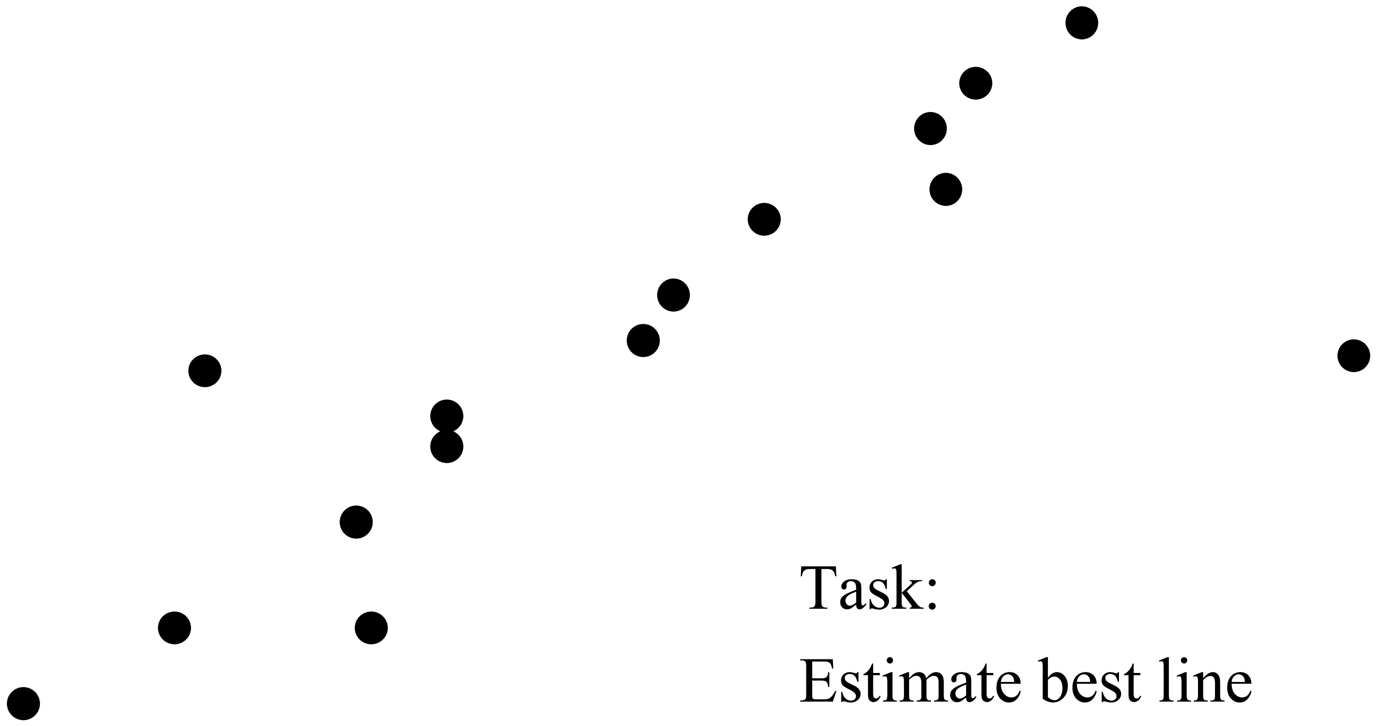
RANSAC

- RANSAC loop:

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

RANSAC Line Fitting Example

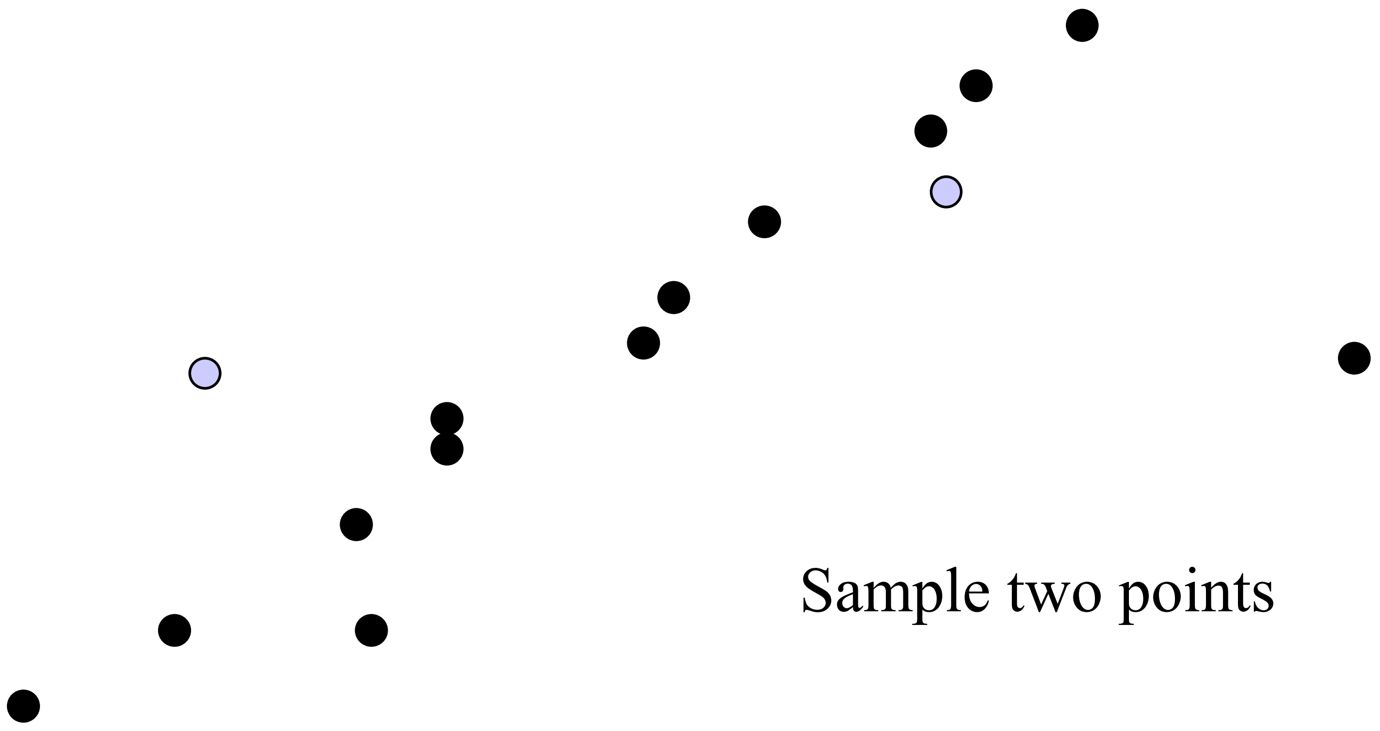


Task:

