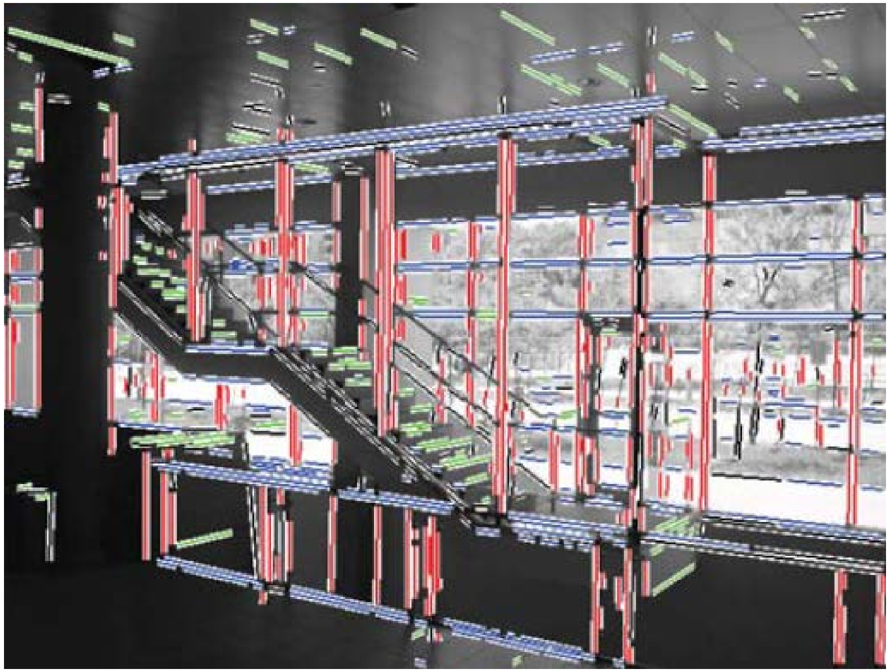
CS 4495 Computer Vision
RANdom **SA**mple **C**onsensus

Aaron Bobick

School of Interactive Computing



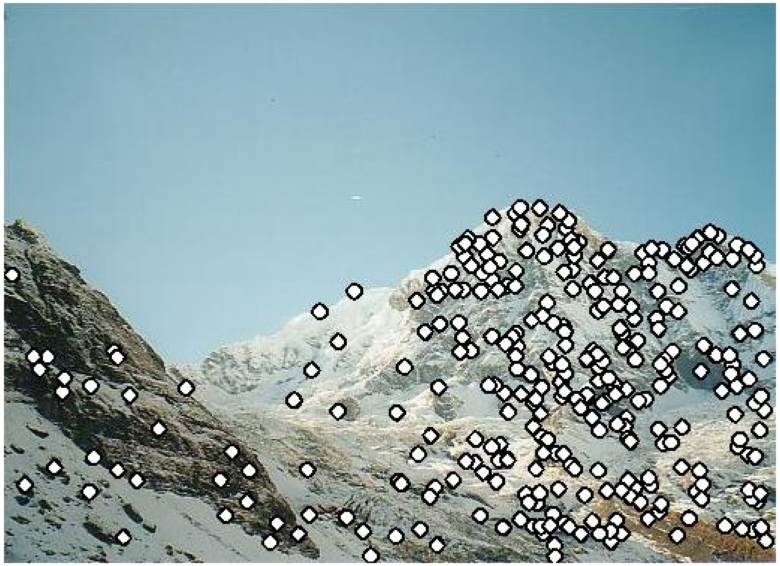
Administrivia

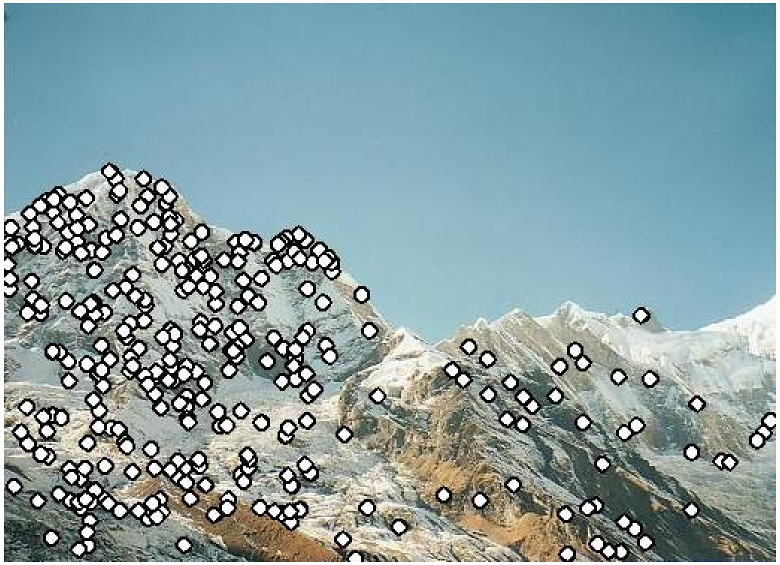
- PS 3:
  - Check Piazza - good conversations
  - The F matrix: the actual numbers may vary quite a bit. But check the epipolar lines you get.
    - Normalization: read extra credit part. At least try removing the centroid. Since we're using homogenous coordinates (2D homogenous have 3 elements) it's easy to have a transformation matrix that subtracts off an offset.

  - Go back an recheck slides: A 3 vector in these projective geometry is both a point and a line.

Matching with Features

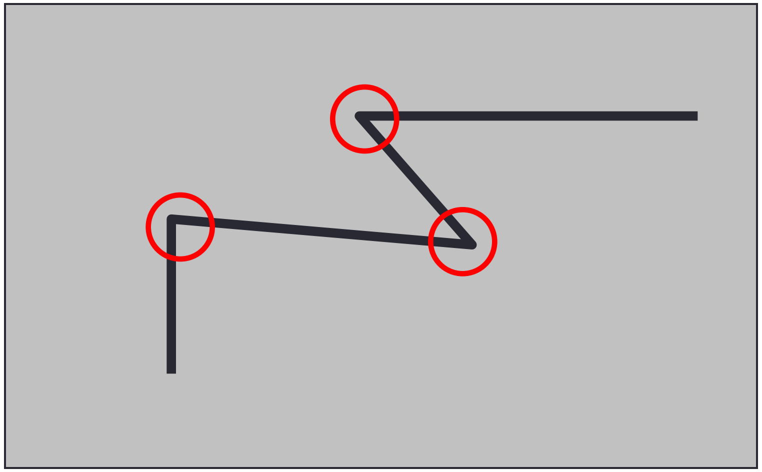
- Want to compute transformation from one image to the other
- Overall strategy:
  - Compute features
  - Match matching features (duh?)
  - Compute best transformation (translation, affine, homography) from matches





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:

