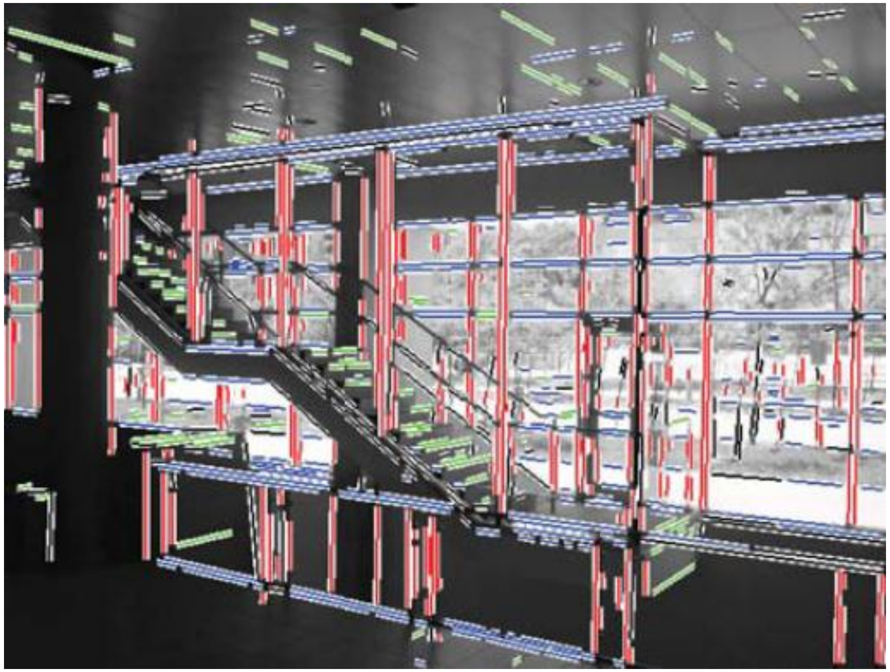
CS 4495 Computer Vision
RANdom **SA**mple **C**onsensus

Aaron Bobick

School of Interactive Computing

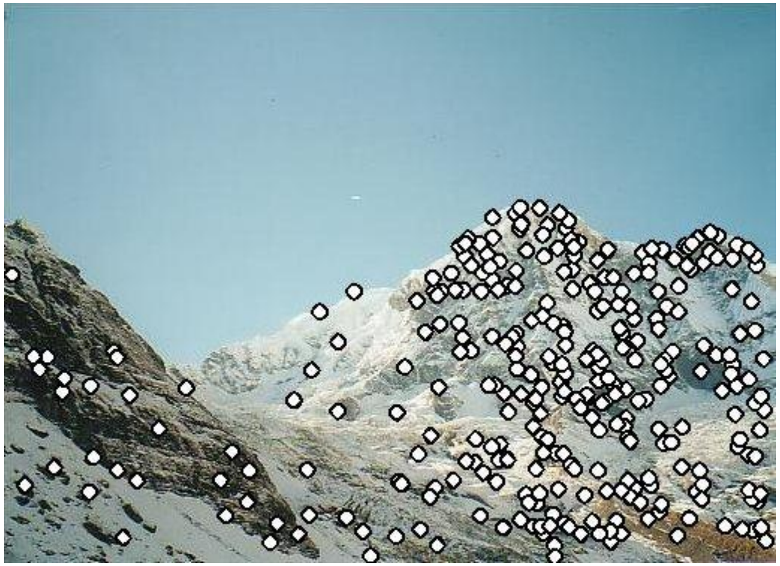


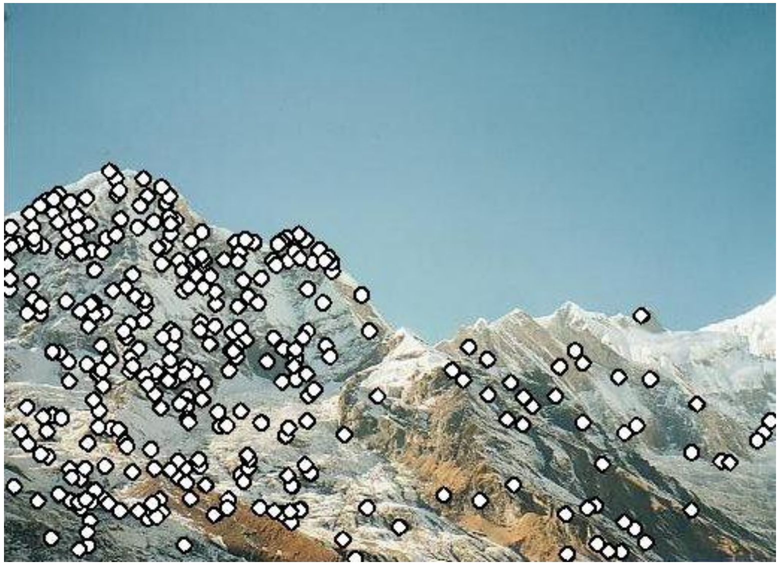
Administrivia

- PS 3:
  - Check Piazza - learn about least squares solutions and pseudo inverses
  - For 1.3, instead of using 4, 8 and 16 points and looking at the average residuals of 5 points, use 4,8, and 15 – since there are only 20 points!!
  - Due TUES October 18. Don't blow it!
- PS 4 will be out the 20th and due the 30th.

Matching with Features

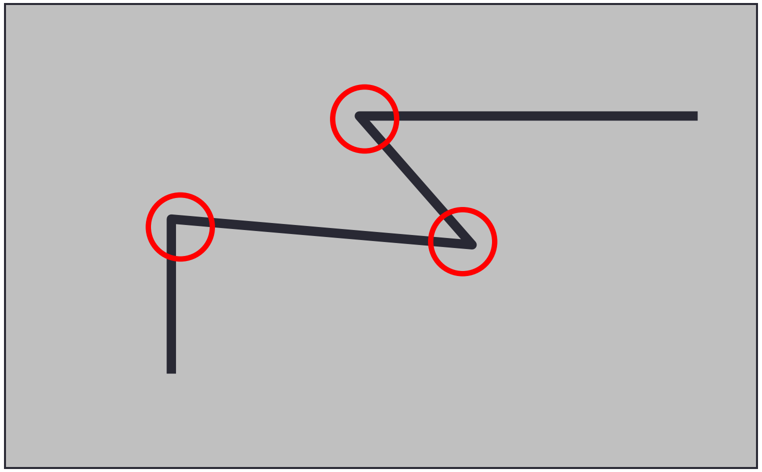
- Detect feature points in both images





An introductory example:

Harris corner detector



C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988

Harris Detector: Mathematics

$$M = A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$

Measure of corner response:

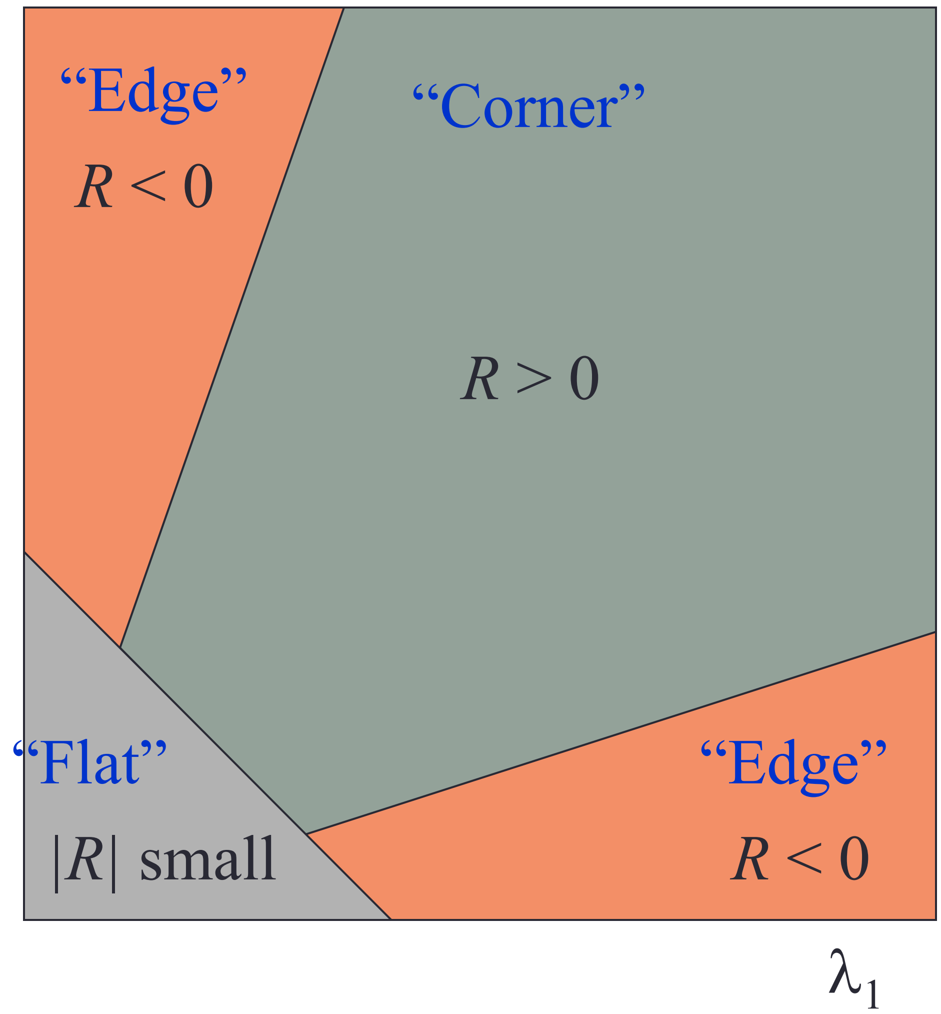
$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

(k – empirical constant, k = 0.04-0.06)

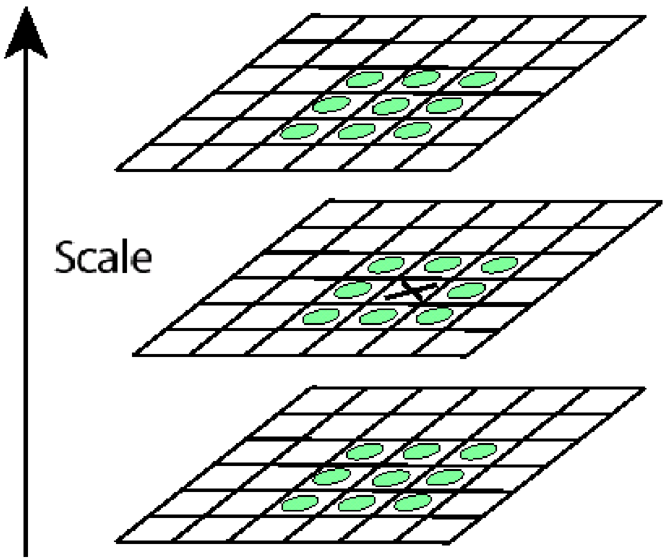
Harris Detector: Mathematics

- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



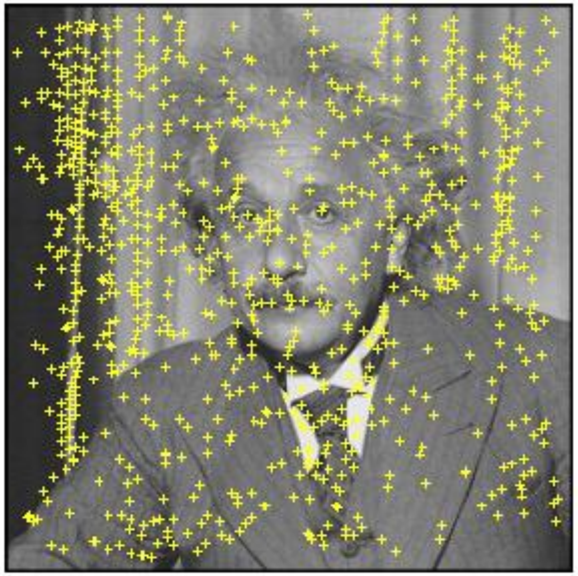
Key point localization

- General idea: find robust extremum (maximum or minimum) both in space and in scale.
- SIFT specific suggestion: use DoG pyramid to find maximum values (remember edge detection?) – then eliminate "edges" and pick only corners.
- More recent: use Harris detector to find maximums in space and then look at the Laplacian pyramid (we'll do this later) for maximum in scale.

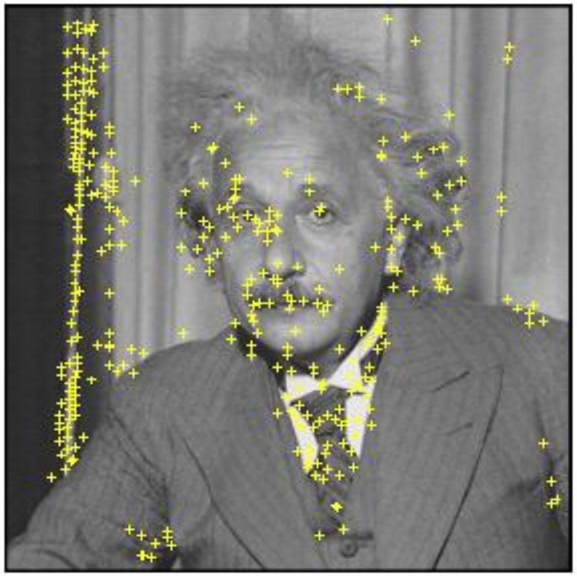


Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below.