Estimate best line

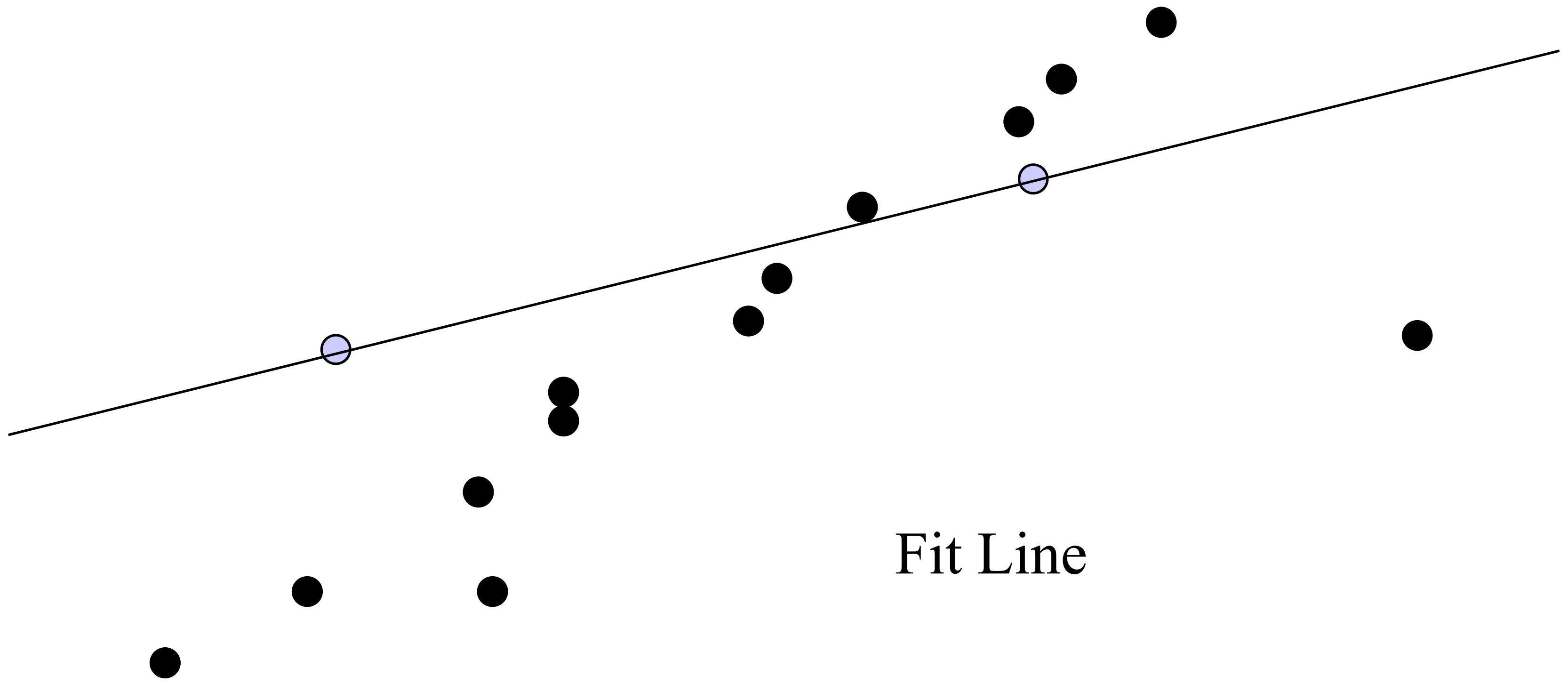
Slide credit: Jinxiang Chai, CMU

RANSAC Line Fitting Example



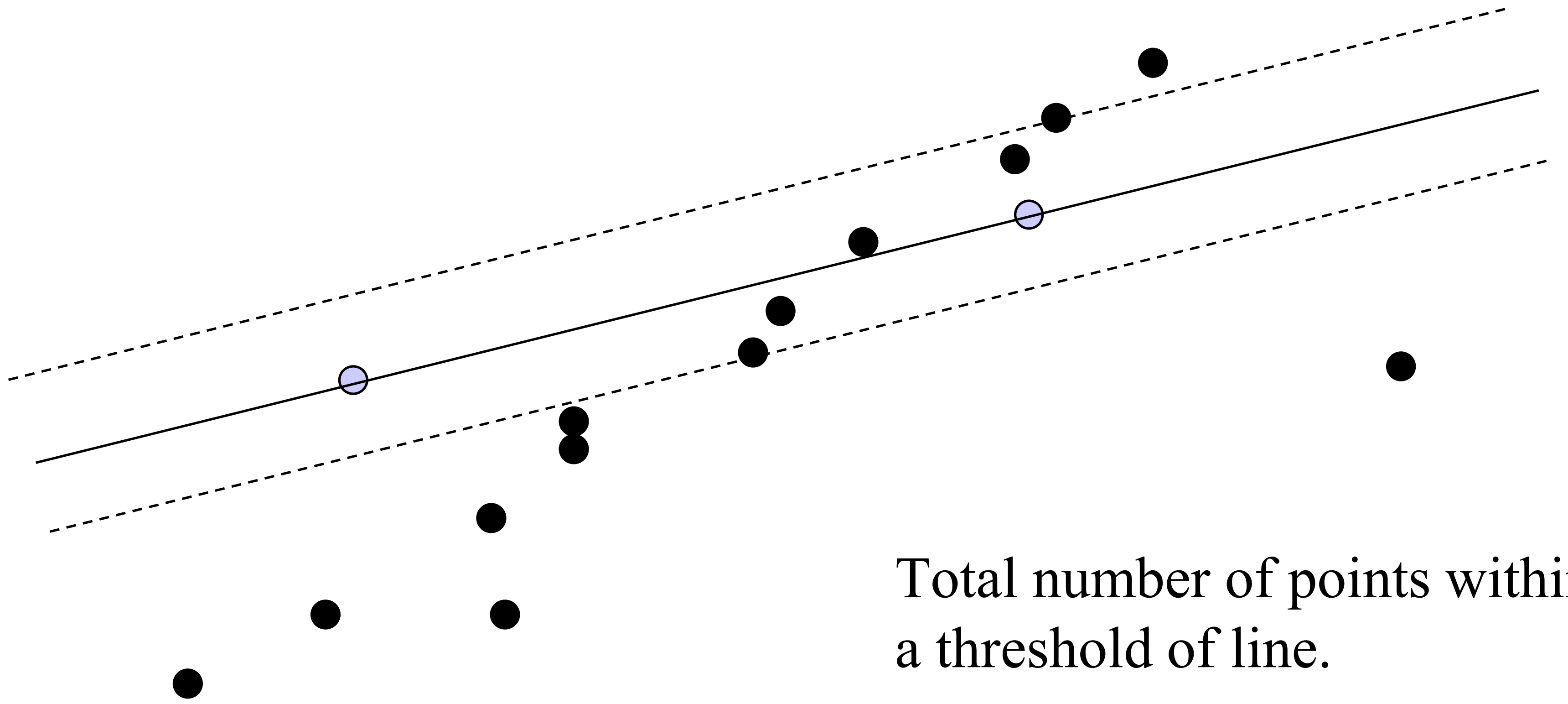
Sample two points