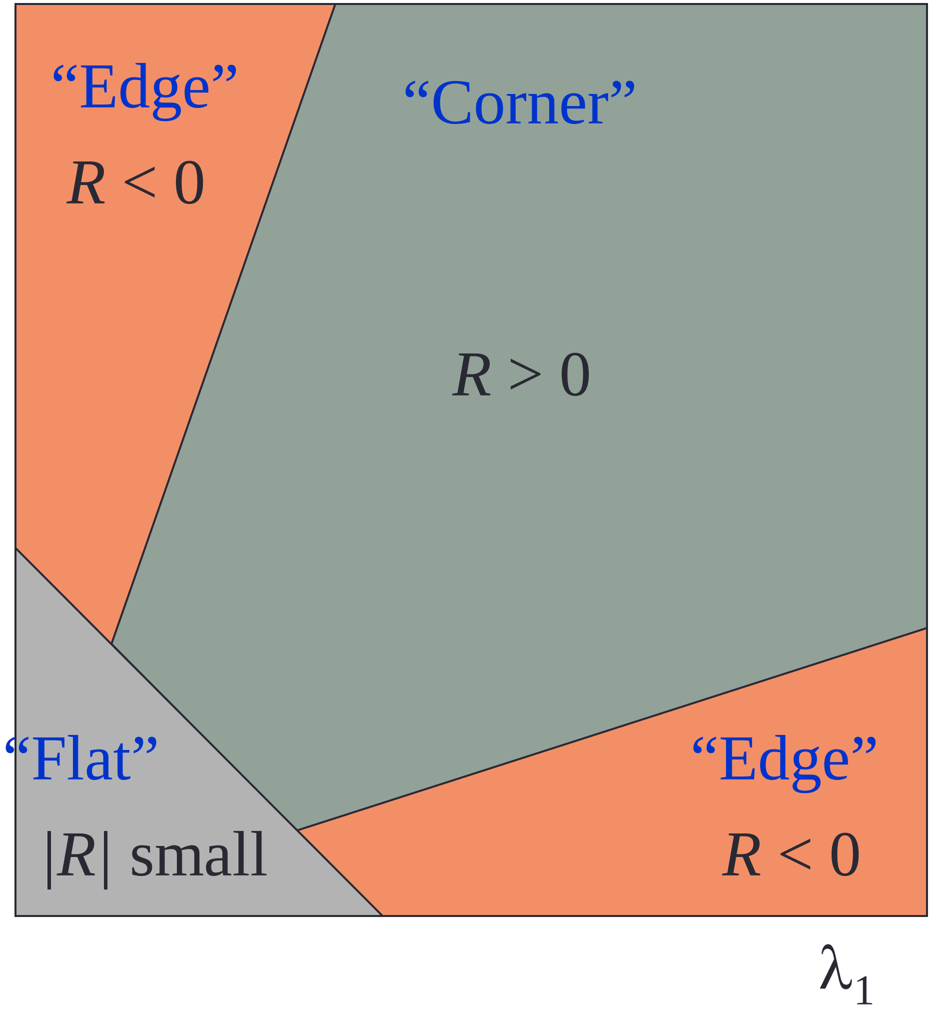
$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

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

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

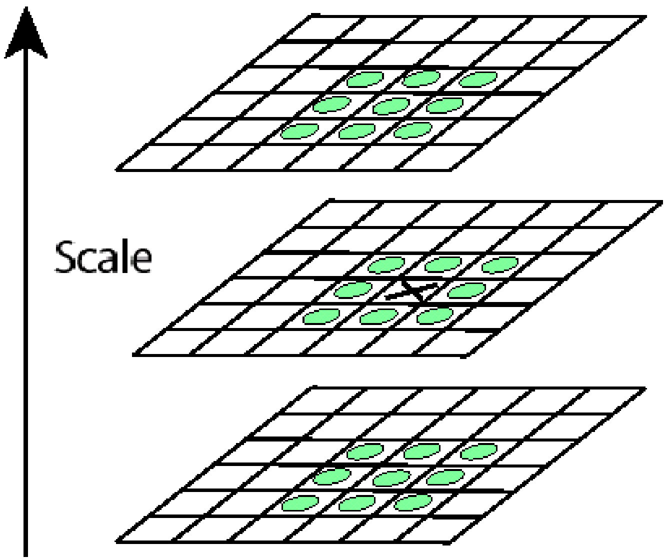
Harris Detector: Mathematics

- 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

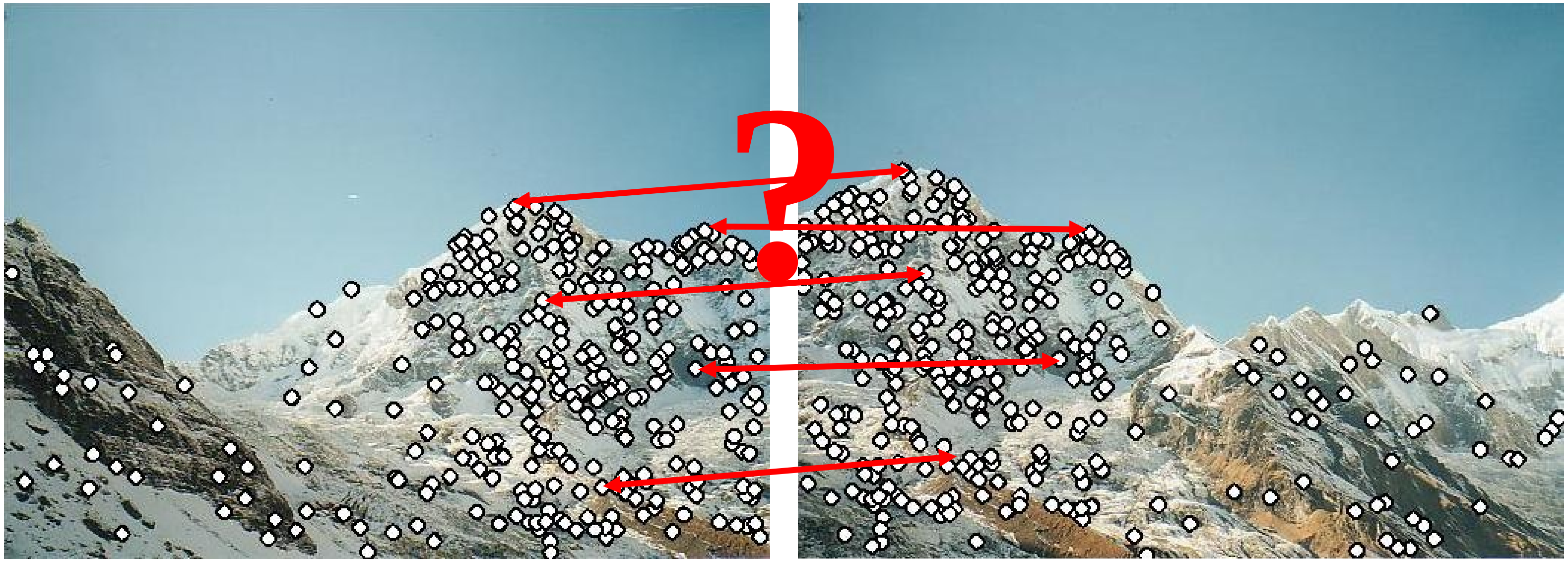
- General idea: find robust extremum (maximum or minimum) both in space and in scale.
- SIFT specific suggestion: use DoG pyramid to find maximum values (remember edge detection?) – then eliminate "edges" and pick only corners.
- More recent: use Harris detector to find maximums in space and then look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

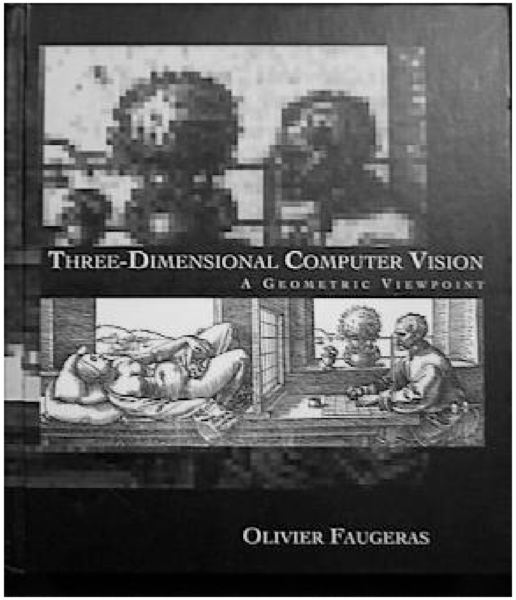
Point Descriptors

- We know how to detect points
- How to match them? Two parts:
  - Compute a descriptor for each and make the descriptor both as invariant and as distinctive as possible. (Competing goals) SIFT one example.