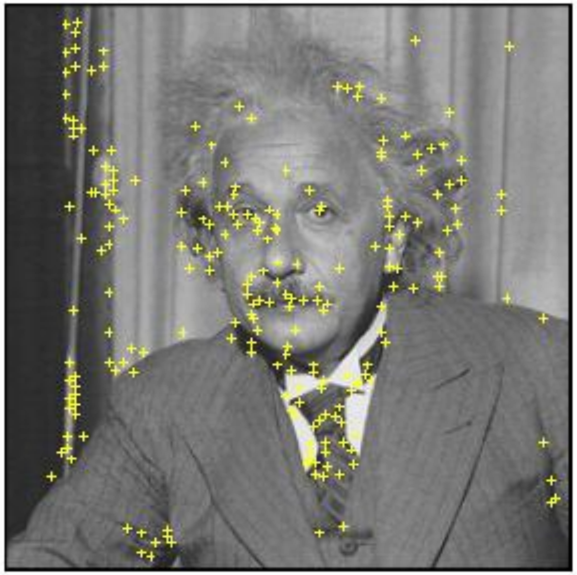
Remove low contrast, edge bound



Extrema points



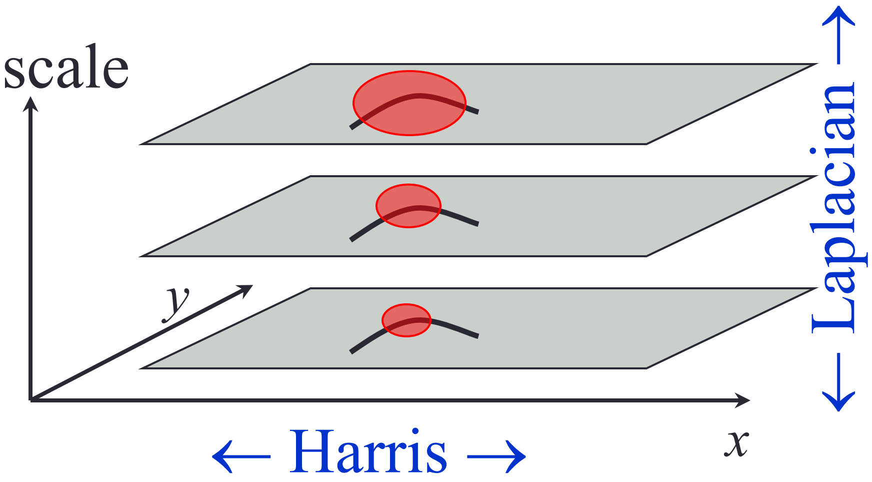
Contrast > C



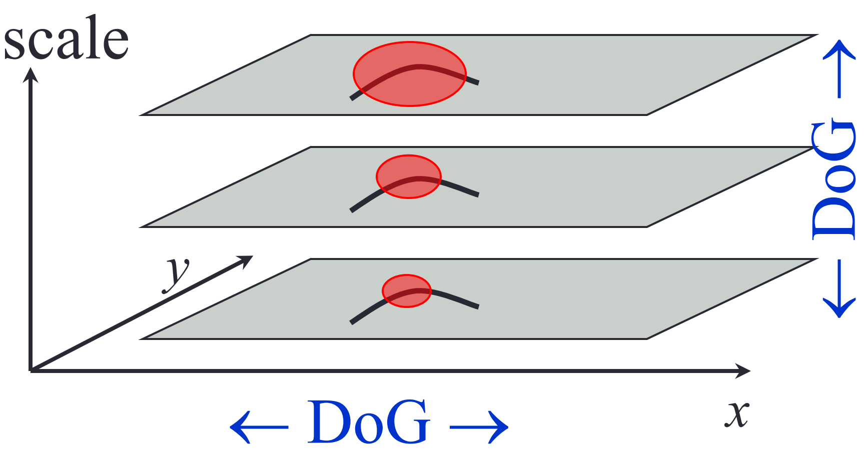
Not only on edge

Scale Invariant Detectors

- Harris-Laplacian[1]
  Find local maximum of:
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale



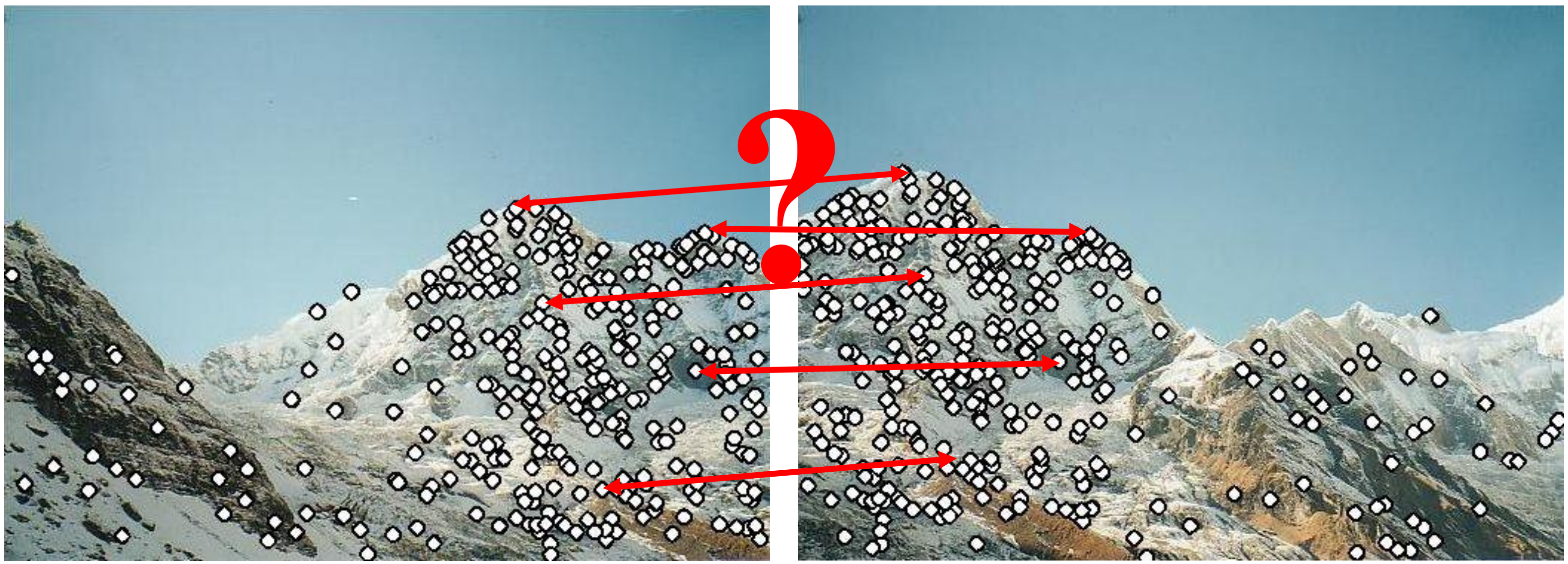
- SIFT (Lowe)[2]
  Find local maximum of:
  – Difference of Gaussians in space and scale



[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Accepted to IJCV 2004

Point Descriptors

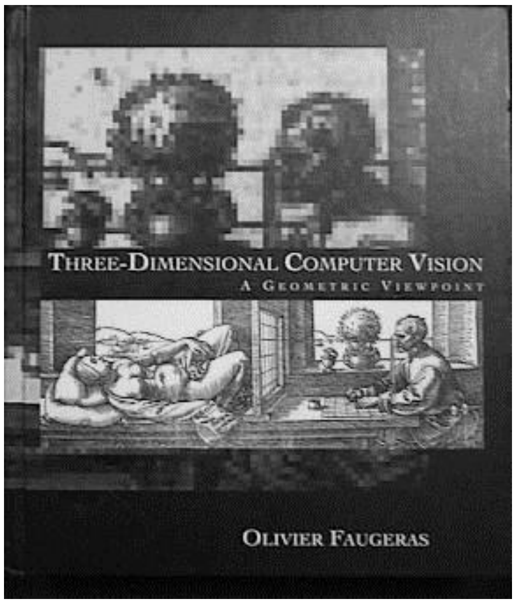
- We know how to detect points
- Next question: How to match them?



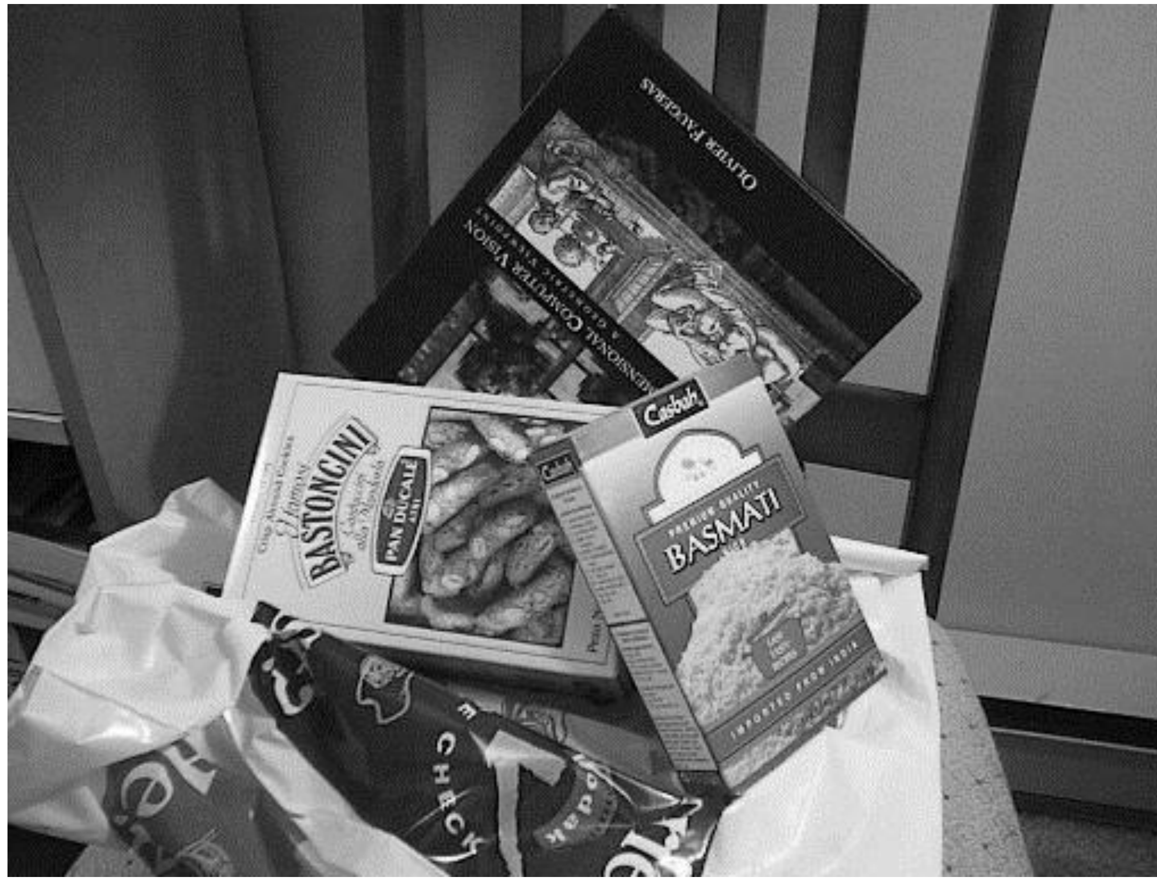
Point descriptor should be:
1. Invariant
2. Distinctive