RANSAC Line Fitting Example



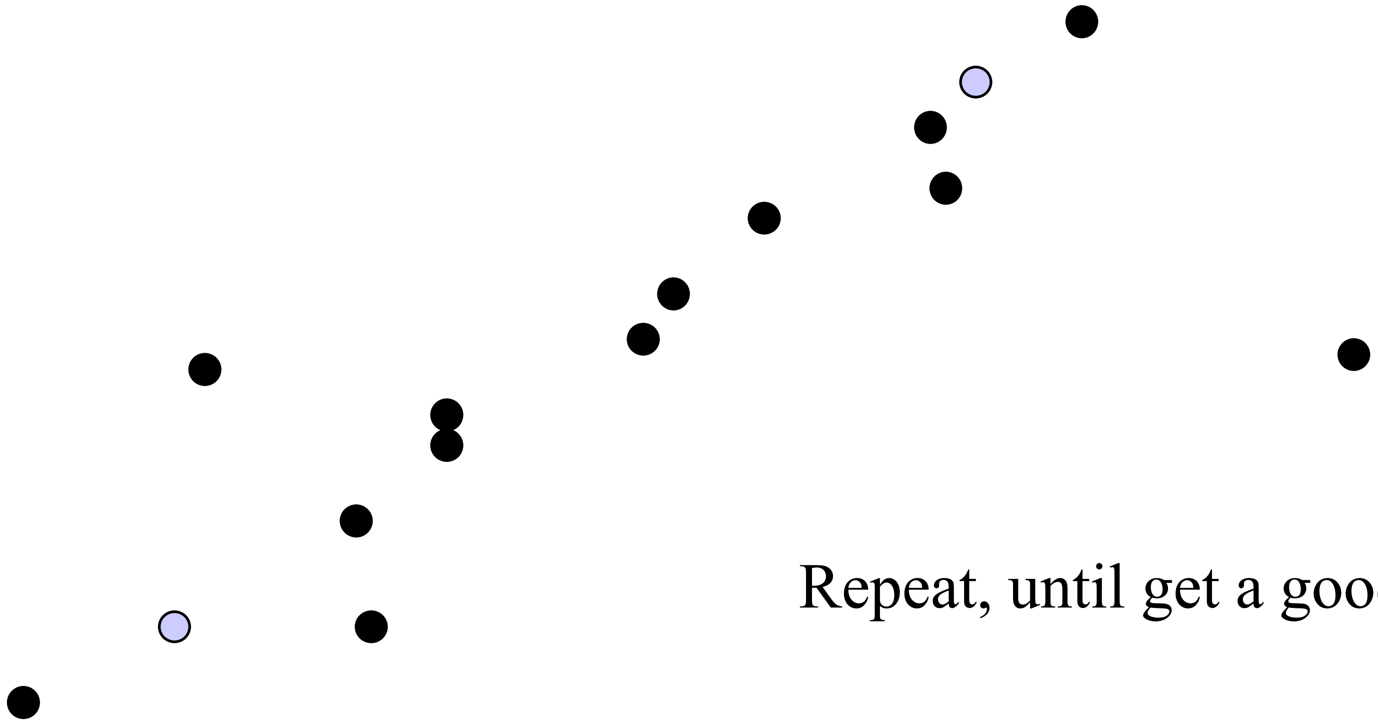
Fit Line

RANSAC Line Fitting Example



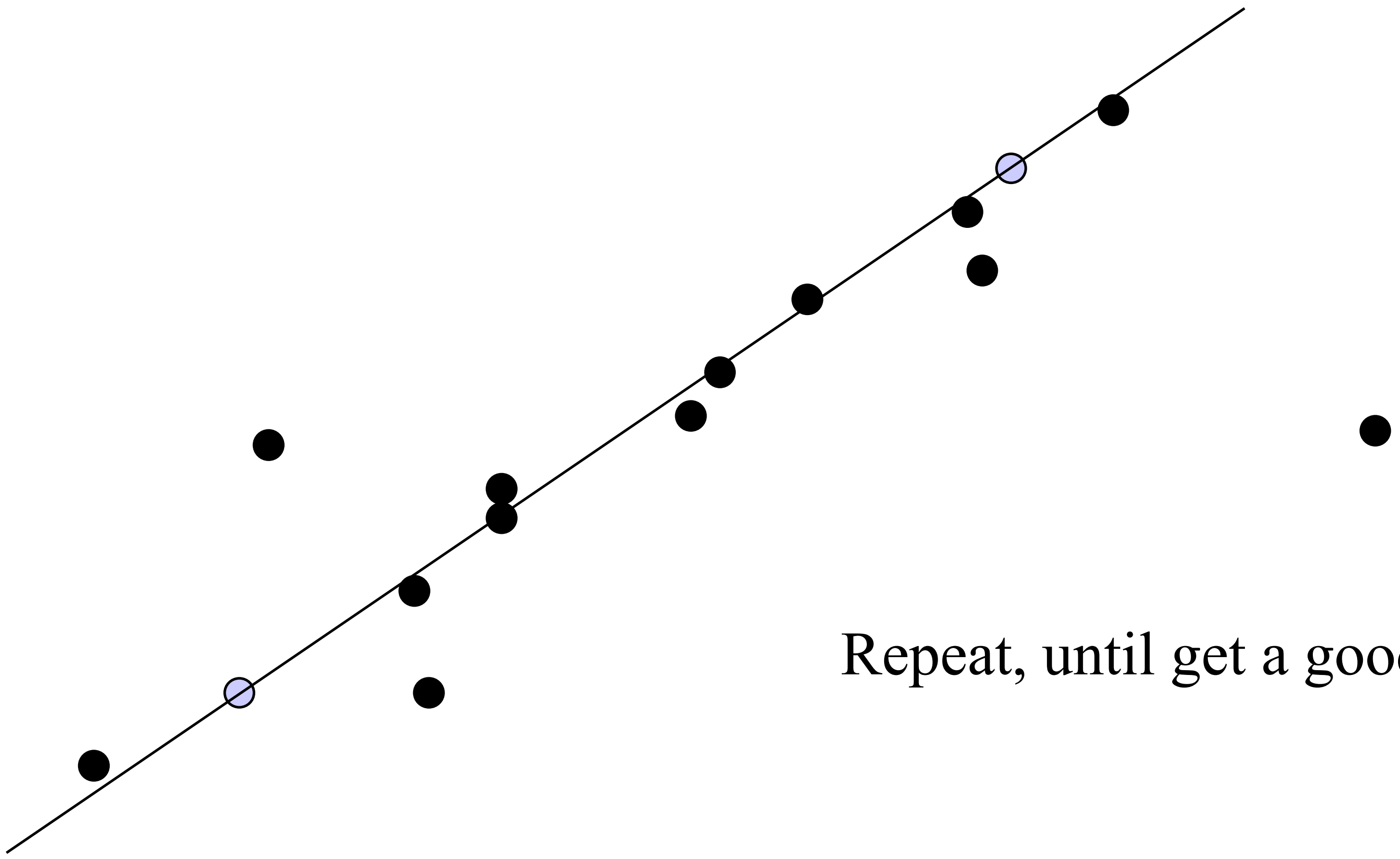
Total number of points within a threshold of line.