Another version of the problem…

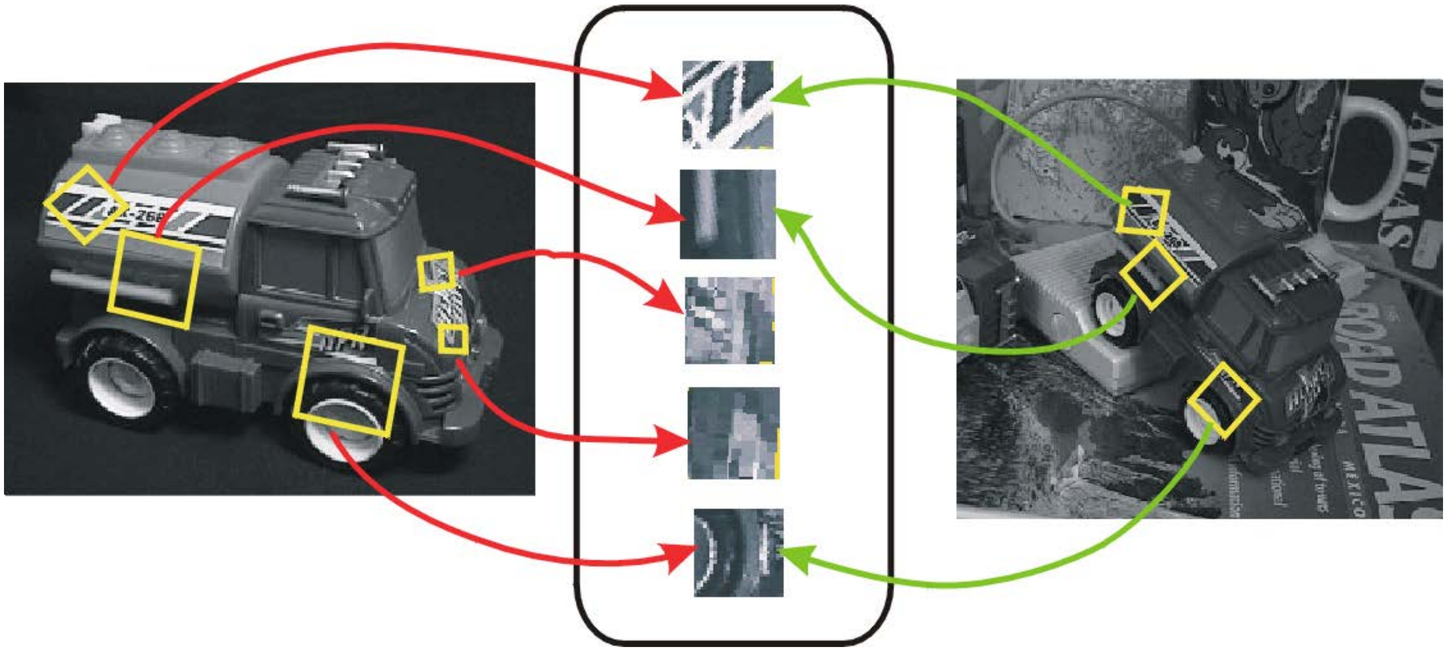
Want to find



… in here

Idea of SIFT

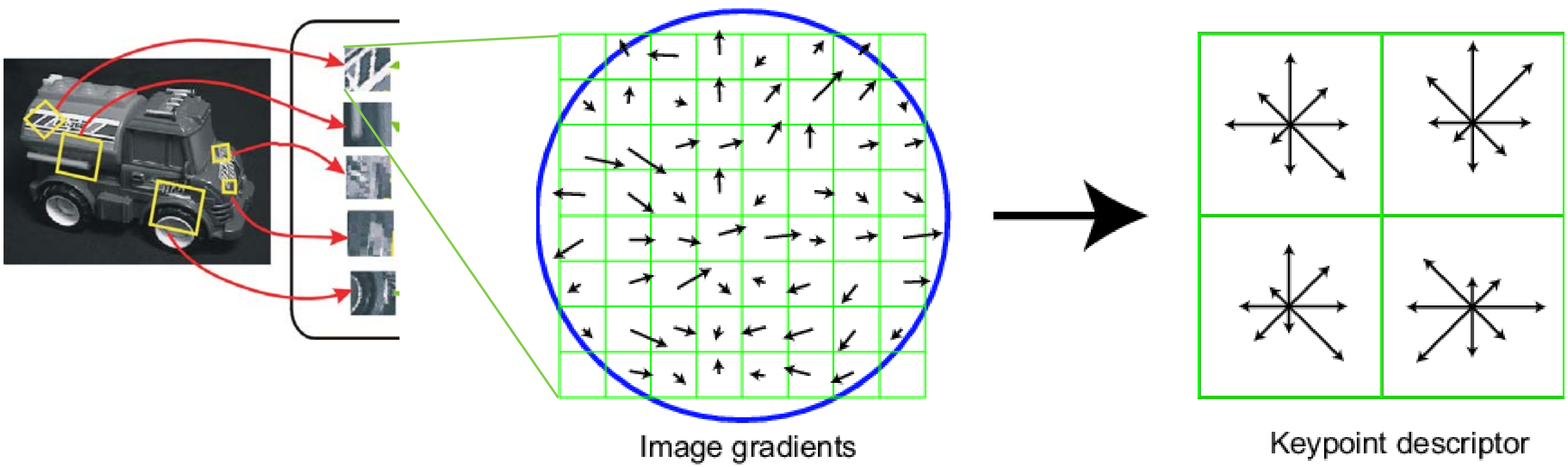
- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

SIFT vector formation

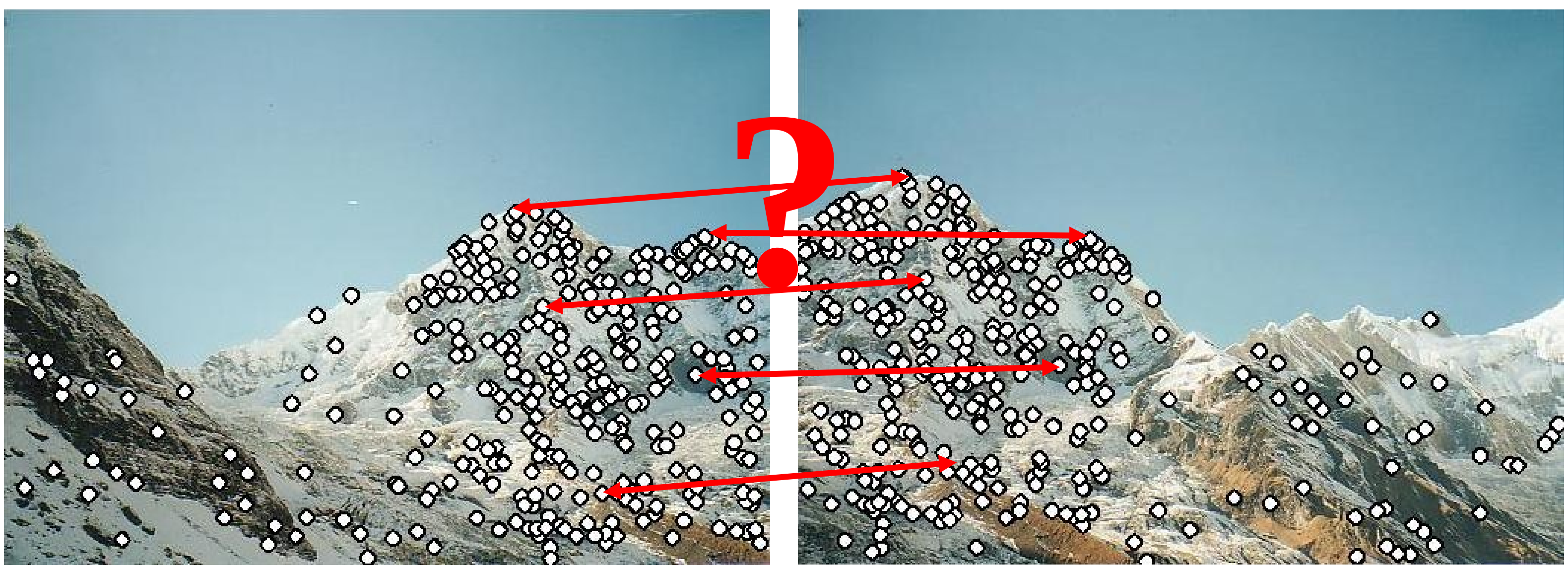
- 4x4 array of gradient orientation histograms over 4x4 pixels
  - not really histogram, weighted by magnitude
- 8 orientations x 4x4 array = 128 dimensions
- Motivation: some sensitivity to spatial layout, but not too much.