Another version of the problem…

Want to find

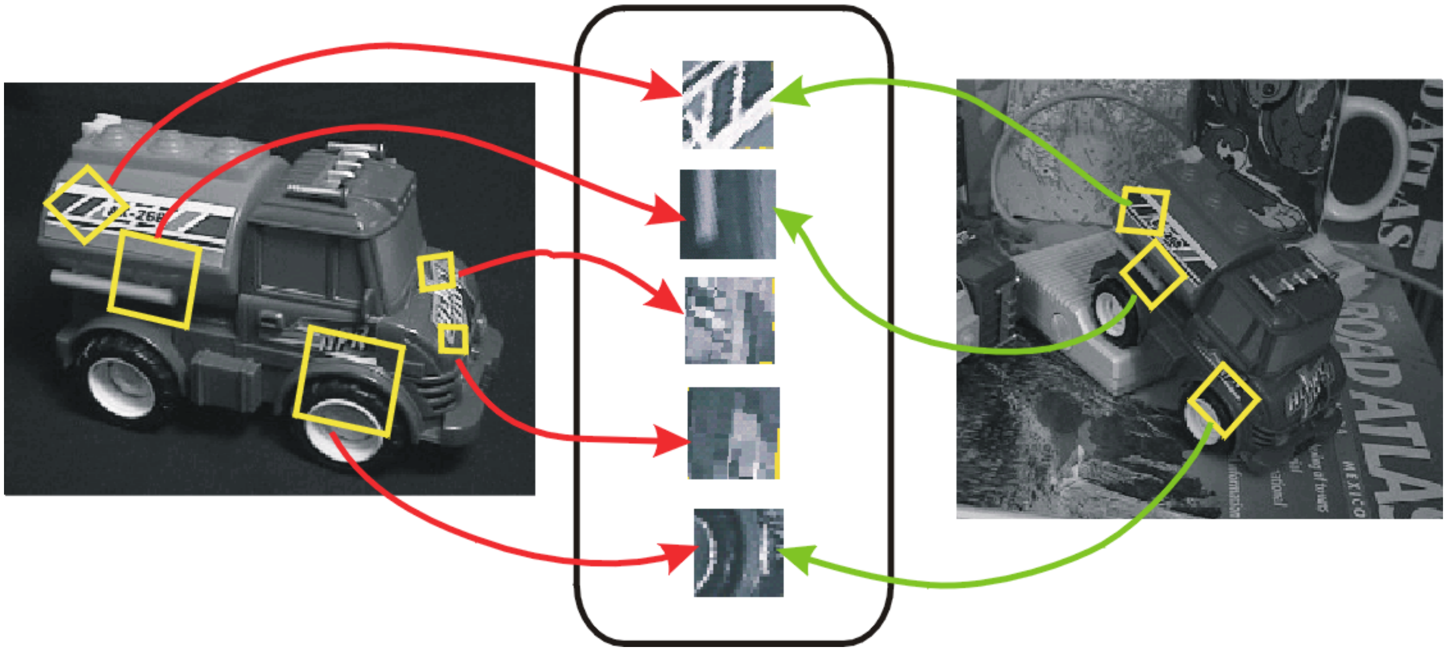


… in here



Idea of SIFT

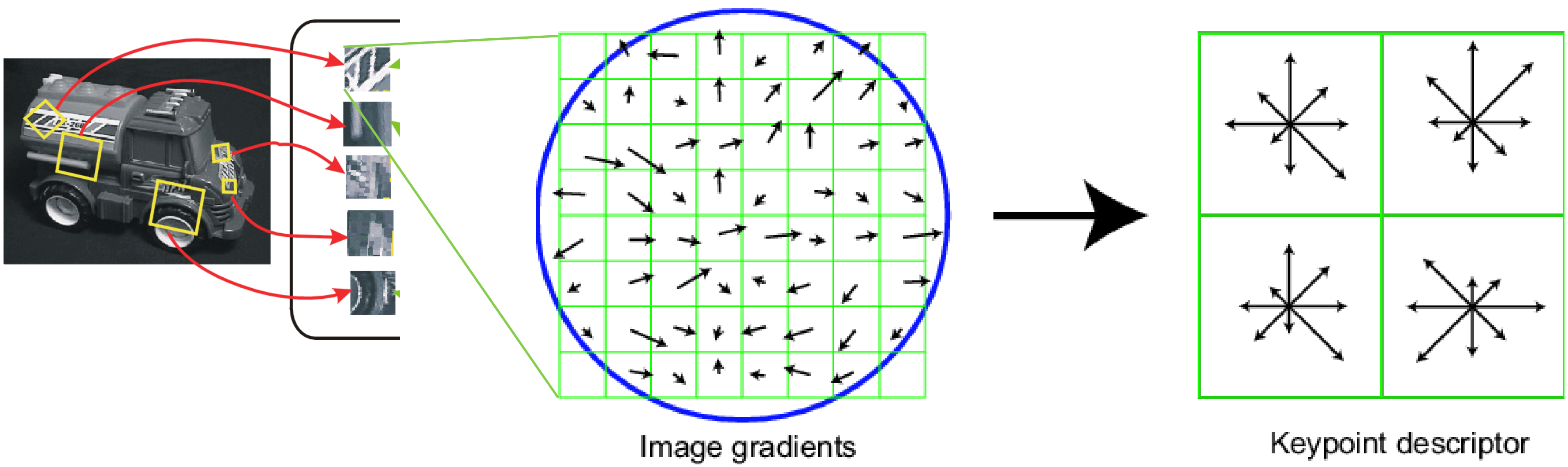
- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

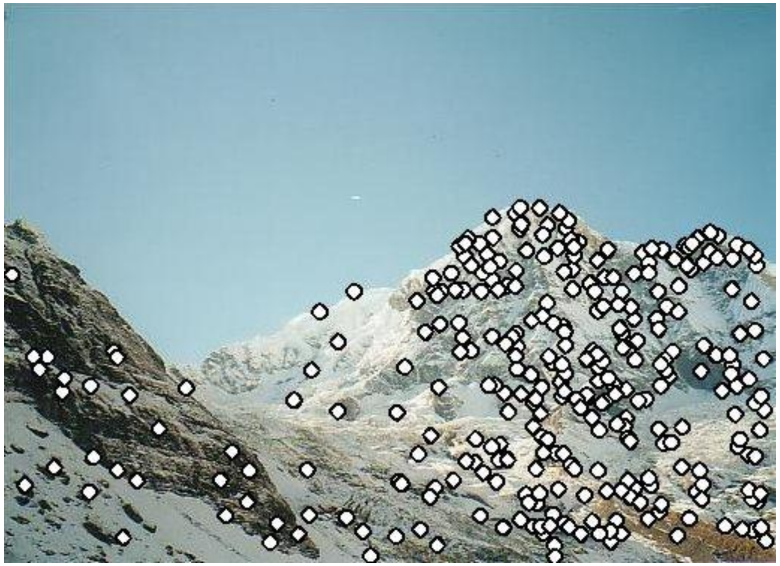
SIFT vector formation

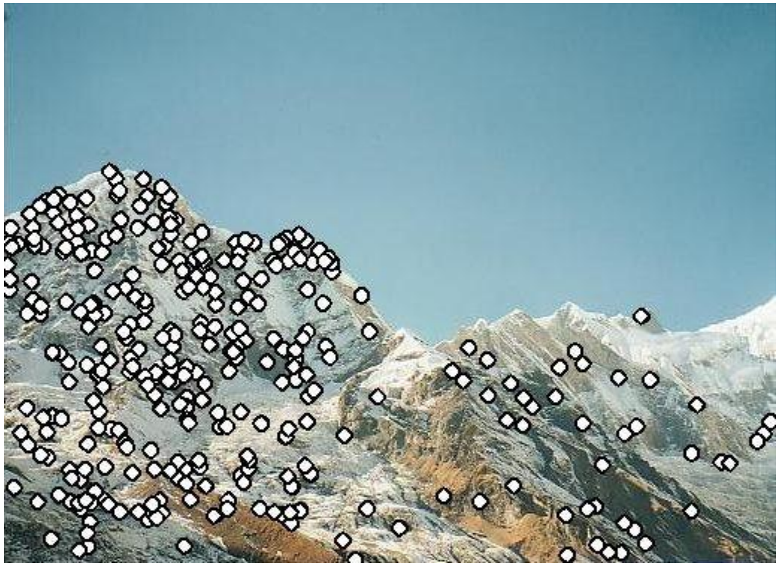
- 4x4 array of gradient orientation histograms over 4x4 pixels
  - not really histogram, weighted by magnitude
- 8 orientations x 4x4 array = 128 dimensions
- Motivation: some sensitivity to spatial layout, but not too much.



showing only 2x2 here but is 4x4

Feature-based alignment outline



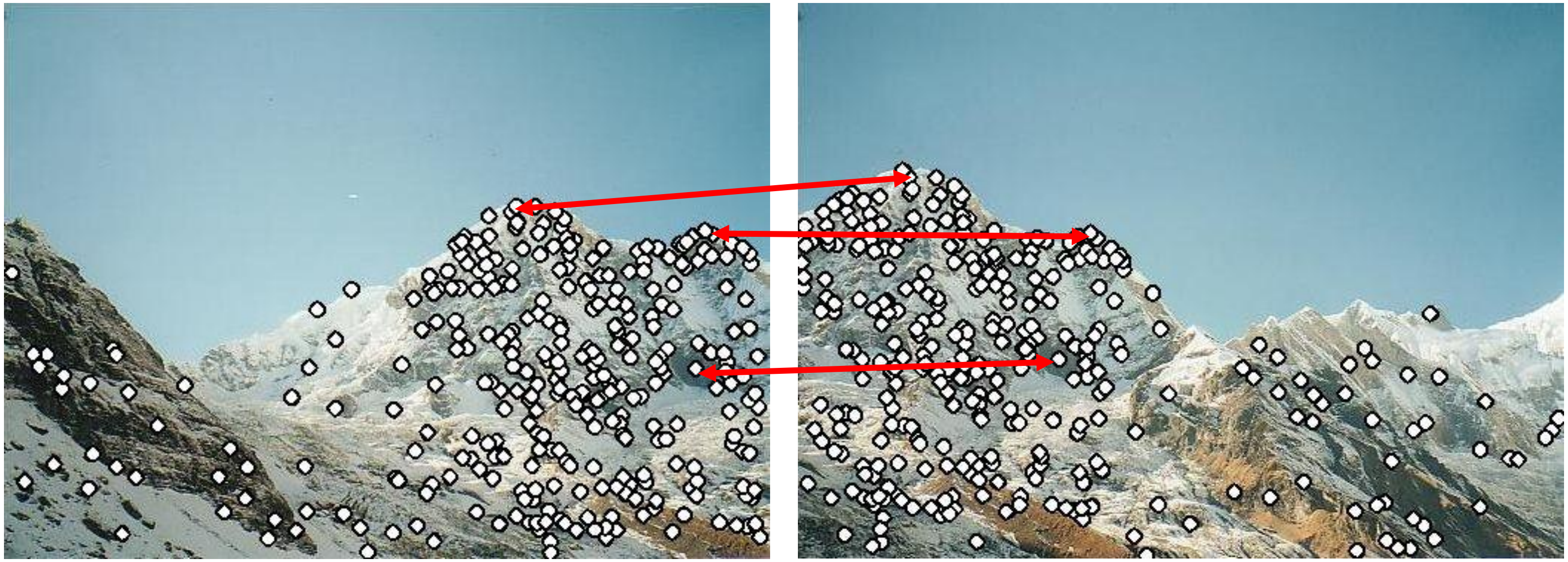


- Extract features

Feature-based alignment outline

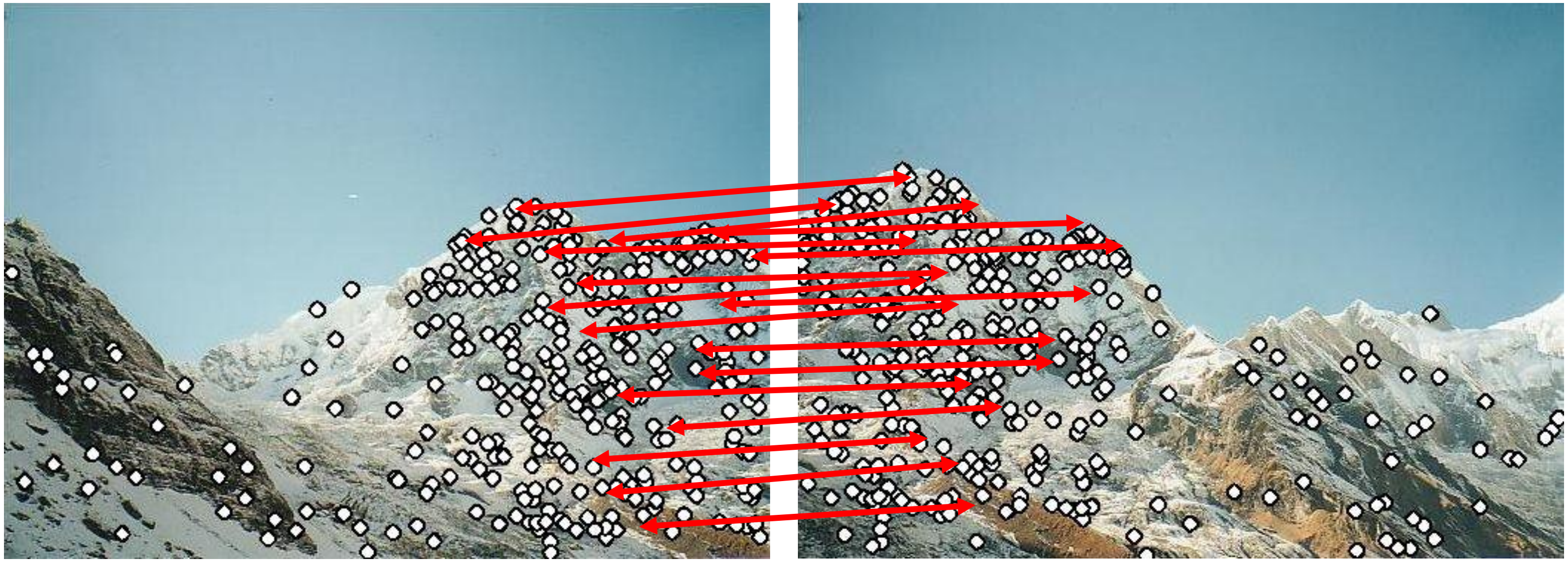
- Extract features
- Compute ***putative*** *matches*

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*)

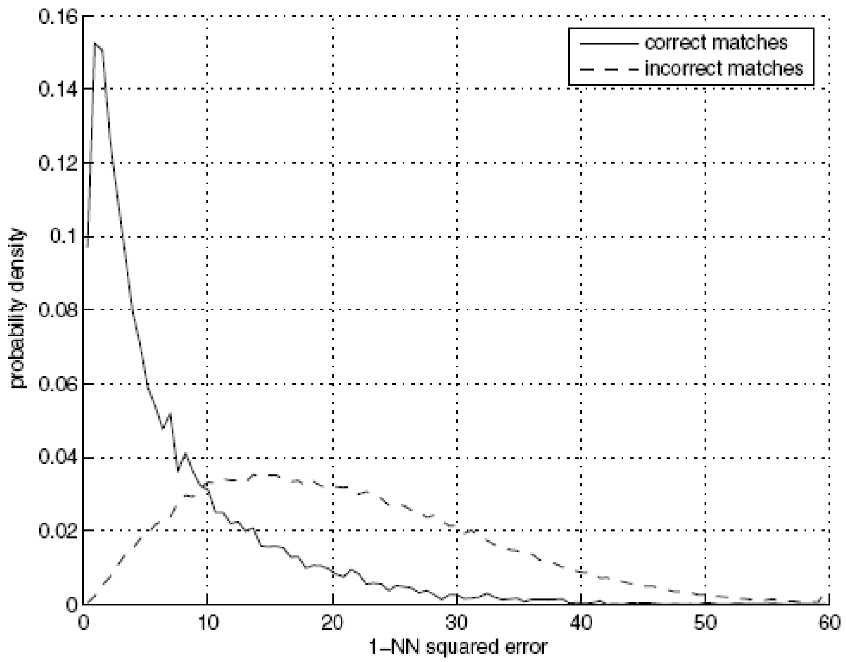
How to get “putative” matches?