RANSAC Line Fitting Example



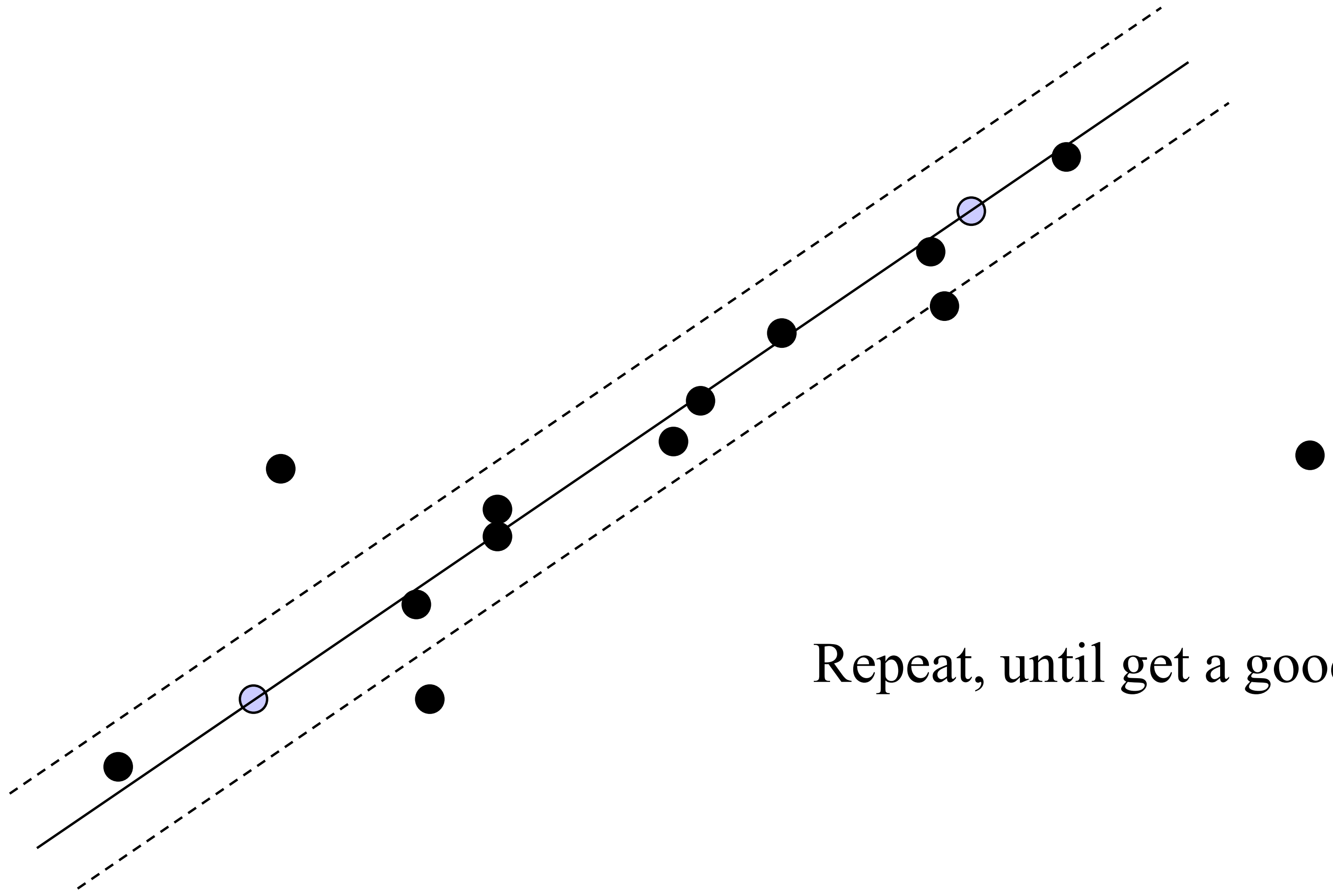
Repeat, until get a good result

RANSAC Line Fitting Example

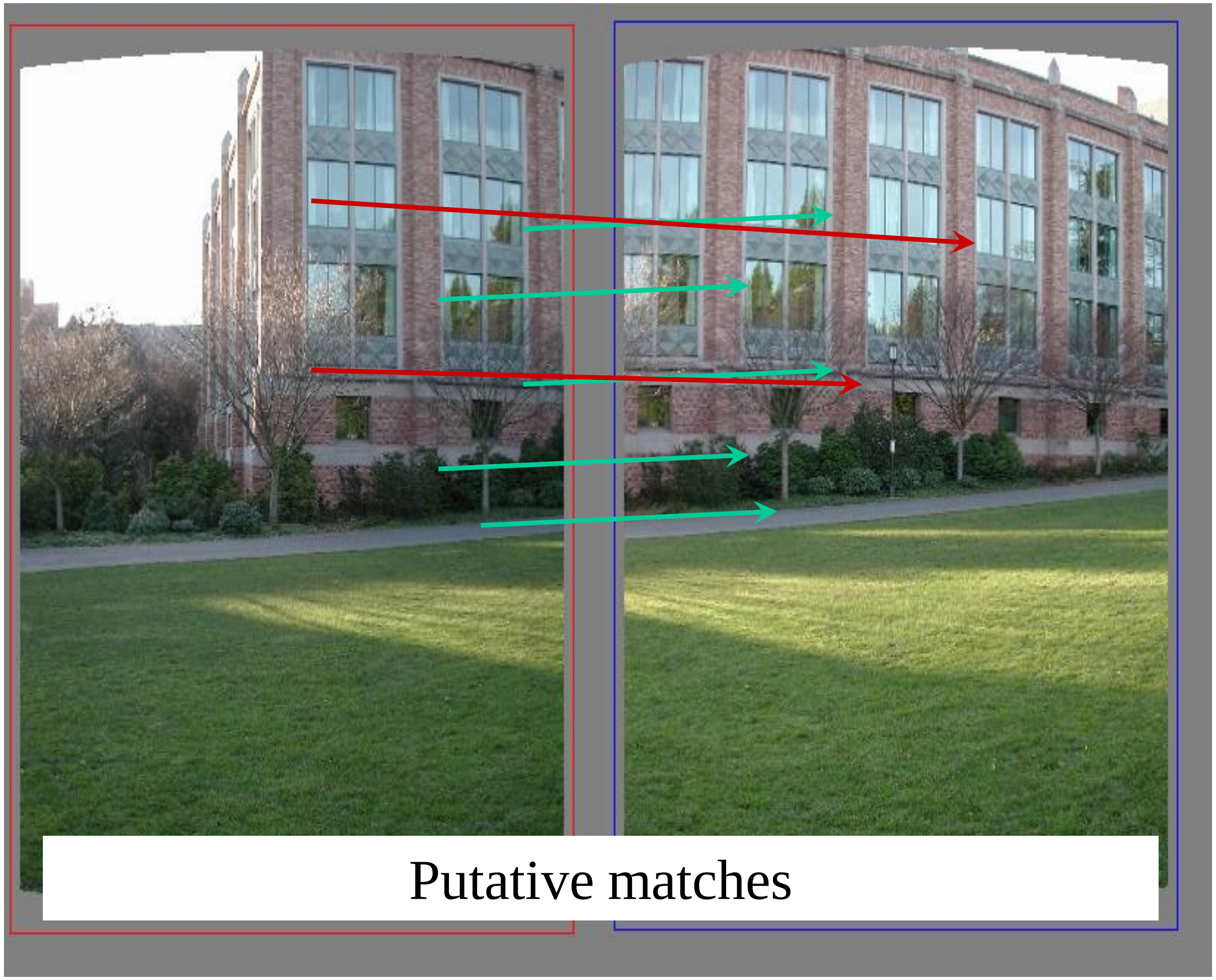


Repeat, until get a good result

RANSAC Line Fitting Example

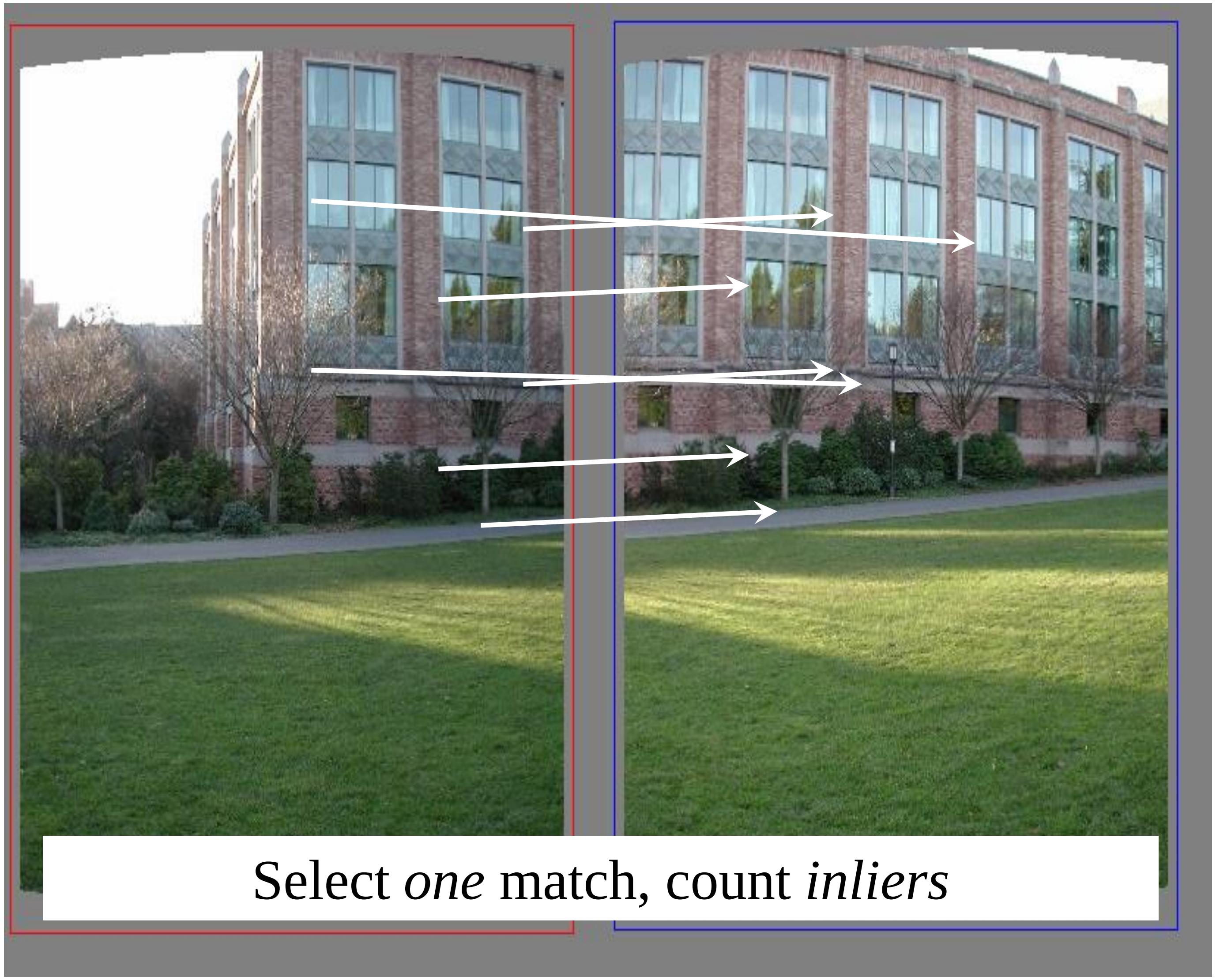


Repeat, until get a good result

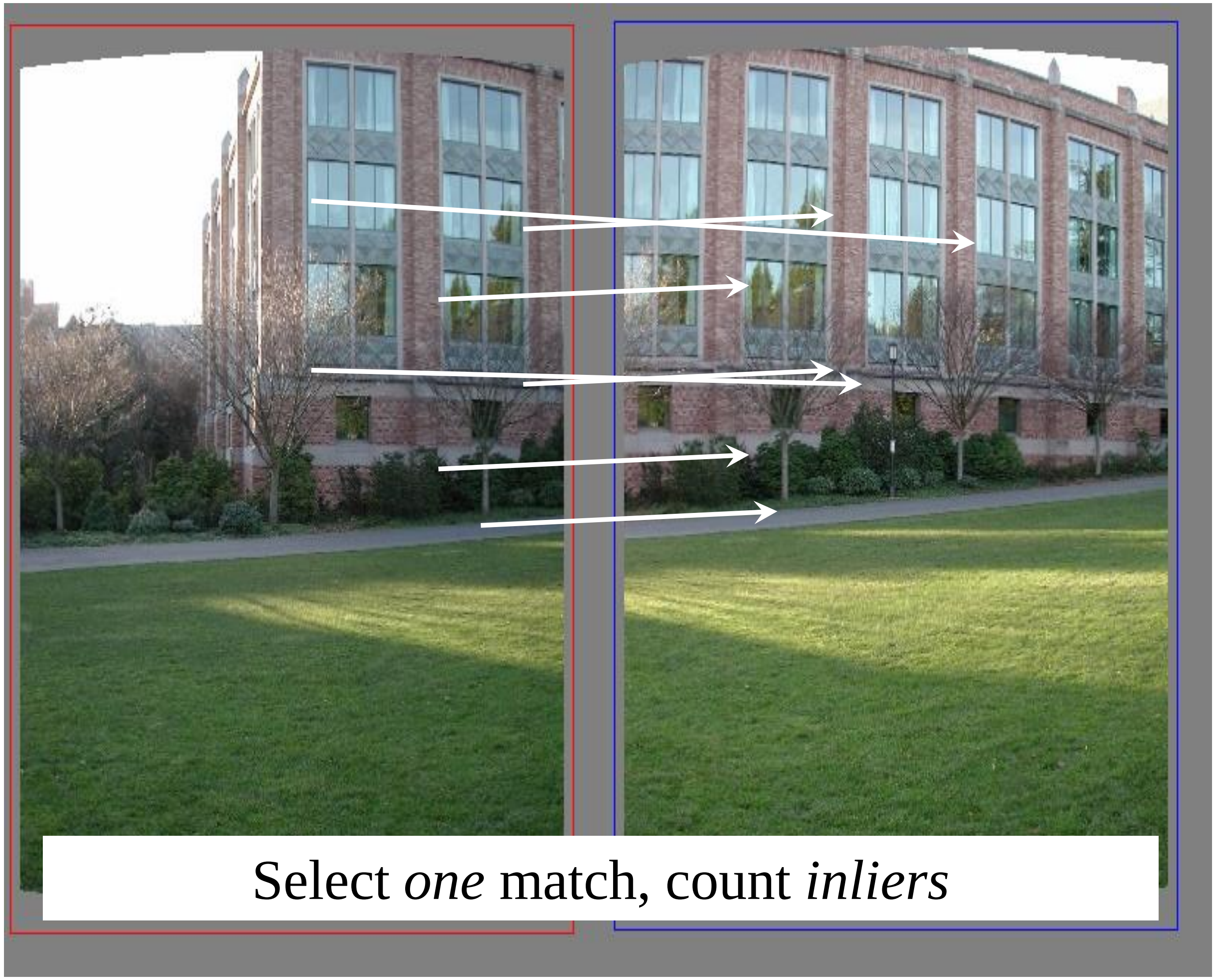