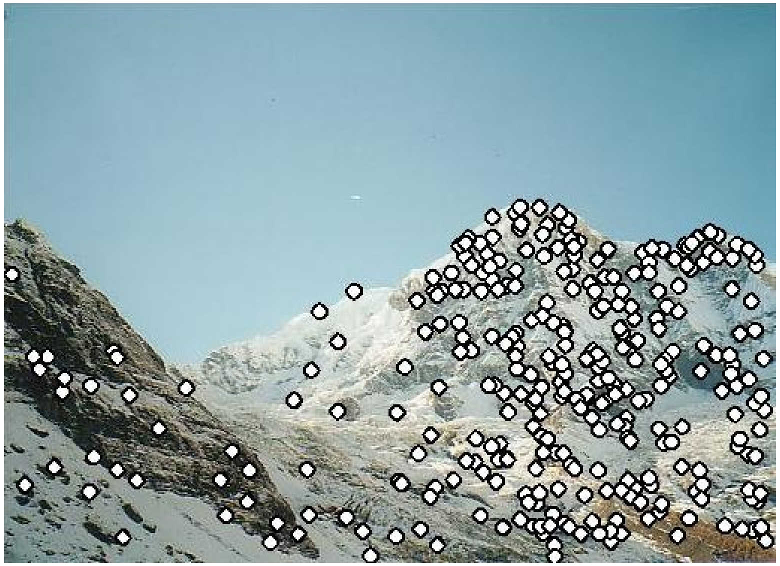
showing only 2x2 here but is 4x4

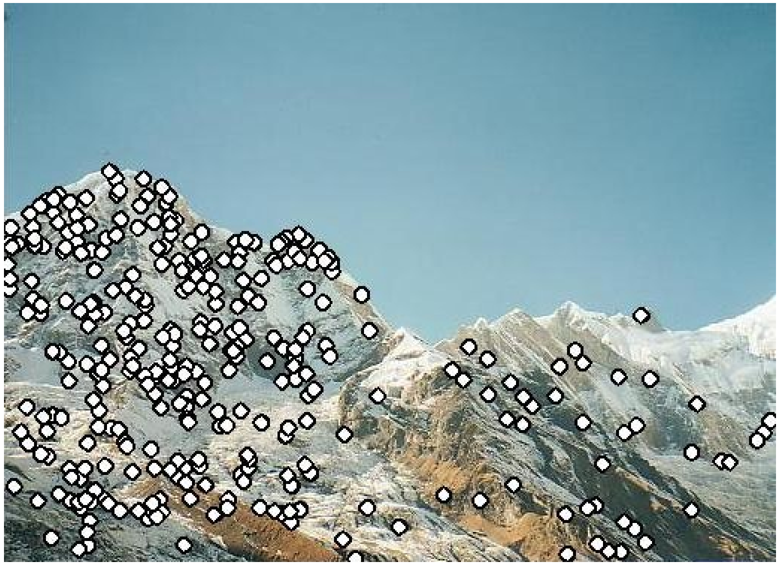
Point Descriptors

- We know how to detect points
- How to match them? Two parts:
  - Compute a descriptor for each and make the descriptor both as invariant and as distinctive as possible. (Competing goals) SIFT one example
  - ***Need to figure out which point matches which***..