Feature matching

- Exhaustive search
  - for each feature in one image, look at *all* the other features in the other image(s) – pick best one
- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
- Nearest neighbor techniques
  - *k*-trees and their variants

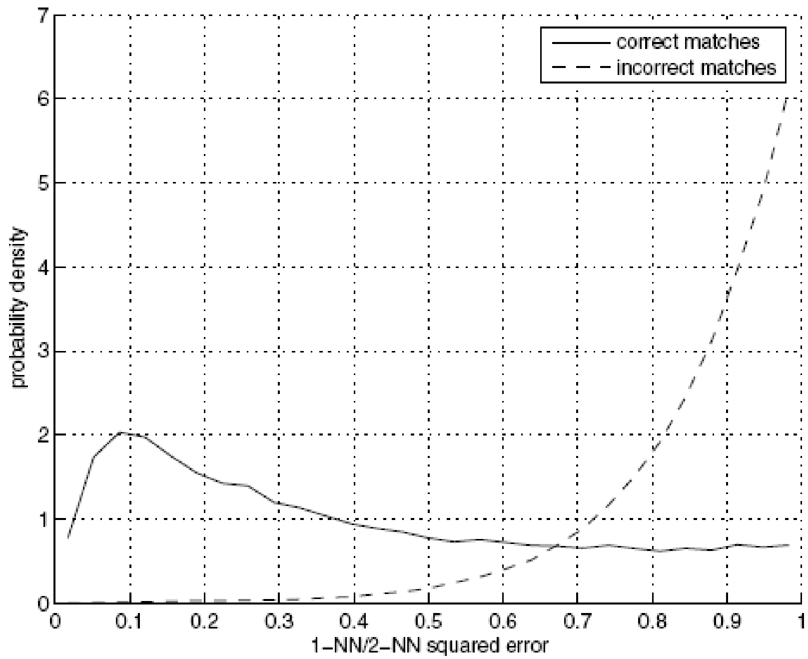
Feature-space outlier rejection

- Let's not match all features, but only these that have "similar enough" matches?
- How can we do it?
  - SSD(patch1,patch2) < threshold
  - How to set threshold?



Feature-space outlier rejection

- A better way [Lowe, 1999]:
  - 1-NN: SSD of the closest match
  - 2-NN: SSD of the <u>second-closest</u> match
  - Look at how much better 1-NN is than 2-NN, e.g. 1-NN/2-NN
  - That is, is our best match so much better than the rest?



Feature matching

- Exhaustive search
  - for each feature in one image, look at *all* the other features in the other image(s) – pick best one
- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
- Nearest neighbor techniques
  - *k*-trees and their variants

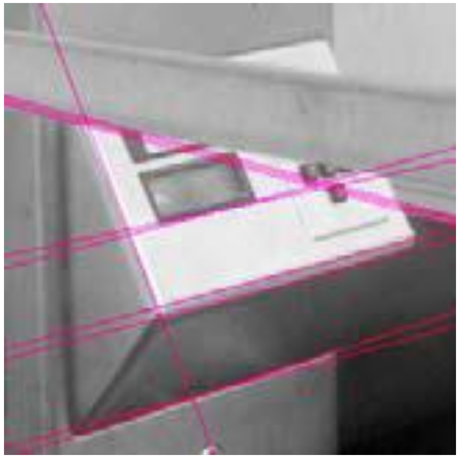
- *Problem: Even when pick best match, still lots (and lots) of wrong matches – "outliers"*

Another way to remove mistakes

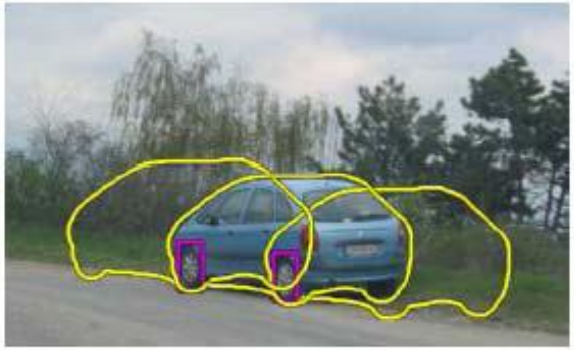
- Why are we doing matching?
  - To compute a model of the relation between entities
- So this is really "model fitting"

Fitting

- Choose a parametric model to represent a set of features – *remember this???*




simple model: lines simple model: circles




complicated model: car

Source: K. Grauman

Fitting: Issues

Case study: Line detection

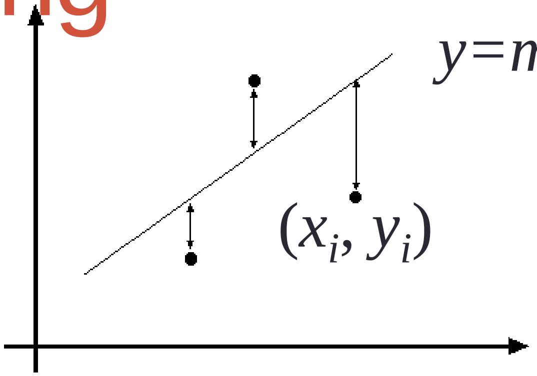
- **Noise** in the measured feature locations
- **Extraneous data:** clutter (outliers), multiple lines
- **Missing data:** occlusions

Slide: S. Lazebnik

Least squares line fitting

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

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



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

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

$$\frac{dE}{dB}=2X^TXB-2X^TY=0$$

$$\boxed{X^TXB=X^TY}$$

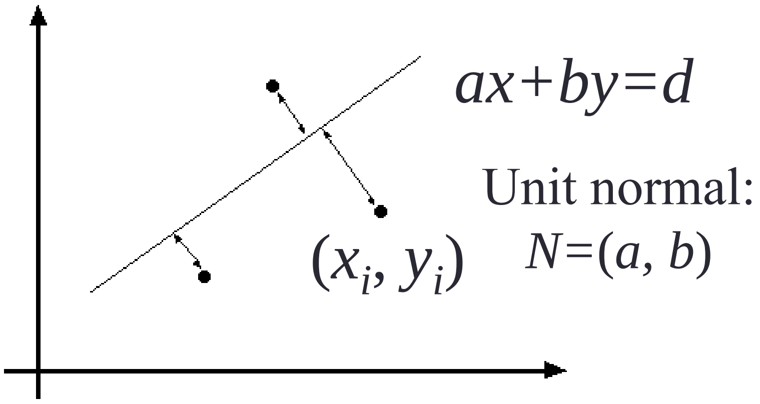
Normal equations: least squares solution to $XB=Y$

Problem with “vertical” least squares

- Not rotation-invariant
- Fails completely for vertical lines

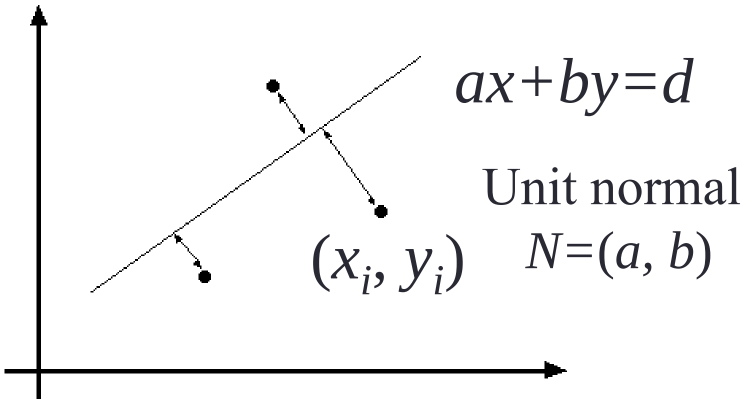
Total least squares

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



Total least squares

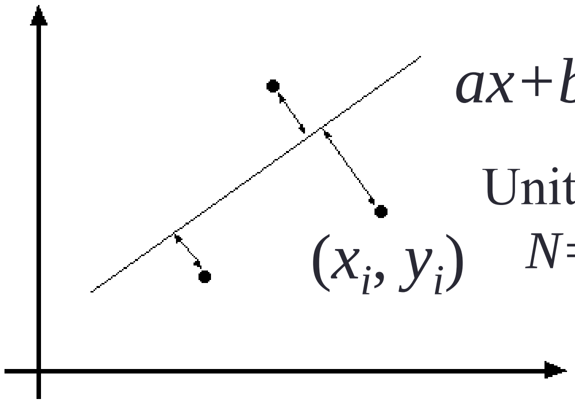
- Distance between point (x_i, y_i) and line ax+by=d
- Find (a, b, d) to minimize the sum of squared perpendicular distances



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

Total least squares

- Distance between point (x_i, y_i) and line $ax+by=d$
- Find (a, b, d) to minimize the sum of squared perpendicular distances



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

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

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

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

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

Total least squares

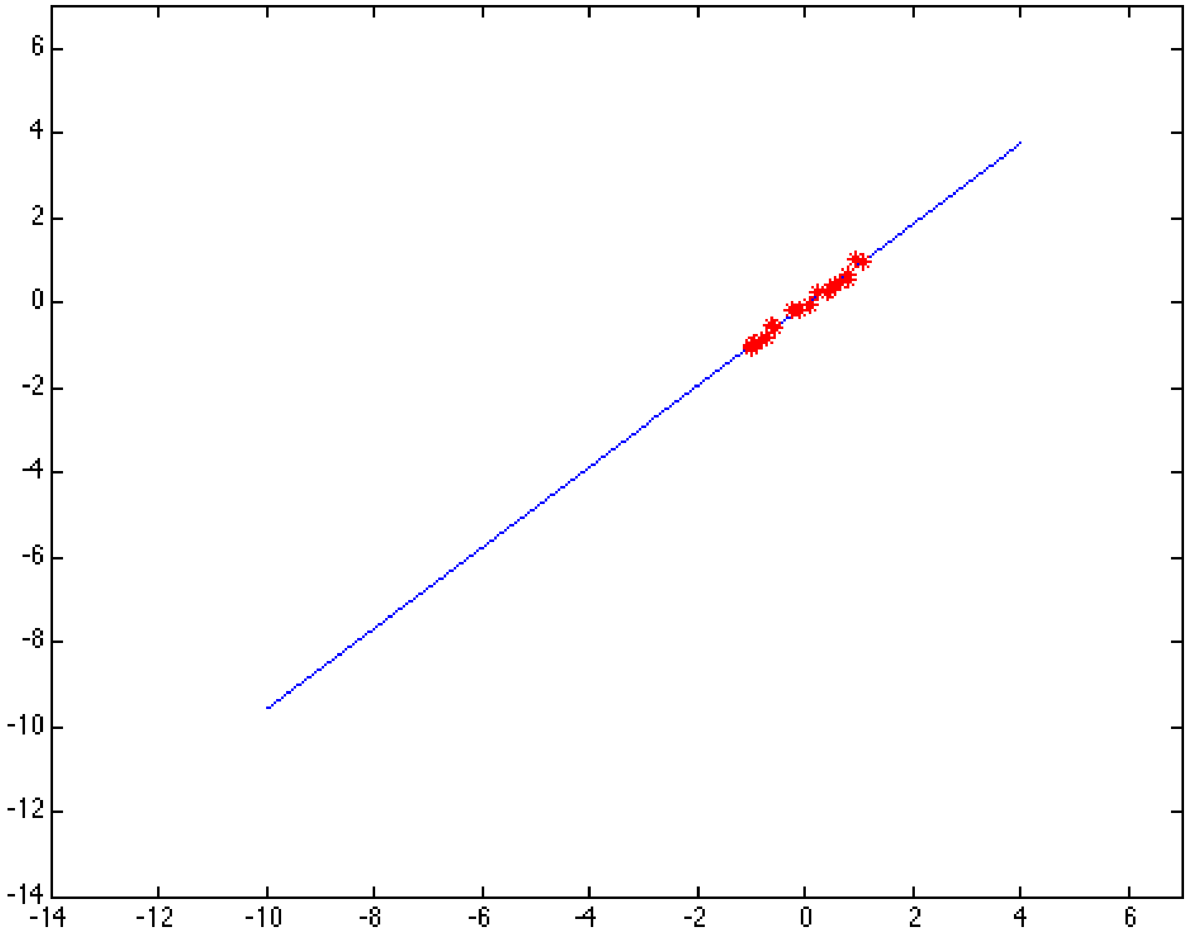
$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}$$