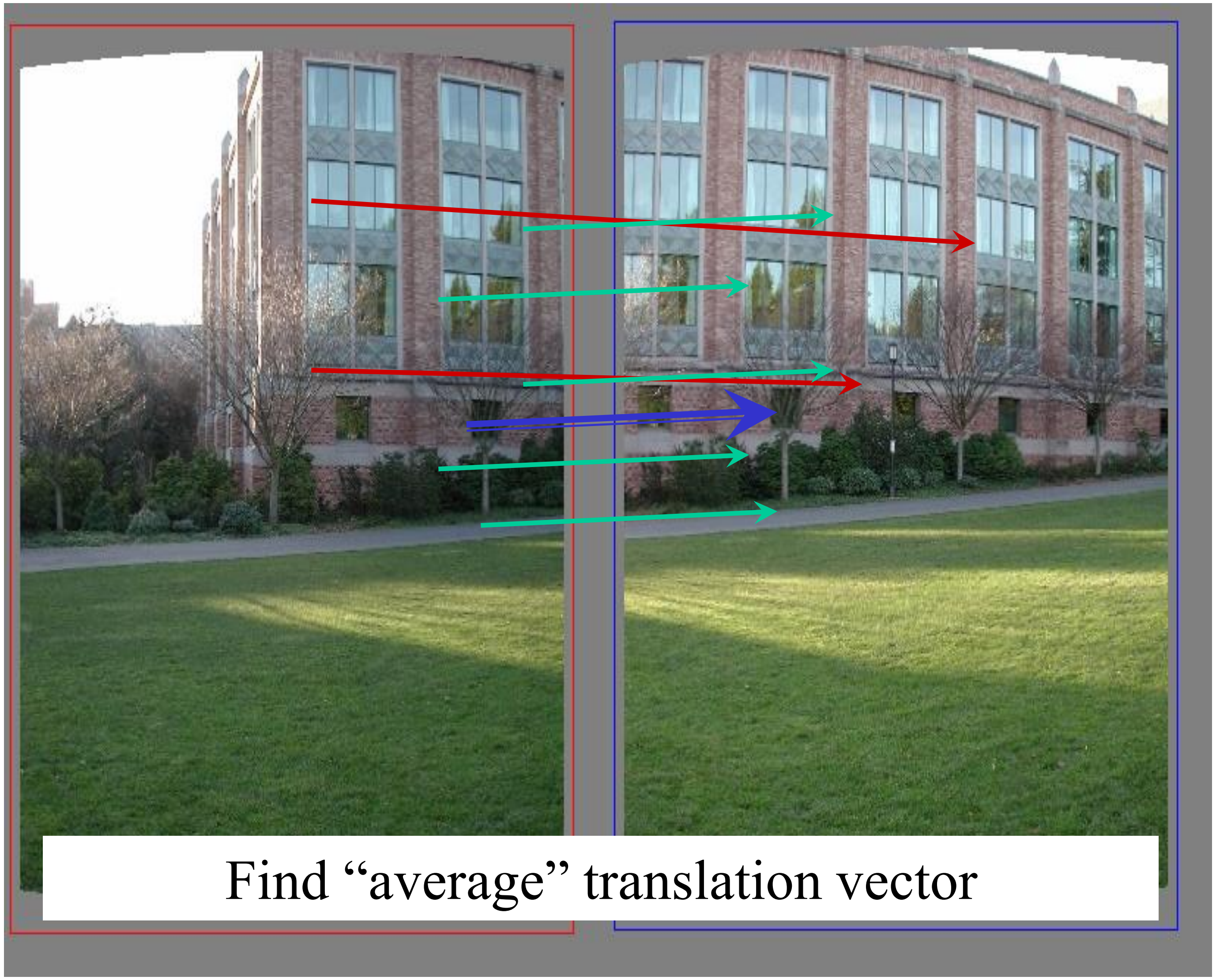
Putative matches

Source: Rick Szeliski



Select *one* match, count *inliers*


Select *one* match, count *inliers*

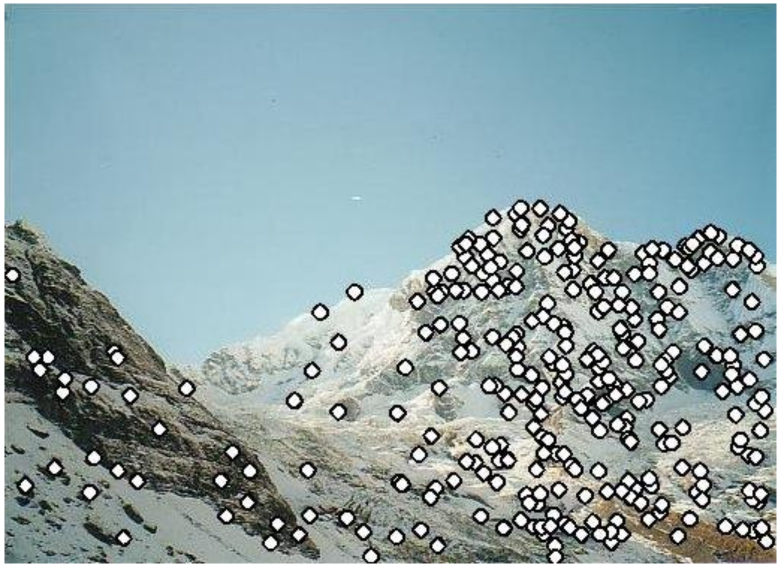


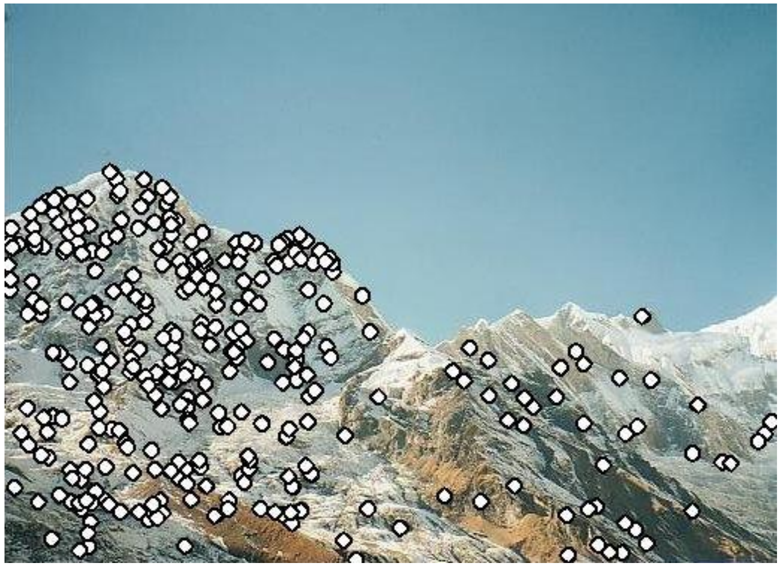
Find "average" translation vector

Feature-based alignment outline

Source: L. Lazebnik

Feature-based alignment outline