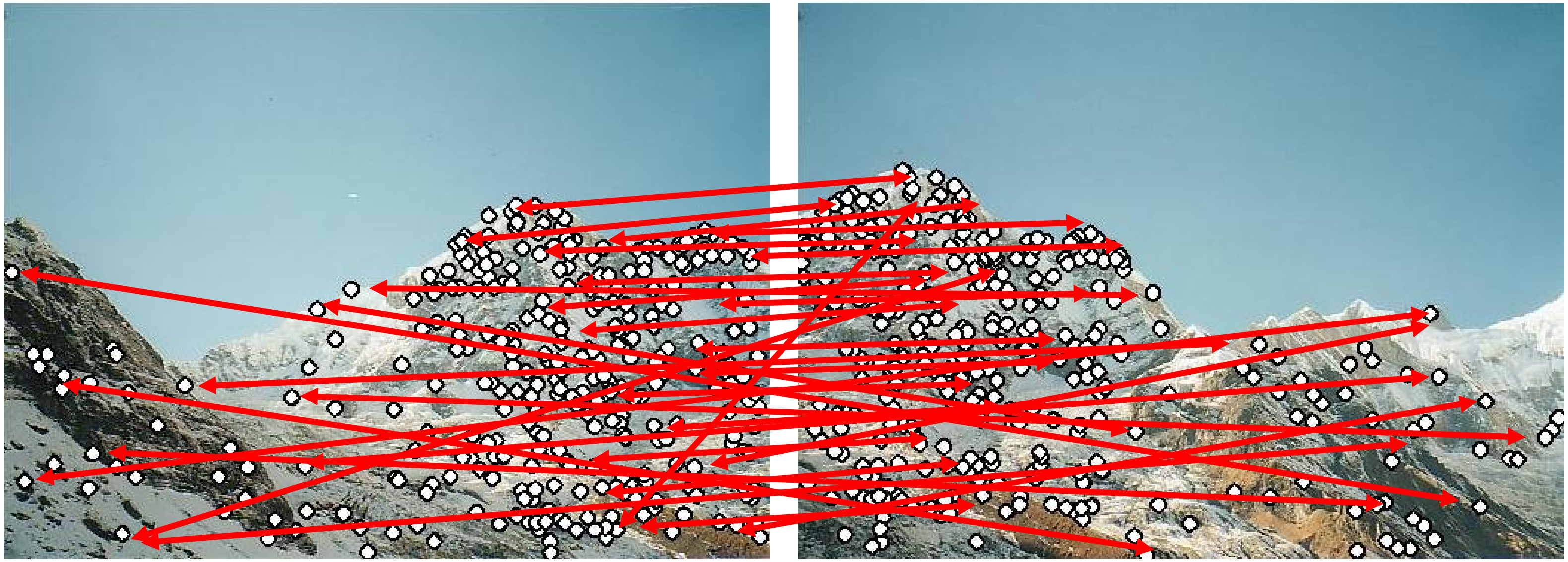
Feature-based alignment outline





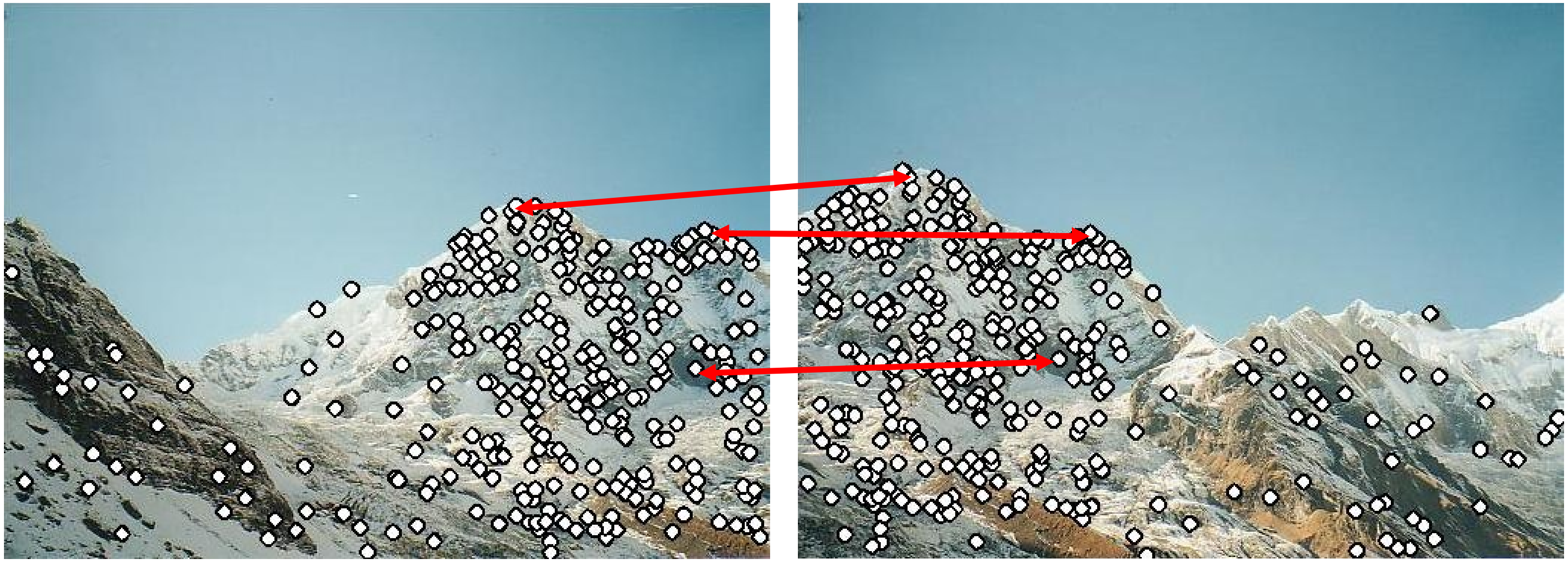
- Extract features

Feature-based alignment outline



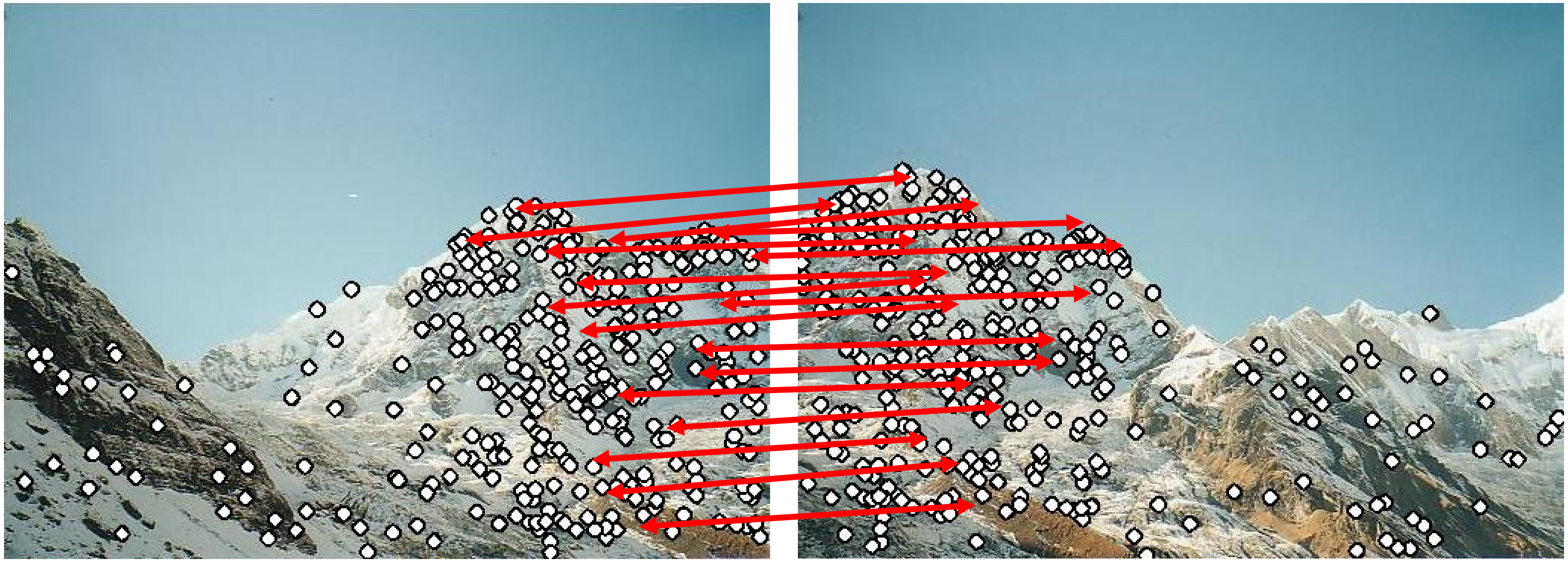
- Extract features
- Compute ***putative*** *matches* – e.g. “closest descriptor”

Feature-based alignment outline



- Extract features
- Compute ***putative*** *matches* – e.g. “closest descriptor”
- Loop:
  - *Hypothesize* transformation *T* from some matches

Feature-based alignment outline



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

Feature-based alignment outline

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

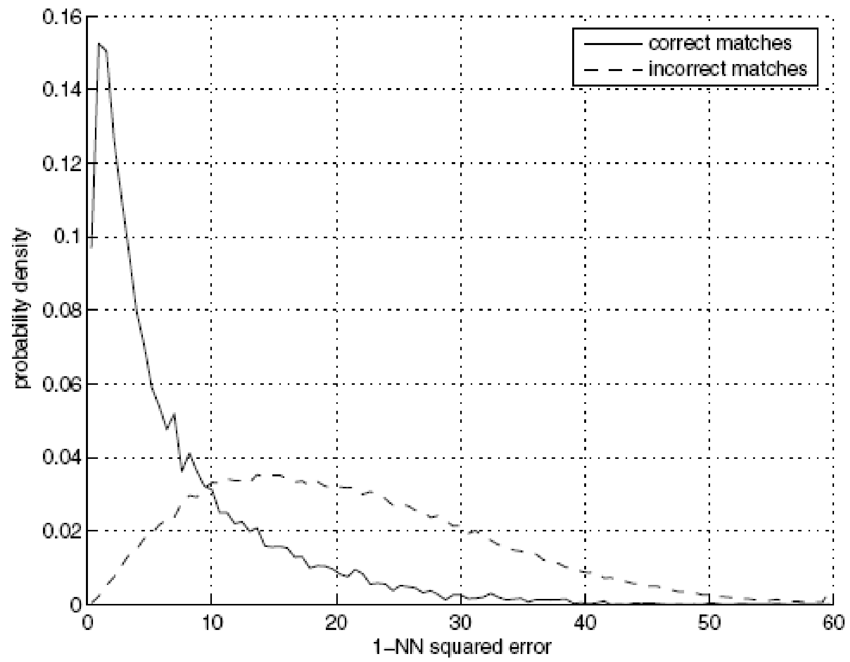
How to get “putative” matches?