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

second moment matrix

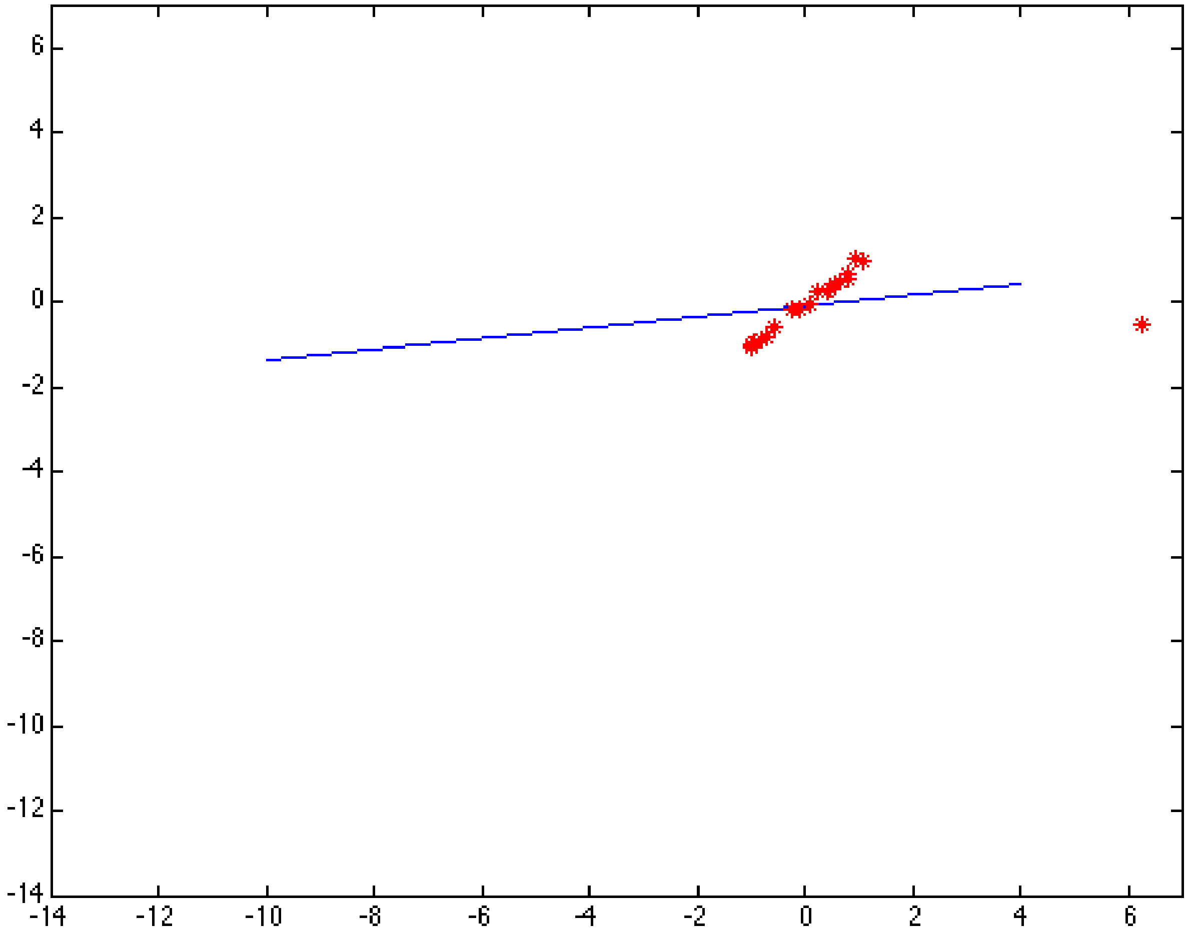
Least squares: Robustness to noise

- Least squares fit to the red points:



Least squares: Robustness to noise

- Least squares fit with an outlier:

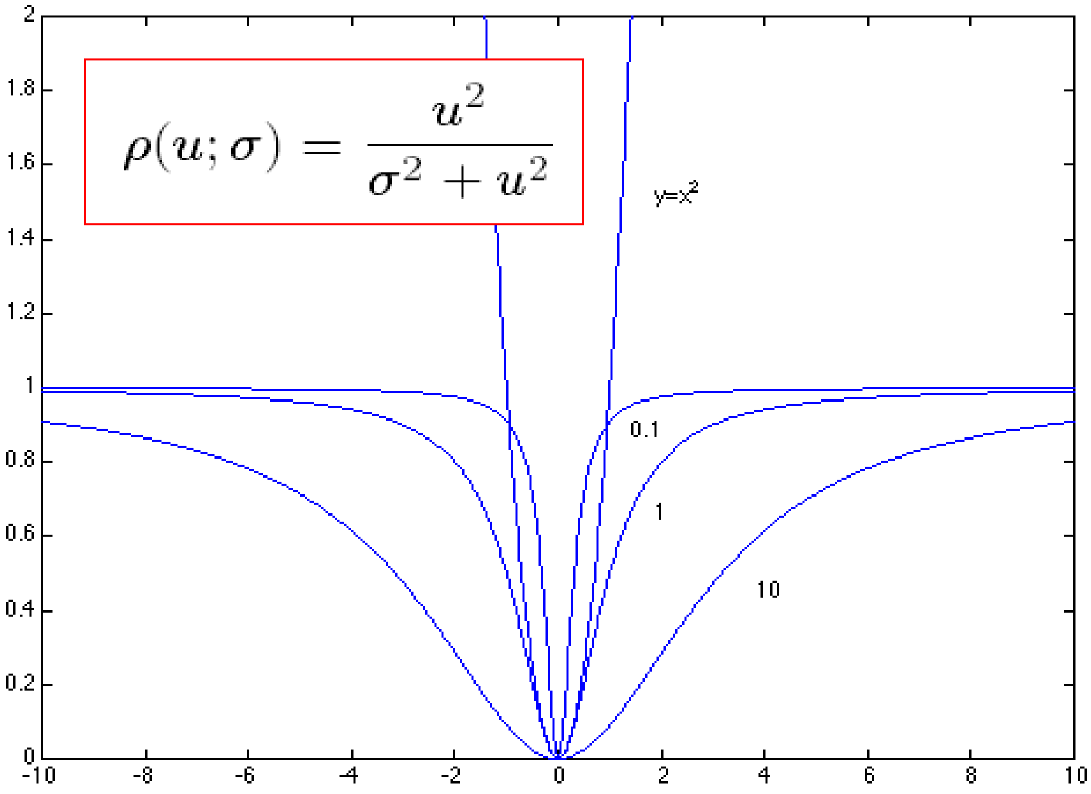


Problem: squared error heavily penalizes outliers

Robust estimators

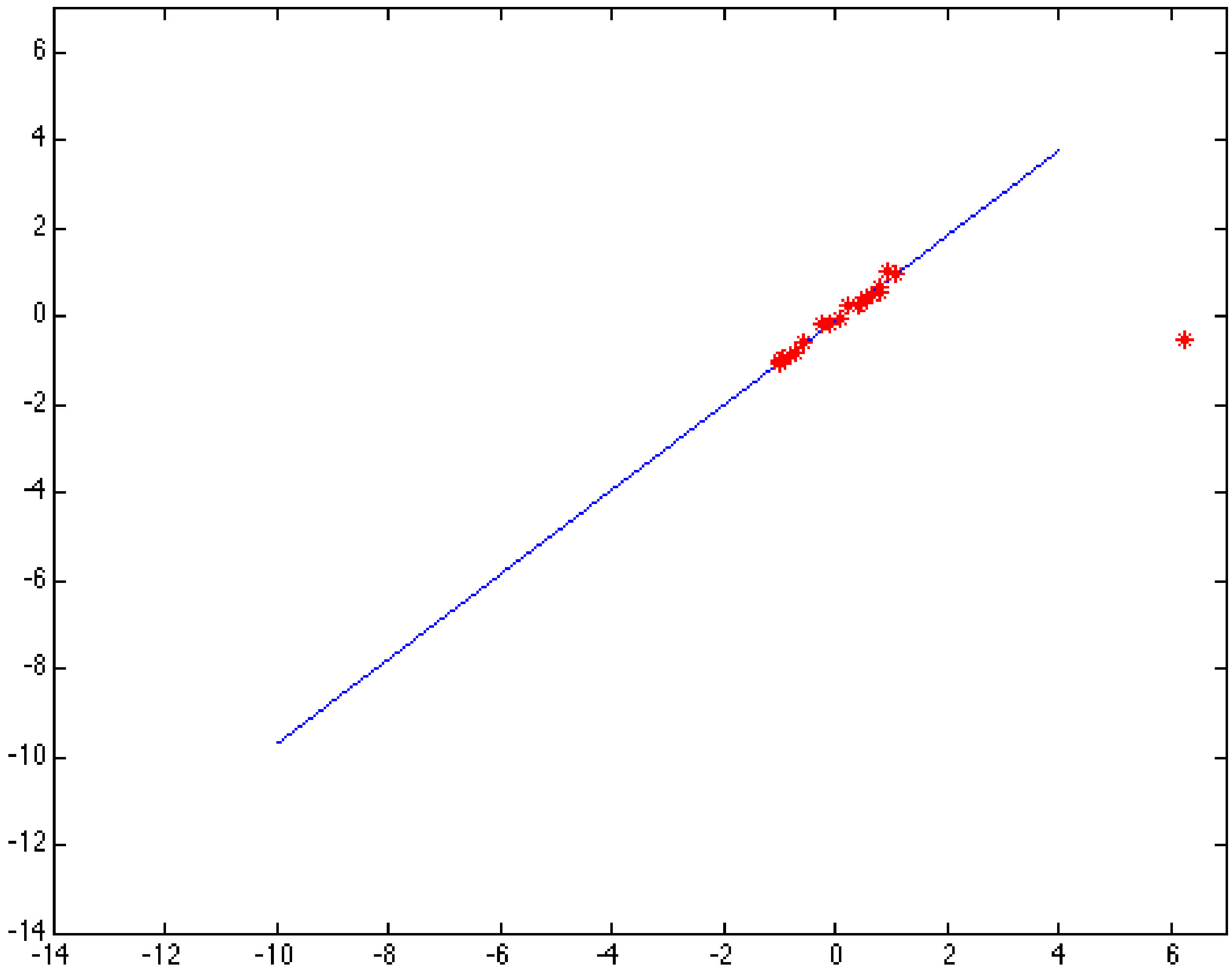
- General approach: minimize $\sum_i \rho\left(r_i\left(x_i, \theta\right); \sigma\right)$

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



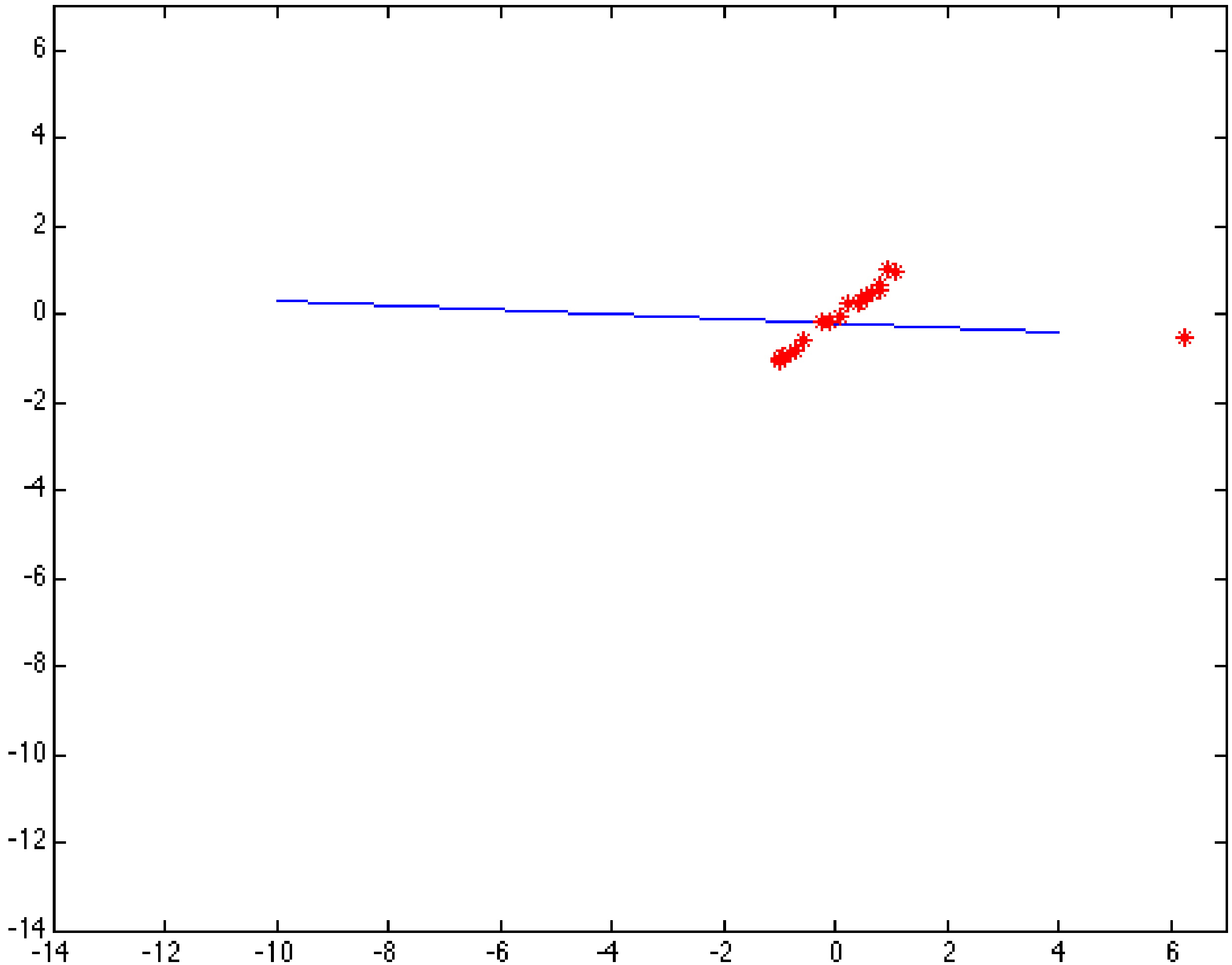
The robust function ρ behaves like squared distance for small values of the residual u but saturates for larger values of u