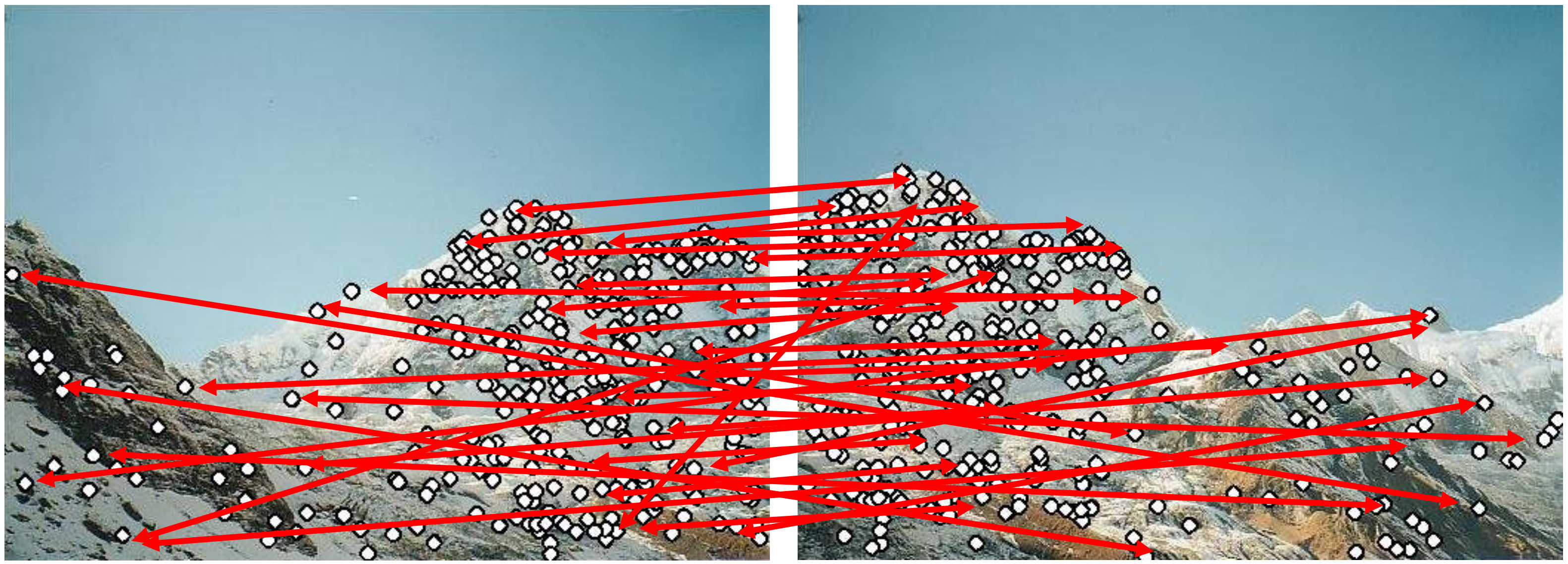




- Extract features

Source: L. Lazebnik

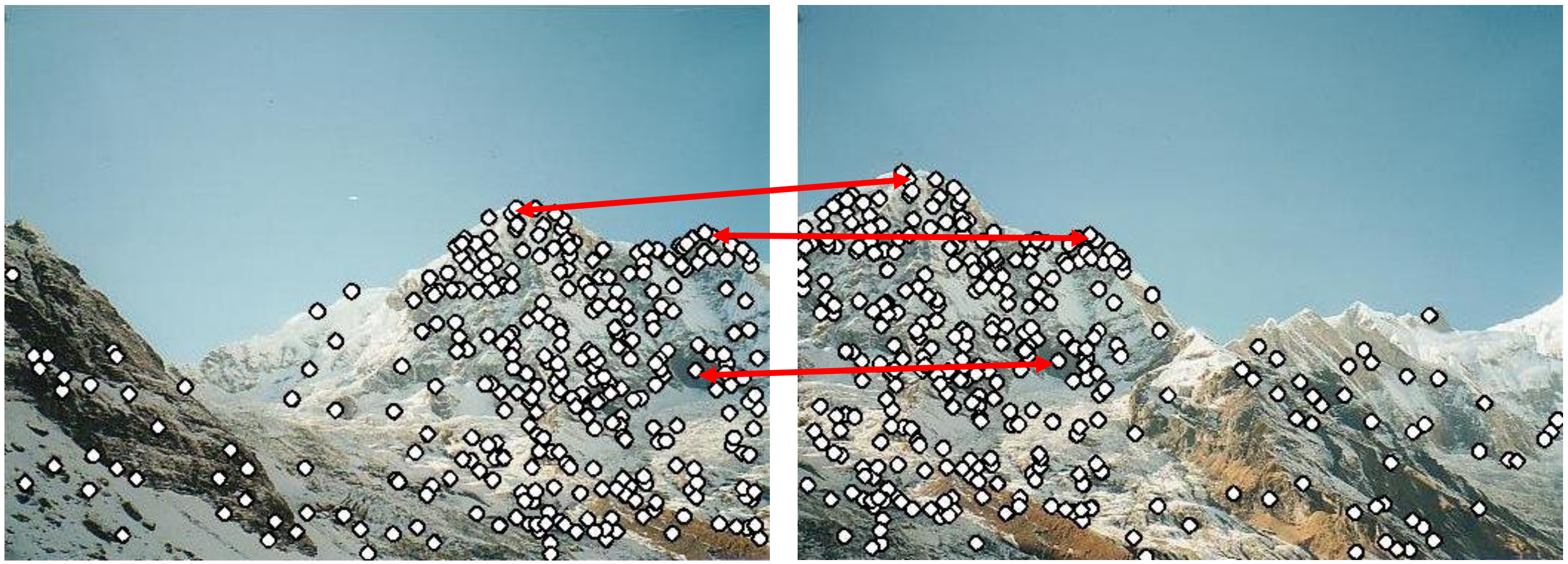
Feature-based alignment outline



- Extract features
- Compute *putative matches*

Source: L. Lazebnik

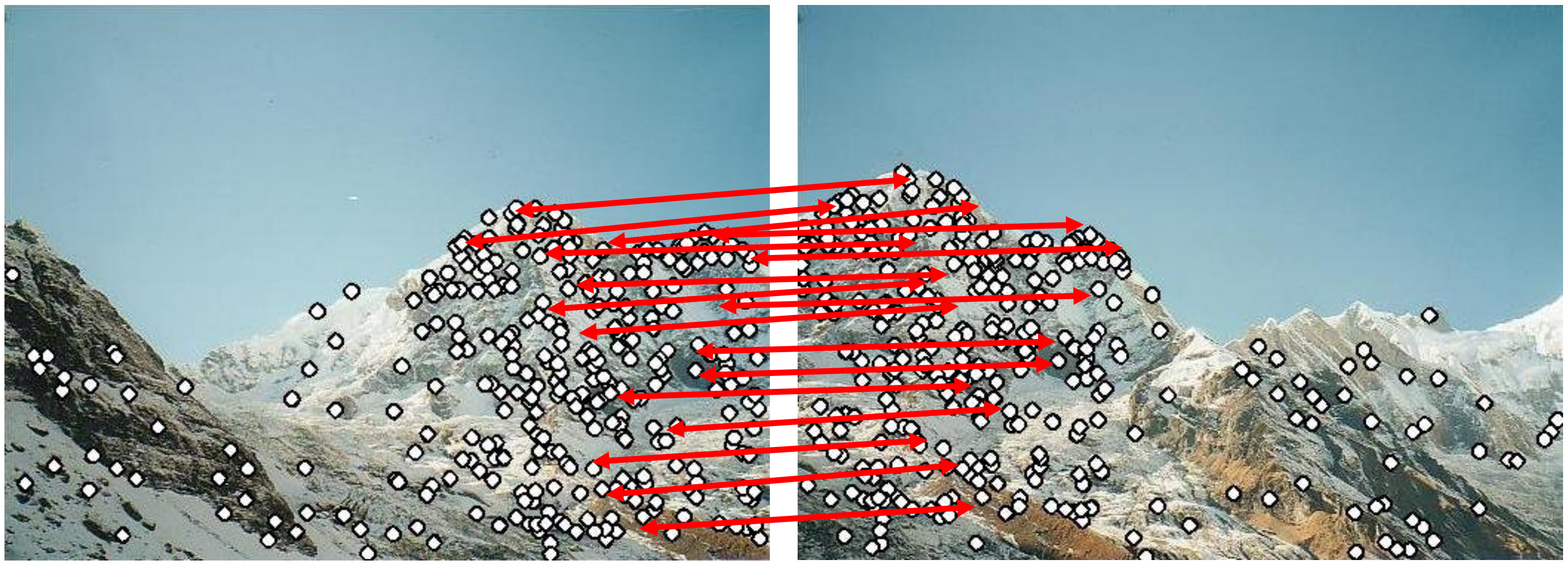
Feature-based alignment outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)