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

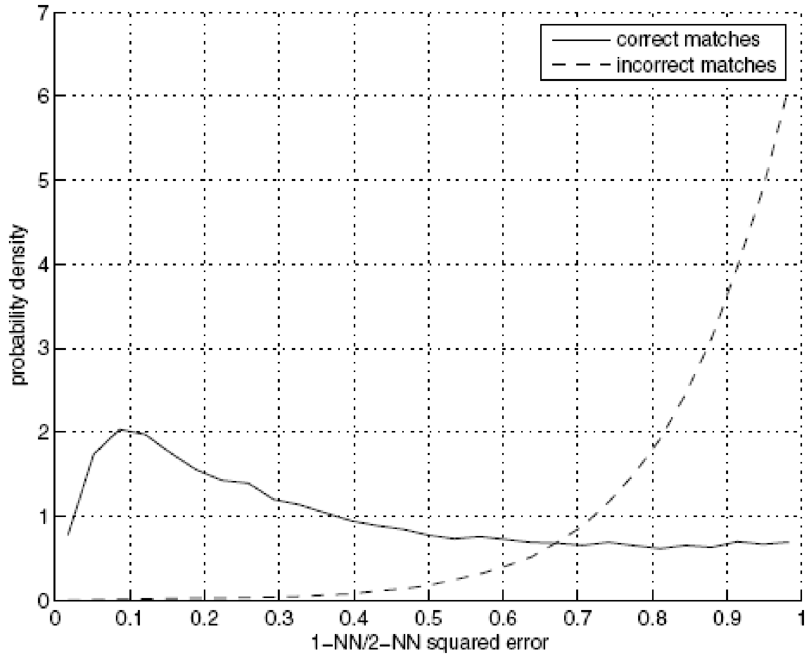
Feature-space outlier rejection

- 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 second-closest 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

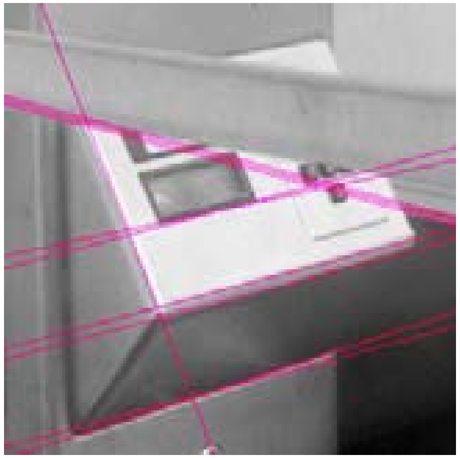
- Exhaustive search
  - for each feature in one image, look at *all* the other features in the other image(s) – pick best one
- Hashing
  - compute a short descriptor from each feature vector, or hash longer descriptors (randomly)
- Nearest neighbor techniques
  - *k*-trees and their variants
- But…
- Remember: distinctive vs invariant competition? Means:
- *Problem: Even when pick best match, still lots (and lots) of wrong matches – "outliers"*

Another way to remove mistakes

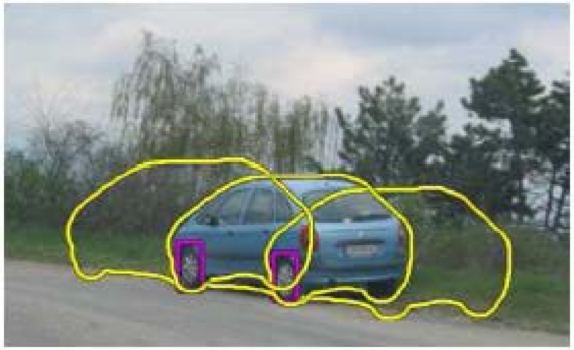
- Why are we doing matching?
  - To compute a model of the relation between entities
- So this is really “model fitting”

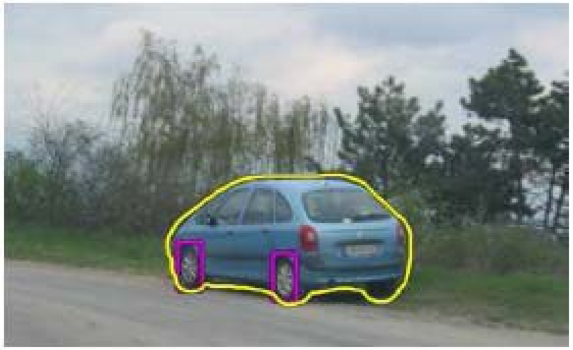
Fitting

- 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

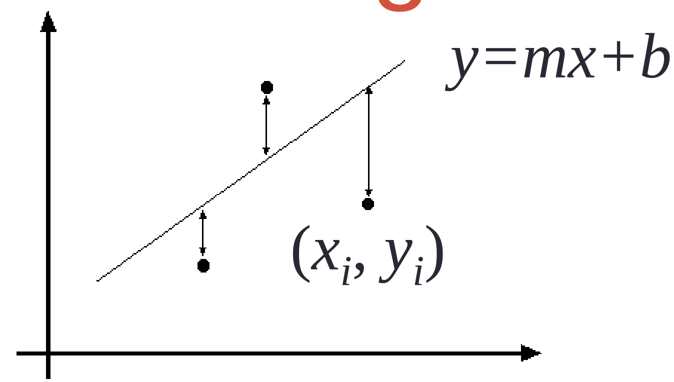
Case study: Line detection

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

Slide: S. Lazebnik

Typical least squares line fitting

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



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

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

$$= (\mathbf{y} - \mathbf{X}\mathbf{b})^T (\mathbf{y} - \mathbf{X}\mathbf{b}) = \mathbf{y}^T\mathbf{y} - 2(\mathbf{X}\mathbf{b})^T\mathbf{y} + (\mathbf{X}\mathbf{b})^T(\mathbf{X}\mathbf{b})$$

$$\frac{dE}{d\mathbf{b}} = 2\mathbf{X}^T\mathbf{X}\mathbf{b} - 2\mathbf{X}^T\mathbf{y} = 0$$

$$\boxed{\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y} \Rightarrow \mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$

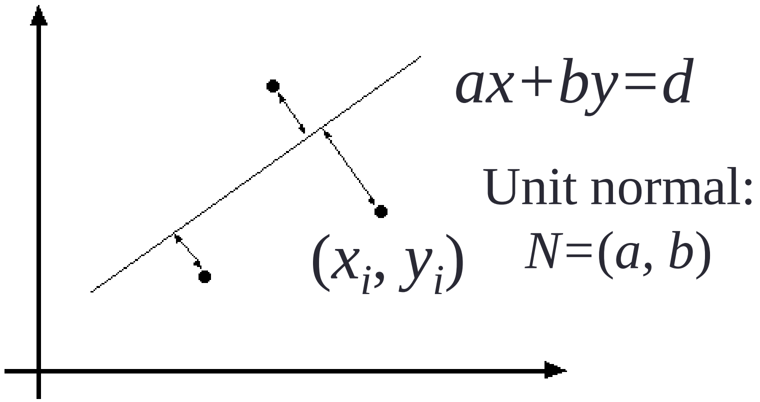
Standard least squares solution to except weird switch of typical names from $\mathbf{A}\mathbf{x}=\mathbf{b}$