Choosing the scale: Just right



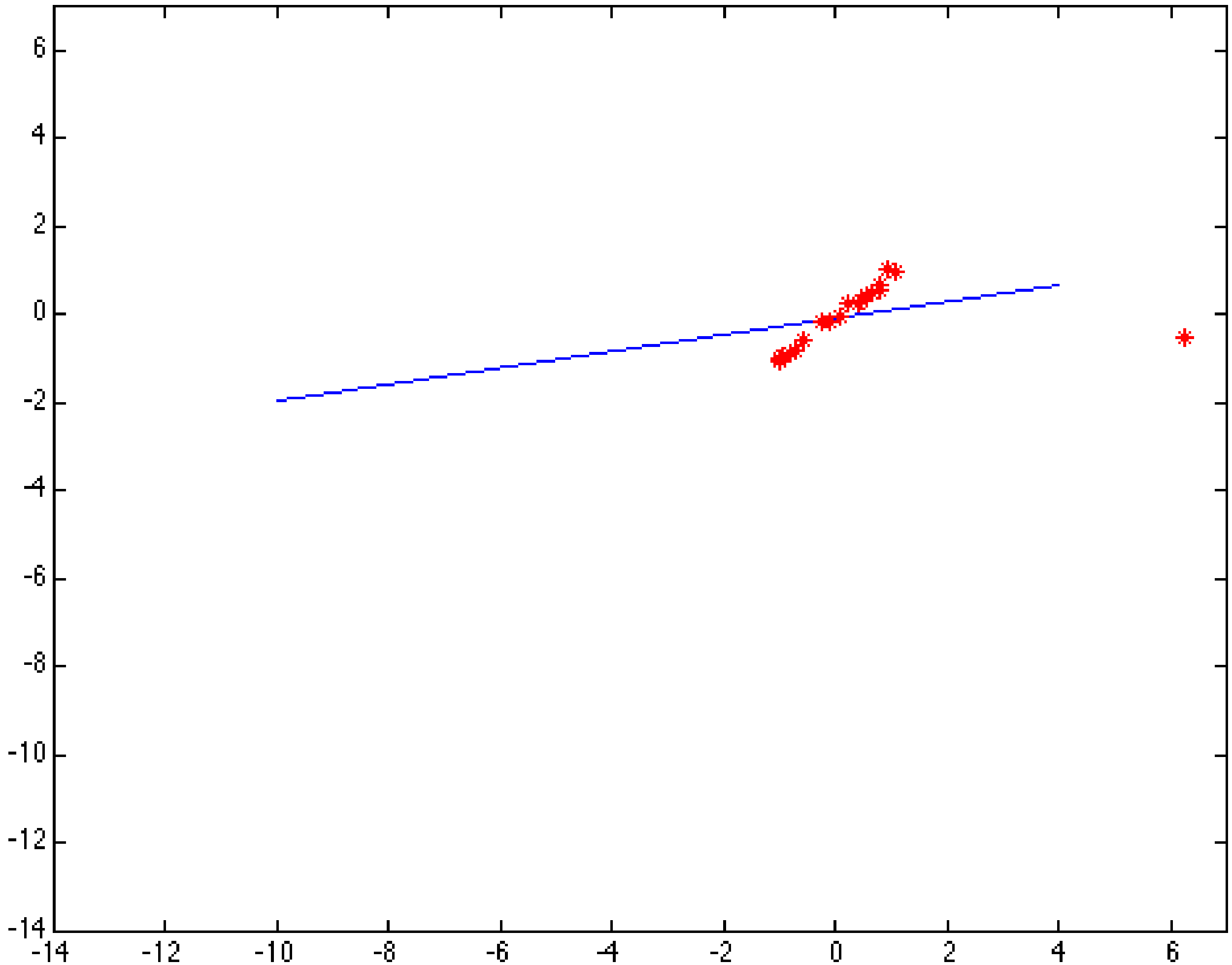
The effect of the outlier is minimized

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



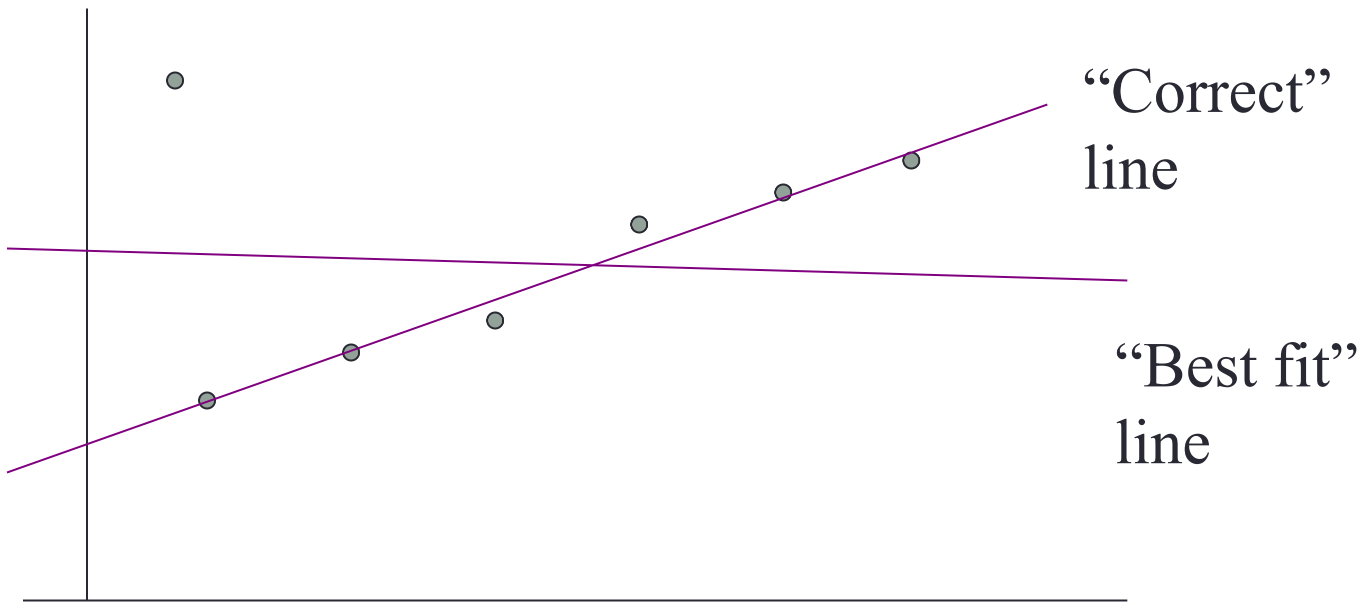
Behaves much the same as least squares

“Find consistent matches”???

- Some points (many points) are static in the world
- Some are not
- Need to find the right ones so can compute pose.
- Well tried approach:
  - Random Sample Consensus (RANSAC)

Simpler Example

- Fitting a straight line



Discard Outliers

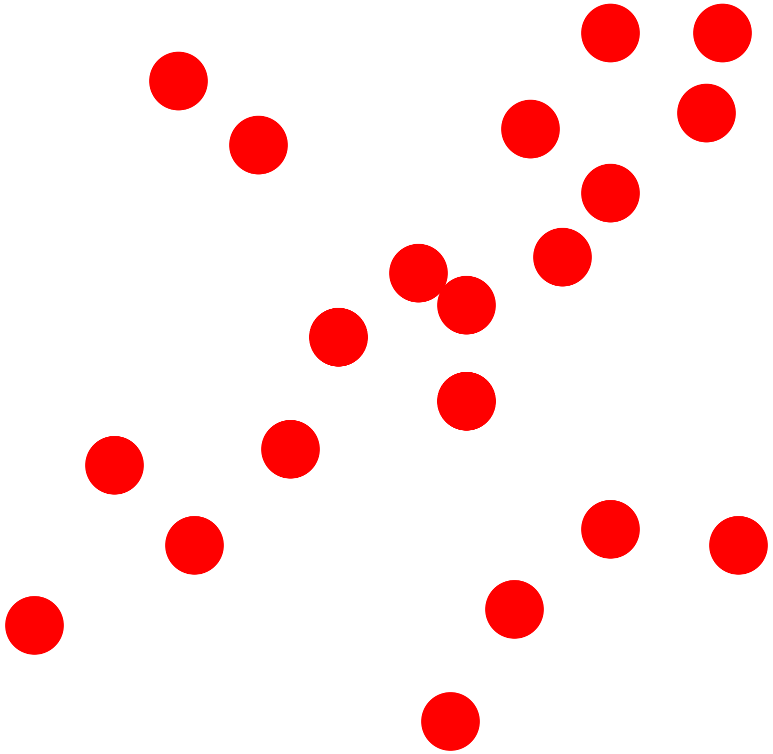
- No point with d>t
- RANSAC:
  - RANdom SAmple Consensus
  - Fischler & Bolles 1981
  - Copes with a large proportion of outliers

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

(RANdom SAmple Consensus) :

Fischler & Bolles in '81.



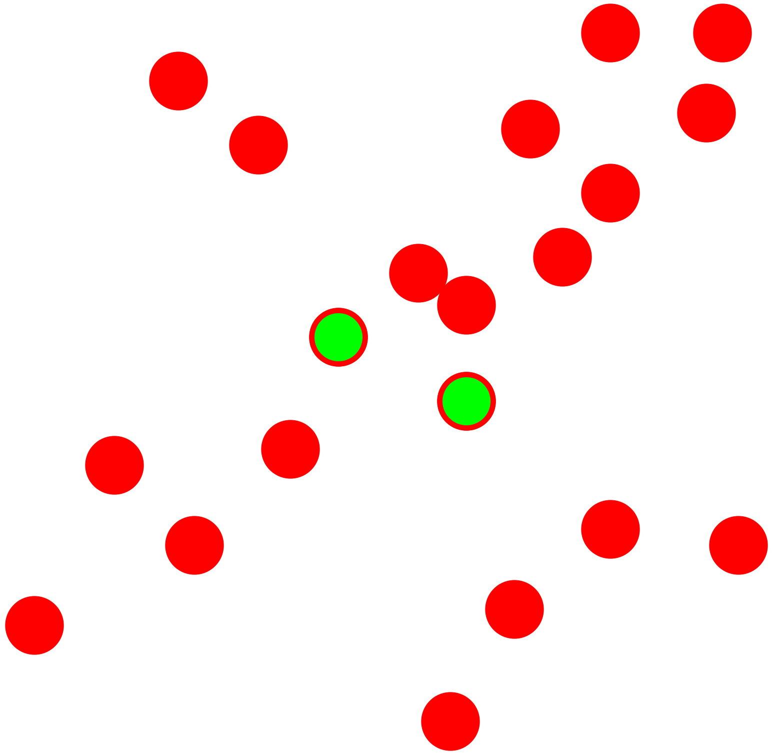
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



Algorithm:

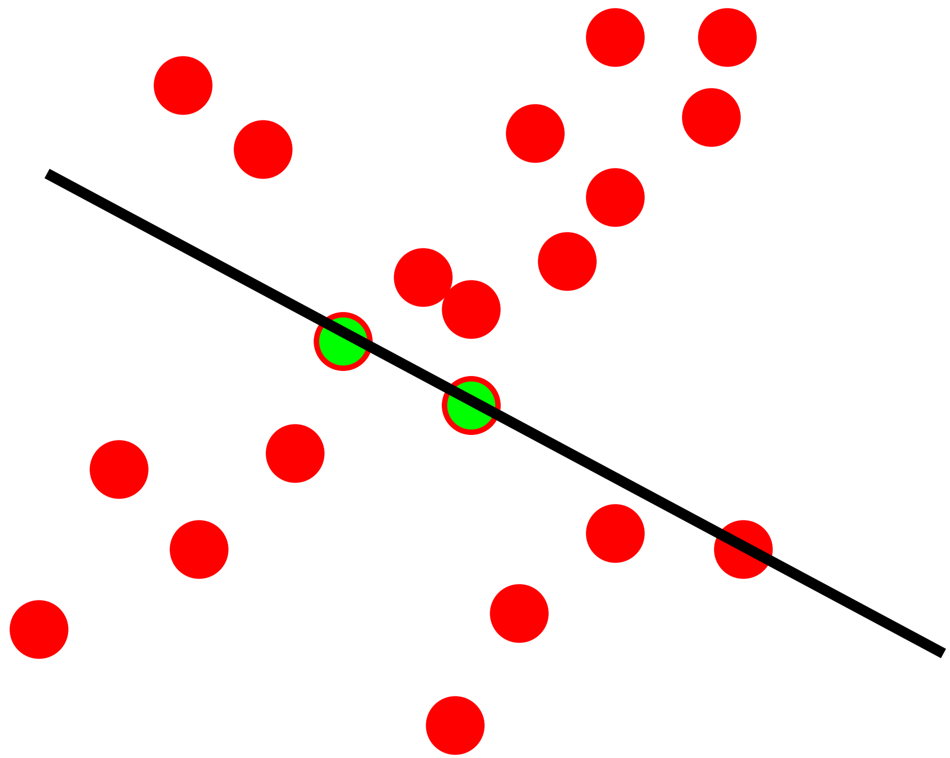
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Illustration by Savarese

RANSAC

Line fitting example



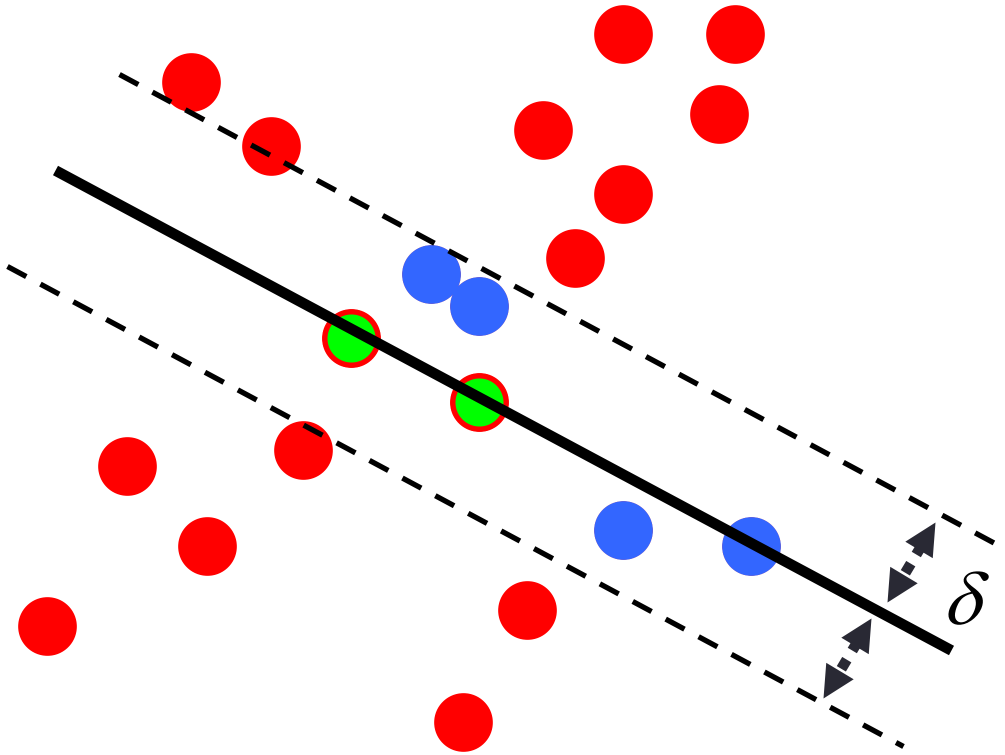
Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC

Line fitting example



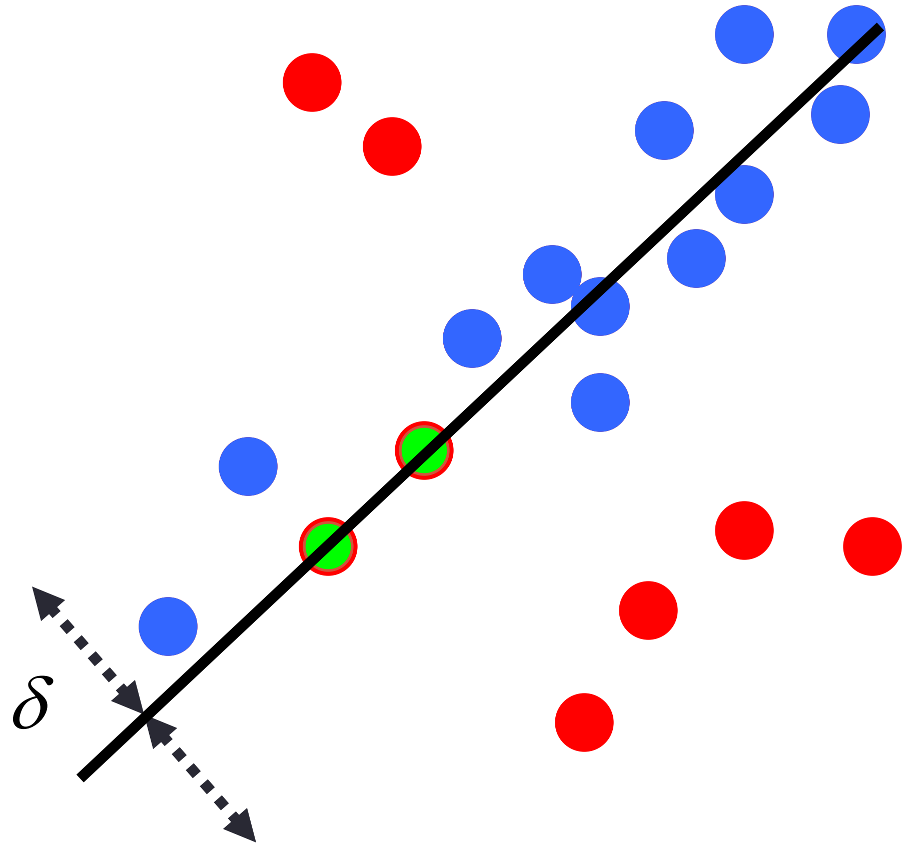
$$N_I = 6$$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

RANSAC



$N_I = 14$

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Best Line has most support

- More support -> better fit

In General

- Fit a more general model
- Sample = minimal subset - s
  - Translation: pick one point pair
  - Homography (for plane) – pick 4 point pairs
  - Fundamental matrix – pick 7 point pairs

- Algorithm
  - Randomly select s points
  - Instantiate a model
  - Get consensus set S_i
  - If $| S_i | > T$, terminate and return model
  - Repeat for N trials, return model with max $| S_i |$

Distance Threshold

- Requires noise distribution
- Location: Gaussian noise with σ
- Distance: Chi-squared distribution with DOF m
  - 95% cumulative:
  - Line, F: m=1, t=3.84 σ^2
- I.e. -> 95% prob that d<t when point is inlier

How many samples ?

- We want: at least one sample with all inliers
- Can't guarantee: probability p
- E.g. p =0.99

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

Calculate N

- s – number of points to compute solution
- p – probability of success
- *e* – proportion outliers, so % inliers = (1-*e*)
- P(sample set with all inliers)=$(1-e)^s$
- P(sample set will have at least one outlier)=
  $(1-(1-e)^s)$
- P(all N samples have outlier)=$(1-(1-e)^s)^N$
- We want P(N samples an outlier)<1-p
- $(1-(1-e)^s)^N < 1-p$

$$\boxed{N > \log(1-p) / \log(1-(1-e)^s)}$$

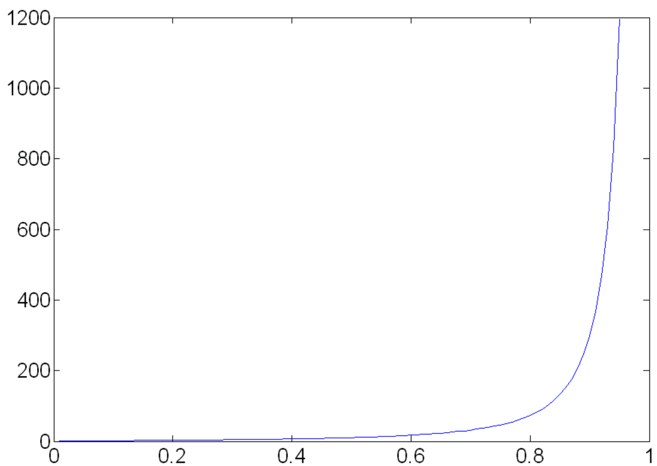
Samples required for inliers only in a sample

- P=0.99
  - s=2, ε=5% => N=2
  - s=2, ε=50% => N=17
  - s=4, ε=5% => N=3
  - s=4, ε=50% => N=72
  - s=8, ε=5% => N=5
  - s=8, ε=50% => N=1177

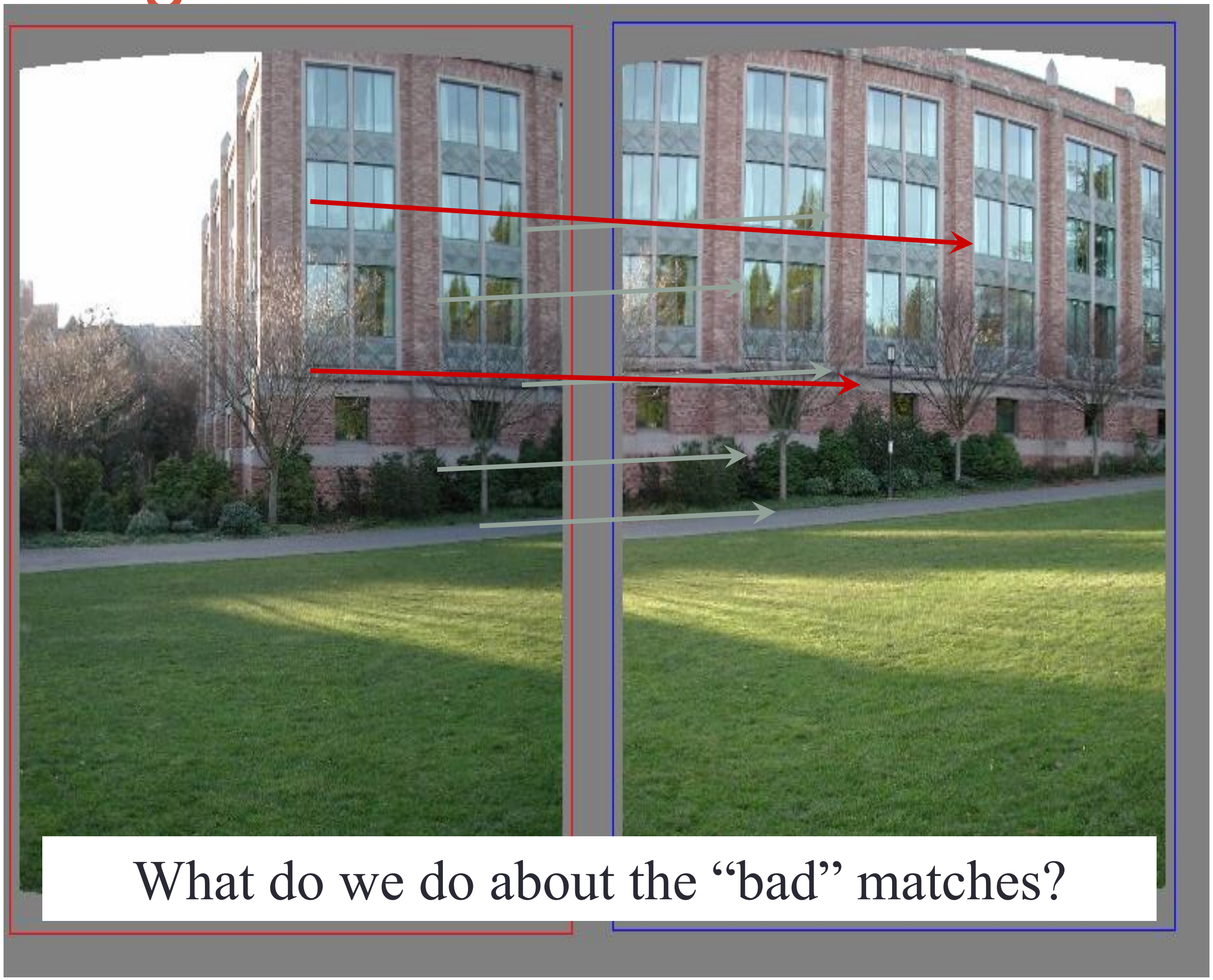
	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