Source: L. Lazebnik

Feature-based alignment outline



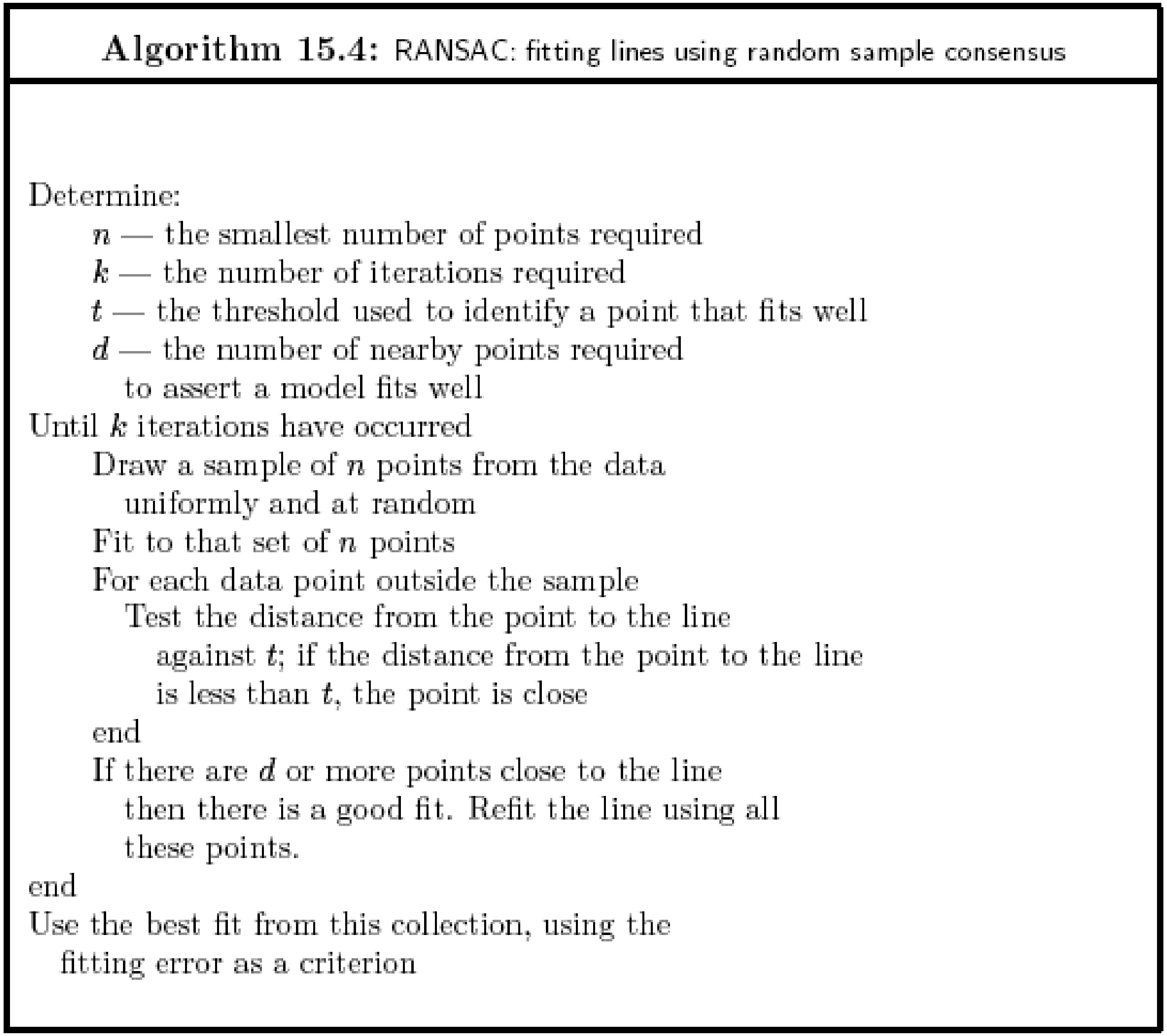
- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik

Feature-based alignment outline

- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

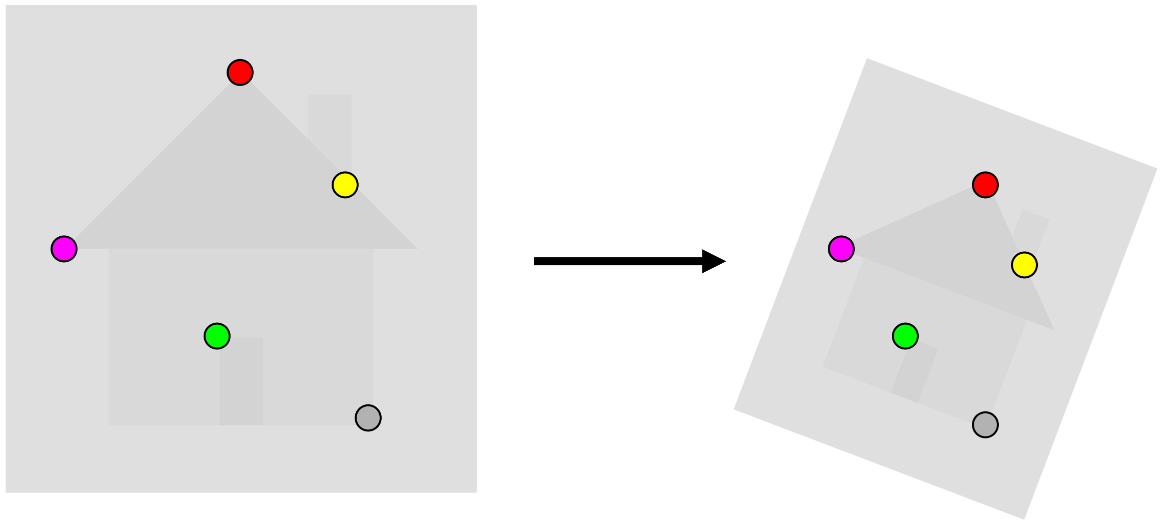
Source: L. Lazebnik



Algorithm 15.4: RANSAC: fitting lines using random sample consensus

```
Determine:
    $n$ — the smallest number of points required
    $k$ — the number of iterations required
    $t$ — the threshold used to identify a point that fits well
    $d$ — the number of nearby points required
      to assert a model fits well
Until $k$ iterations have occurred
    Draw a sample of $n$ points from the data
      uniformly and at random
    Fit to that set of $n$ points
    For each data point outside the sample
      Test the distance from the point to the line
        against $t$; if the distance from the point to the line
        is less than $t$, the point is close
    end
    If there are $d$ or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion
```

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

Source: L. Lazebnik

Image Alignment Using Least Squares

Given a set of matched feature points $\{(\mathbf{x}_i, \mathbf{x}'_i)\}$ and a planar parametric transformation[1] of the form

$$\mathbf{x}' = \mathbf{f}(\mathbf{x}; \mathbf{p}), \tag{8.1}$$

how can we produce the best estimate of the motion parameters $\mathbf{p}$? The usual way to do this is to use least squares, i.e., to minimize the sum of squared residuals

$$E_{\mathrm{LS}} = \sum_i \|\mathbf{r}_i\|^2 = \sum_i \|\mathbf{f}(\mathbf{x}_i; \mathbf{p}) - \mathbf{x}'_i\|^2, \tag{8.2}$$

where

$$\mathbf{r}_i = \mathbf{x}'_i - \mathbf{f}(\mathbf{x}_i; \mathbf{p}) = \hat{\mathbf{x}}'_i - \tilde{\mathbf{x}}'_i \tag{8.3}$$

is the *residual* between the measured location $\hat{\mathbf{x}}'_i$ and its corresponding current *predicted* location $\tilde{\mathbf{x}}'_i = \mathbf{f}(\mathbf{x}_i; \mathbf{p})$.

LS Image Alignment

- Many of the motion models have a linear relationship between the amount of motion Δx = x’ − x and the unknown motion parameters **p**,

$$\Delta\mathbf{x} = \mathbf{x}' - \mathbf{x} = \mathbf{J}(\mathbf{x})\mathbf{p}, \tag{8.4}$$

where **J** = ∂f/∂**p** is the Jacobian of transformation f w.r.t motion parameters **p**.

- Least-Square Formulation:

$$\begin{aligned} E_{\text{LLS}} &= \sum_i \|\mathbf{J}(\mathbf{x}_i)\mathbf{p} - \Delta\mathbf{x}_i\|^2 & (8.5) \\ &= \mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i)\right] \mathbf{p} - 2\mathbf{p}^T \left[\sum_i \mathbf{J}^T(\mathbf{x}_i)\Delta\mathbf{x}_i\right] + \sum_i \|\Delta\mathbf{x}_i\|^2 & (8.6) \\ &= \mathbf{p}^T\mathbf{A}\mathbf{p} - 2\mathbf{p}^T\mathbf{b} + c. & (8.7) \end{aligned}$$

The minimum can be found by solving the symmetric positive definite (SPD) system of *normal equations*[2]

$$\mathbf{A}\mathbf{p} = \mathbf{b}, \tag{8.8}$$

where

$$\mathbf{A} = \sum_i \mathbf{J}^T(\mathbf{x}_i)\mathbf{J}(\mathbf{x}_i) \tag{8.9}$$

Global Transforms for Image Alignment

Transform	**Matrix**	**Parameters p**	**Jacobian J**
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 8.1.3)

Jacobians of the 2D coordinate transformations x' = f(x; p)