Problem with “vertical” least squares

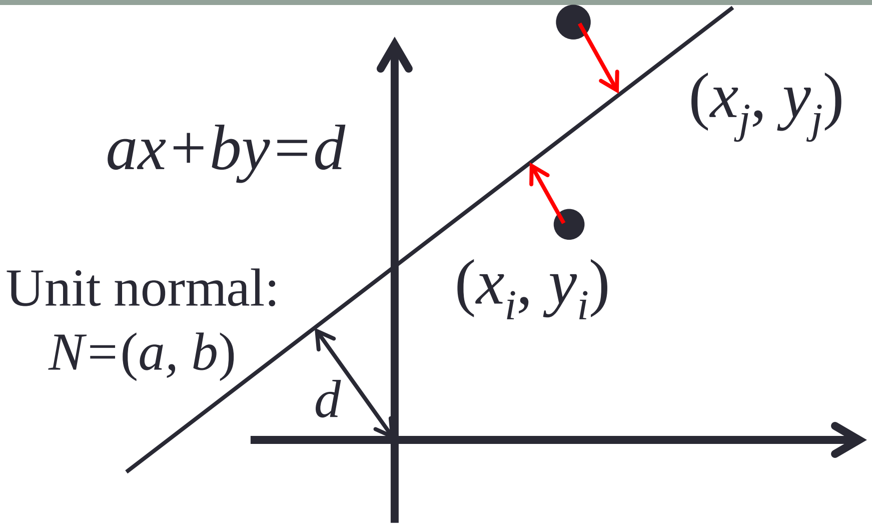
- Not rotation-invariant
- Fails completely for vertical lines

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



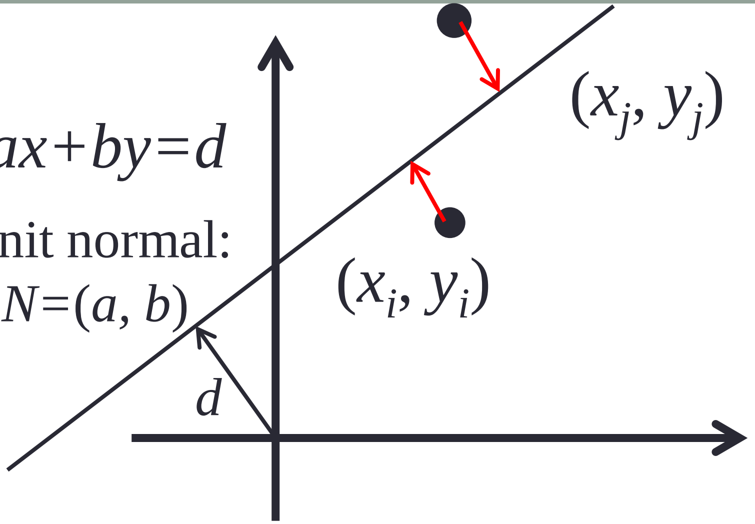
- Distance between point (x_i, y_i) and line $ax+by=d$

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

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

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



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

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

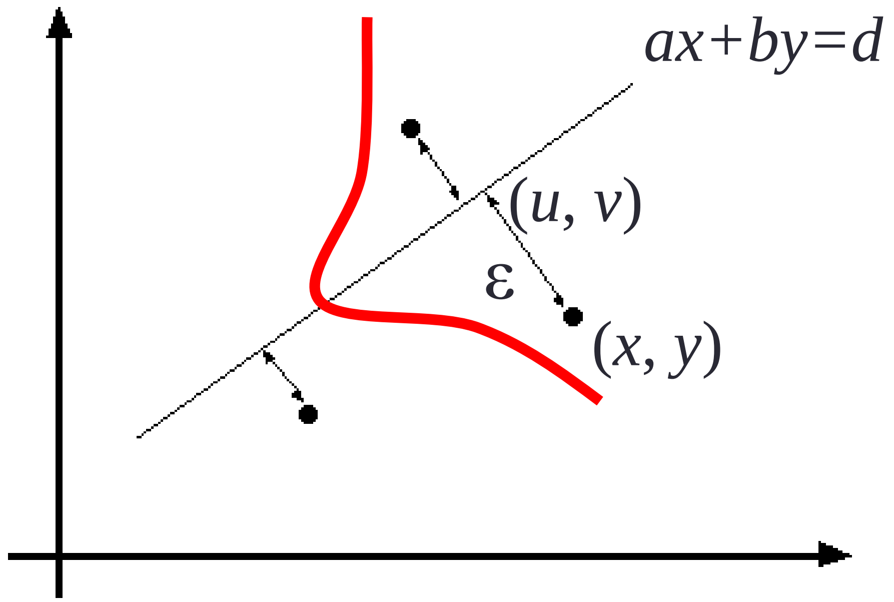
$$E = \sum_{i=1}^{n} (a(x_i - \bar{x}) + b(y_i - \bar{y}))^2 = \left\| \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \right\|^2 = (UN)^T (UN)$$

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

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

Least squares as likelihood maximization

- **Generative model**: line points are corrupted by Gaussian noise in the direction perpendicular to the line



$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + \varepsilon \begin{pmatrix} a \\ b \end{pmatrix}$$