- N = f(ε), *not the number of points*
- N increases steeply with s

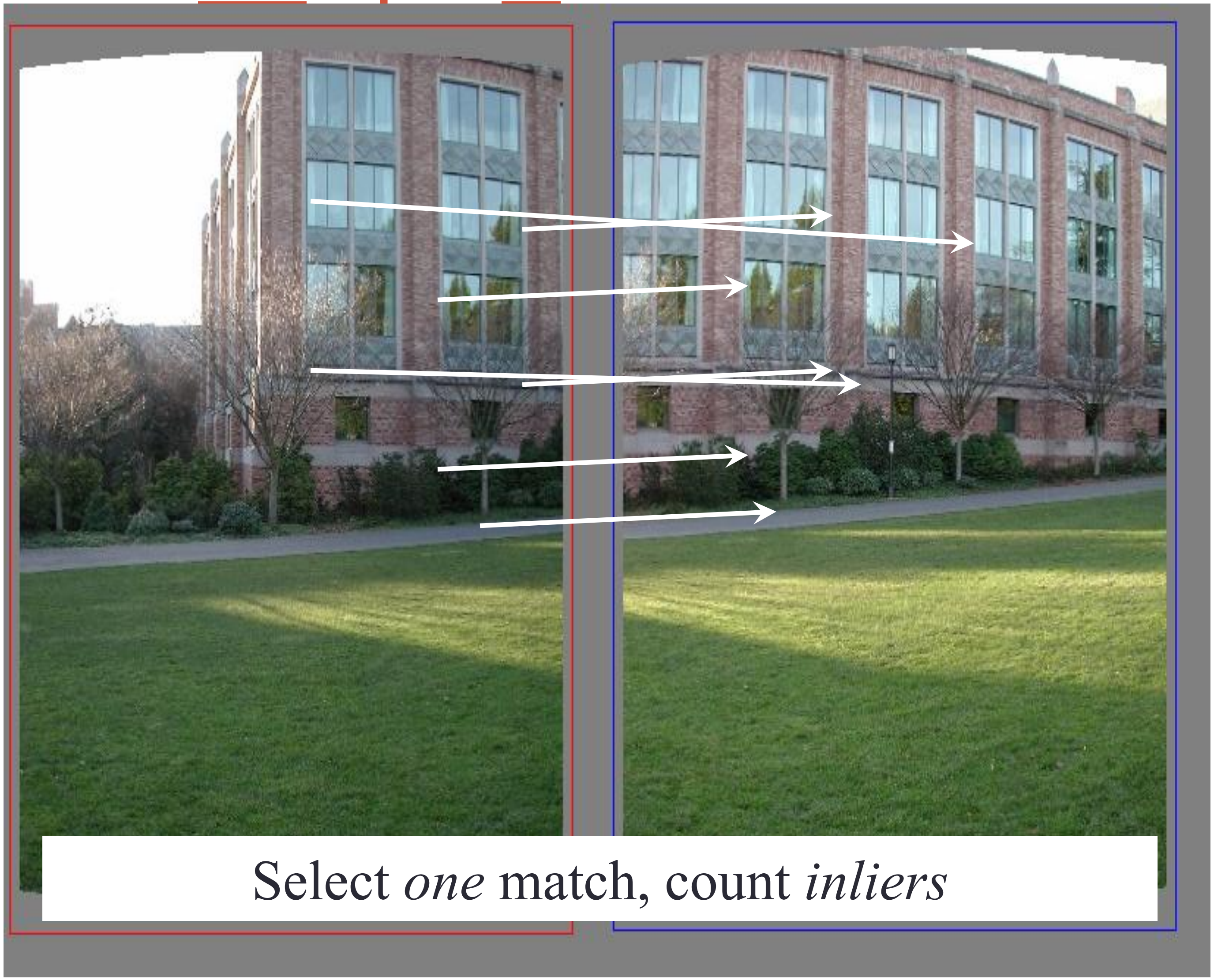
$$\boxed{N > \log(1-p)/\log(1-(1-e)^s)}$$



Matching features

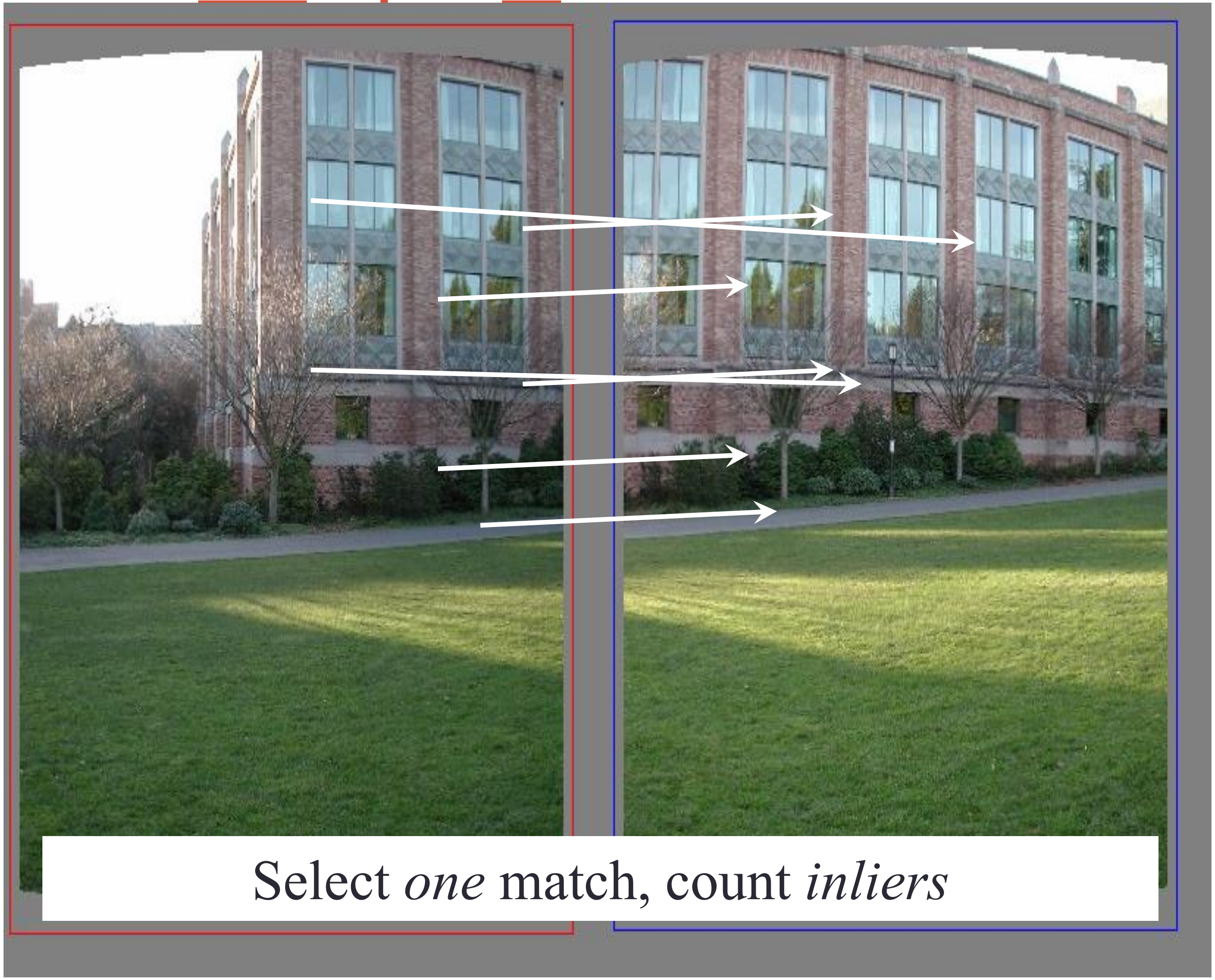


RAndom SAmple Consensus



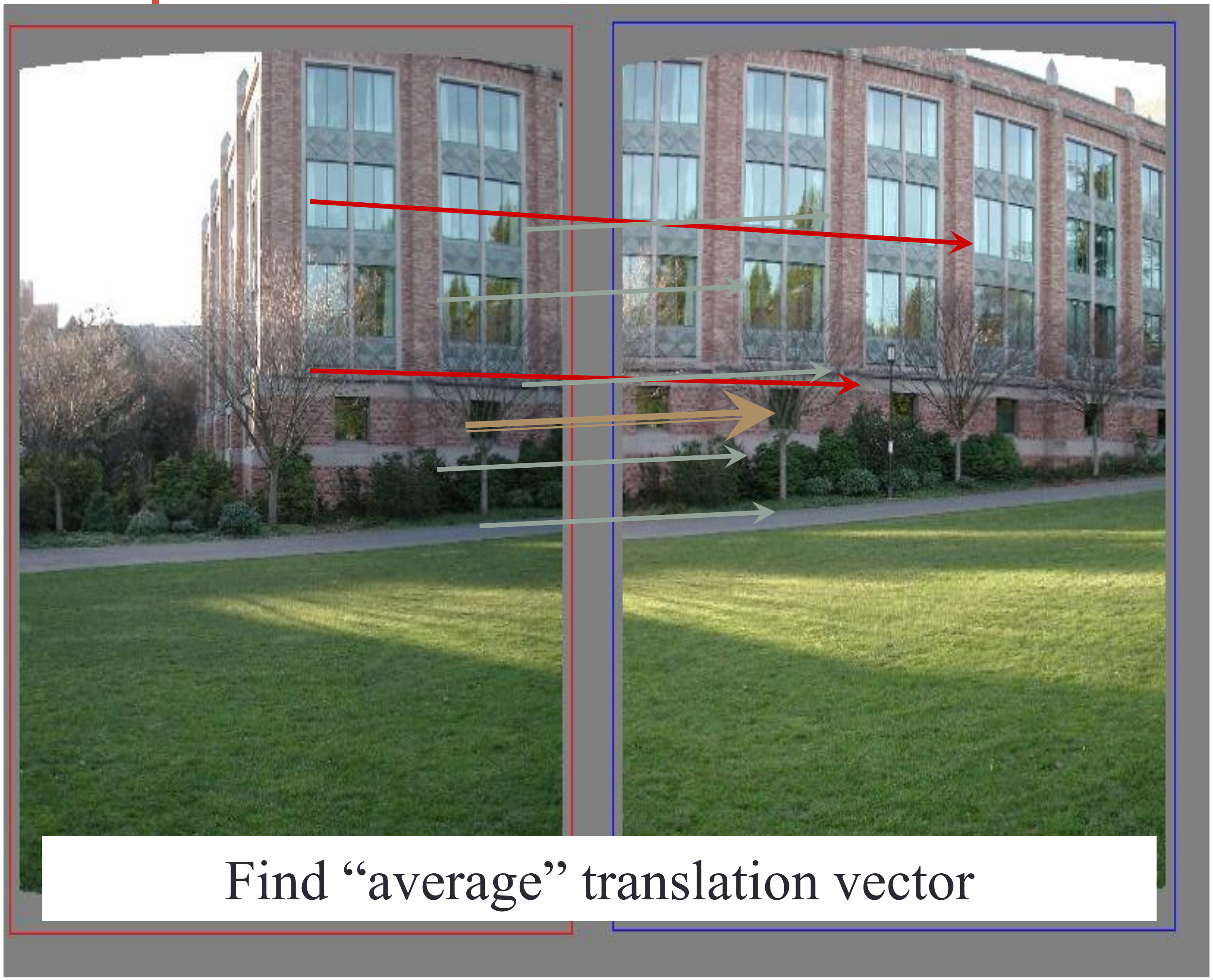
Select *one* match, count *inliers*

RAndom SAmple Consensus



Select *one* match, count *inliers*

Least squares fit



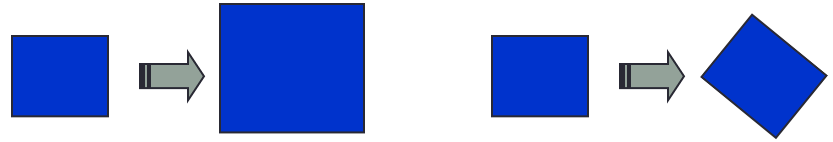
RANSAC for estimating homography

- RANSAC loop:

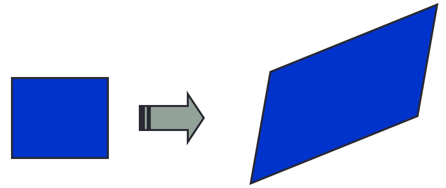
1. Select four feature pairs (at random)
2. Compute homography H (exact)
3. Compute *inliers* where $SSD(p_i', \mathbf{H}\, p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares H estimate on all of the inliers

2D transformation models

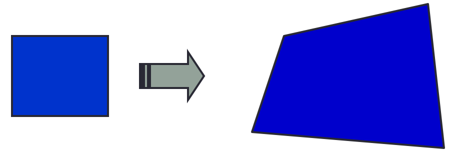
- Similarity (translation, scale, rotation)



- Affine



- Projective (homography)



Source: S. Lazebnik

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=∞, sample_count =0
  - While N >sample_count
    - Choose a sample 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

Source: M. Pollefeys

RANSAC conclusions

Good

- Simple and general
- Applicable to many different problems, often works well in practice
- Robust to outliers
- Applicable for larger number of parameters than Hough transform
- Parameters are easier to choose than Hough transform

Bad

- Computational time grows quickly with fraction of outliers and number of parameters
- Not good for getting multiple fits

Common applications

- Computing a homography (e.g., image stitching)
- Estimating fundamental matrix (relating two views)
- Every problem in robot vision