2D Planar Transformation for Image Alignment

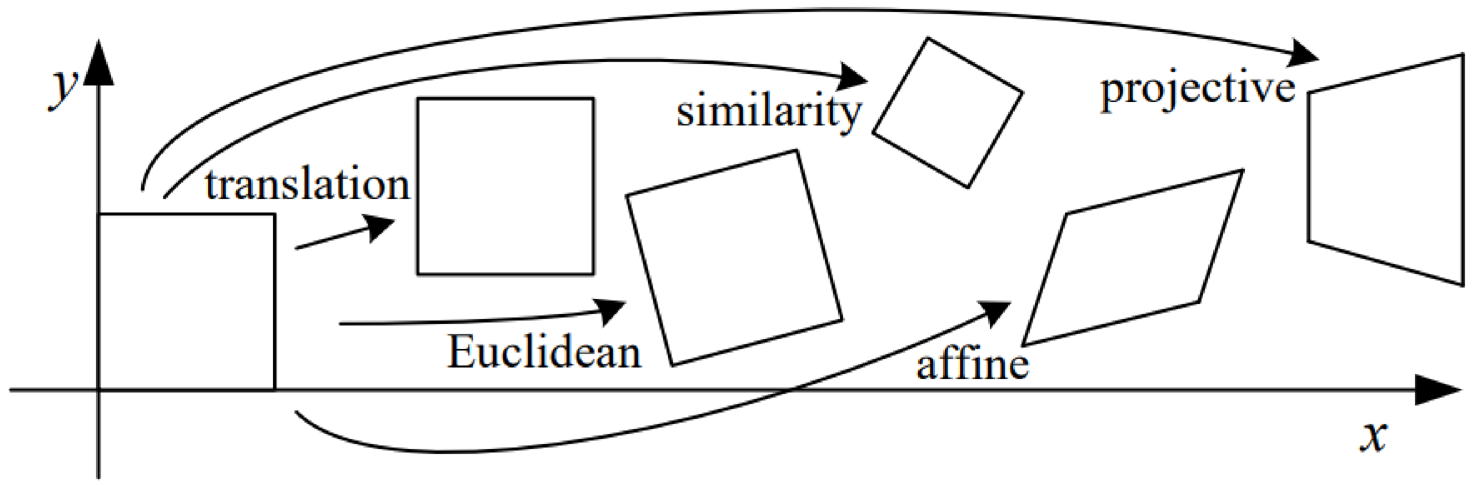
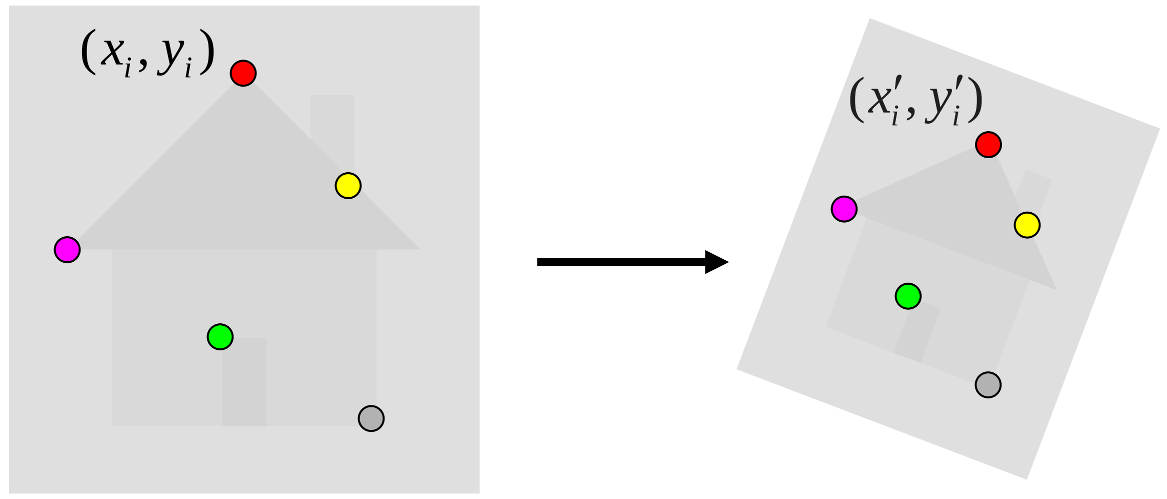


Figure 8.2 *Basic set of 2D planar transformations*

Fitting an affine transformation

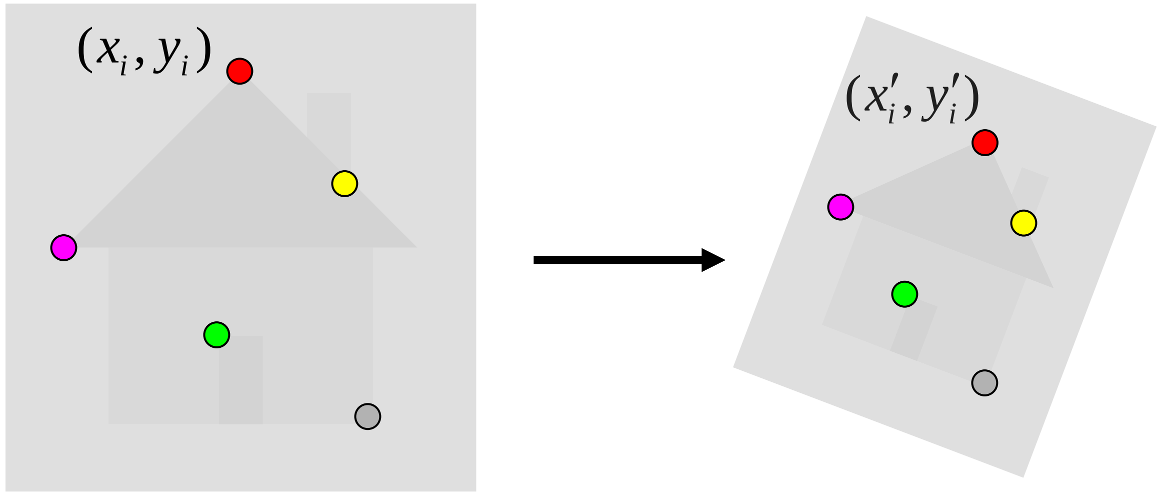
- Assuming we know the correspondences, how do we get the transformation?



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

Fitting an affine transformation

- Assuming we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x'_i \\ y'_i \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

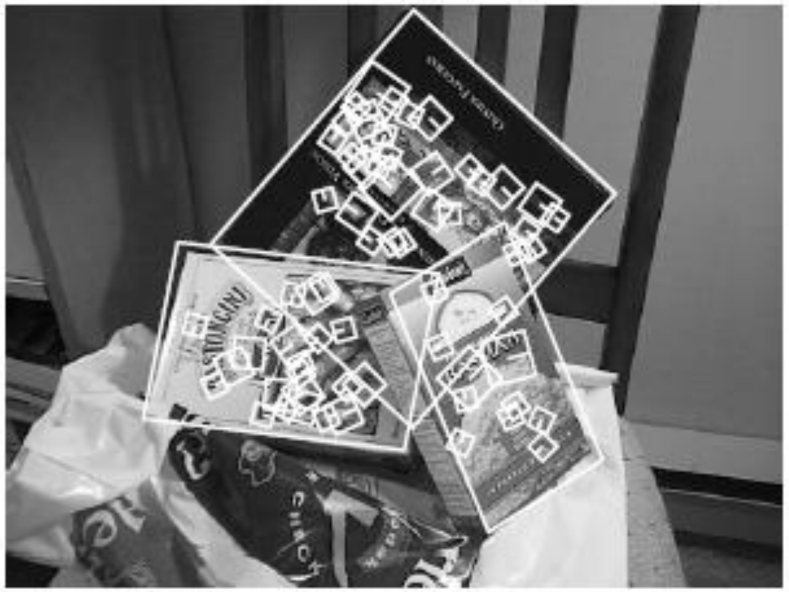
$$\begin{bmatrix} \quad \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

Fitting an Affine Transformation

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

- How many matches (correspondence pairs) do we need to solve for the transformation parameters?
- Once we have solved for the parameters, how do we compute the coordinates of the corresponding point for (x_{new}, y_{new})?
- How to deal with outliers in point correspondences?
  - M-estimator
  - RANSAC

Fitting an Affine Transformation



Affine model approximates perspective projection of planar objects.

Figures from David Lowe, ICCV 1999

Applications of Image Alignment

Image Stitching

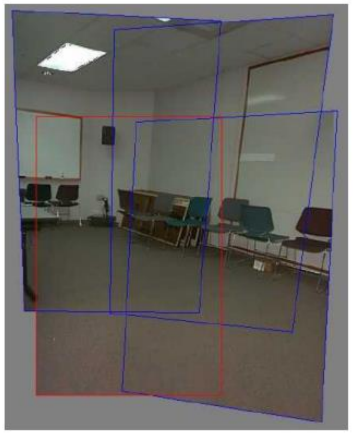


Image Panorama

Conclusion

- Finding lines and other parameterized objects, such as circles or ellipses, is an important task for computer vision.
- Success rate depends directly upon the noise in the edge image.
- The concept of Hough transform can be extended for robust model fitting.
- M-estimator is an iteratively reweighted least squares solution for model fitting
- RANSAC is a random sampling approach for model fitting under high percentages of outliers
- Robust image alignment is very crucial for image stitching and multi-image panorama generation applications.