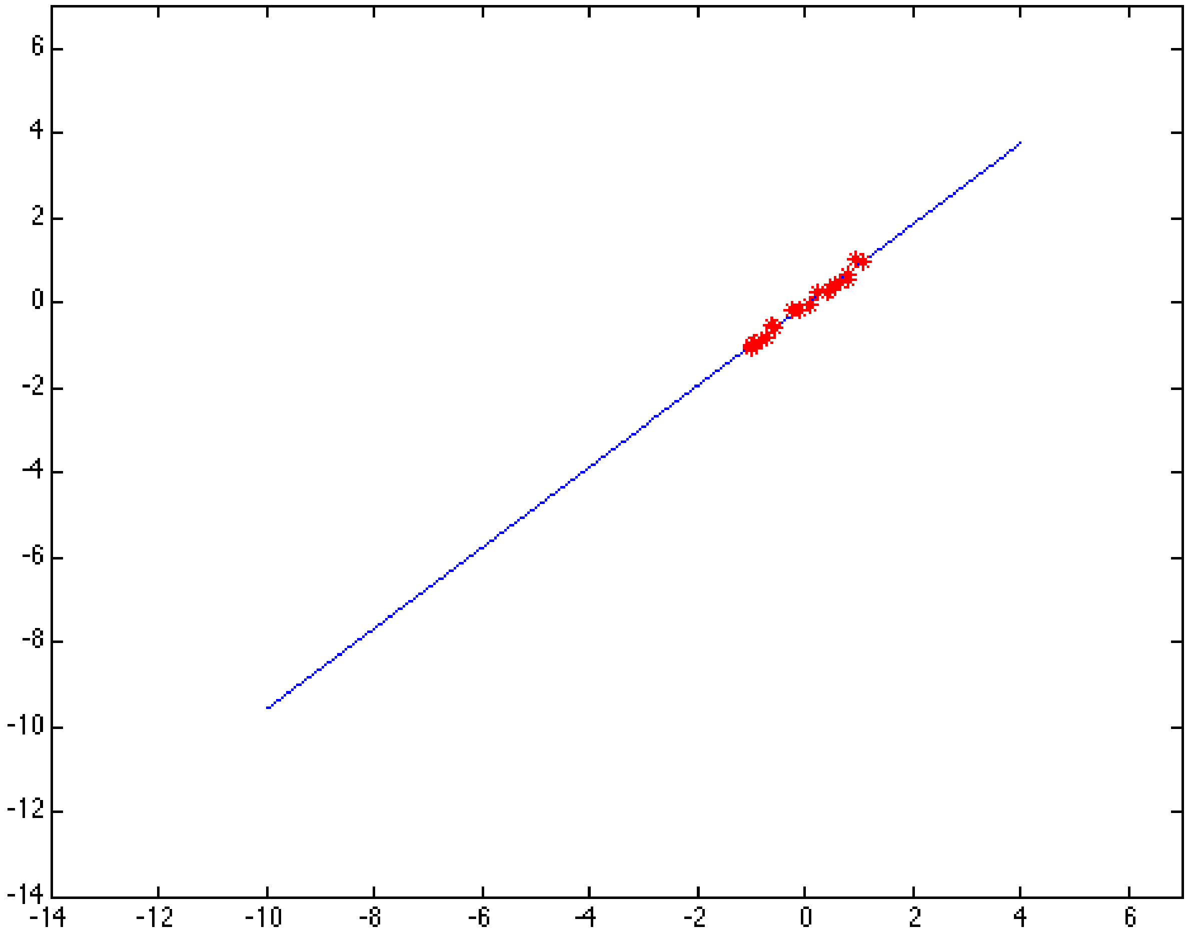
point on the line

noise: sampled from zero-mean Gaussian with std. dev. σ

normal direction

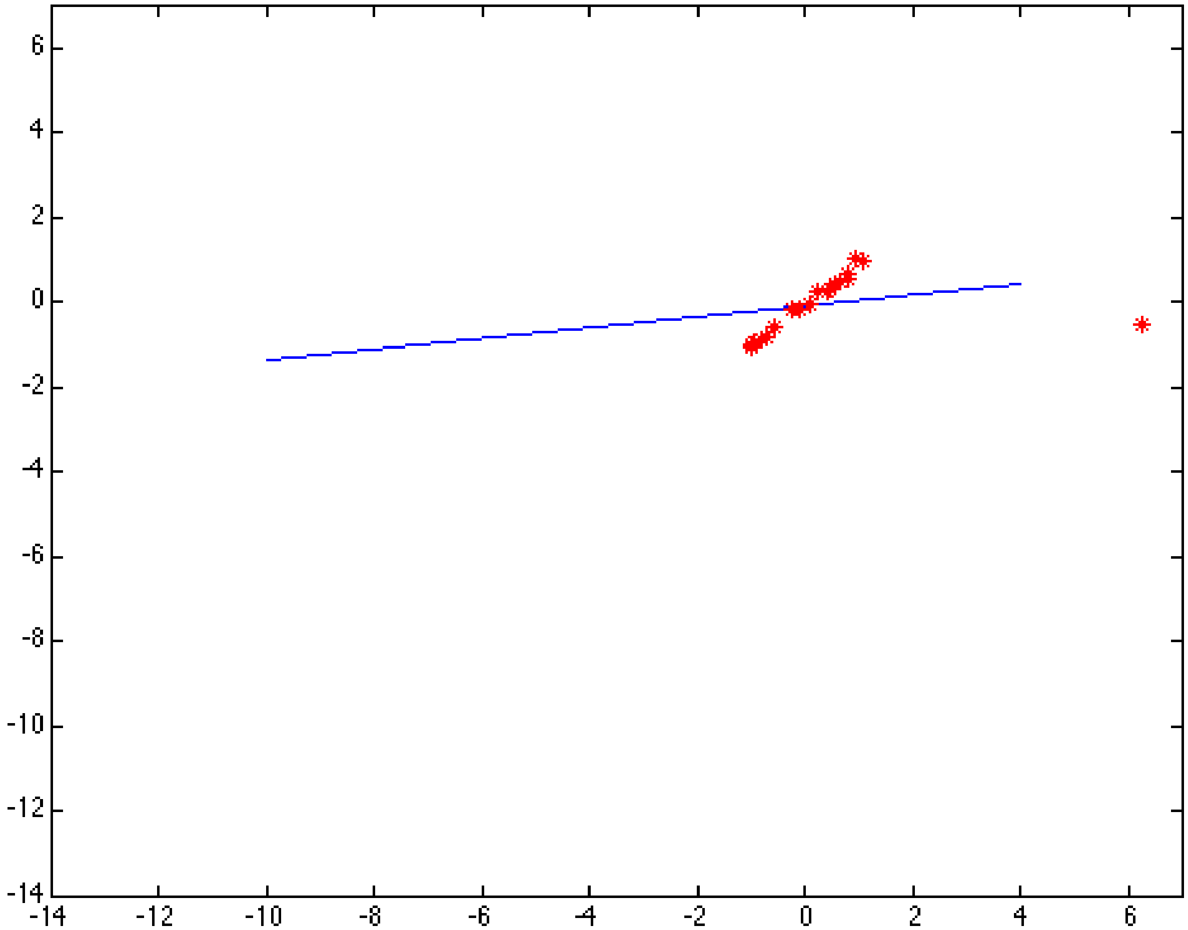
Least squares: Robustness to (very) non-Gaussian noise

- Least squares fit to the red points:



Least squares: Robustness to (very) non-Gaussian noise

- Least squares fit with an outlier:

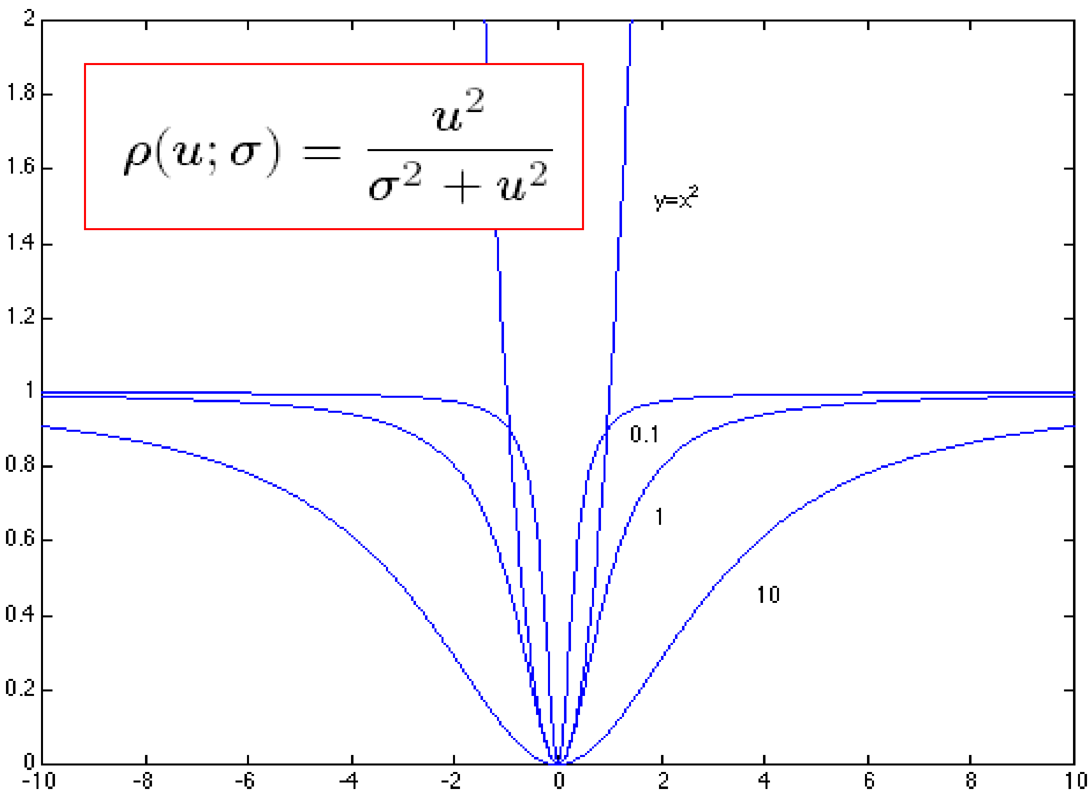


Problem: squared error heavily penalizes outliers

Robust estimators

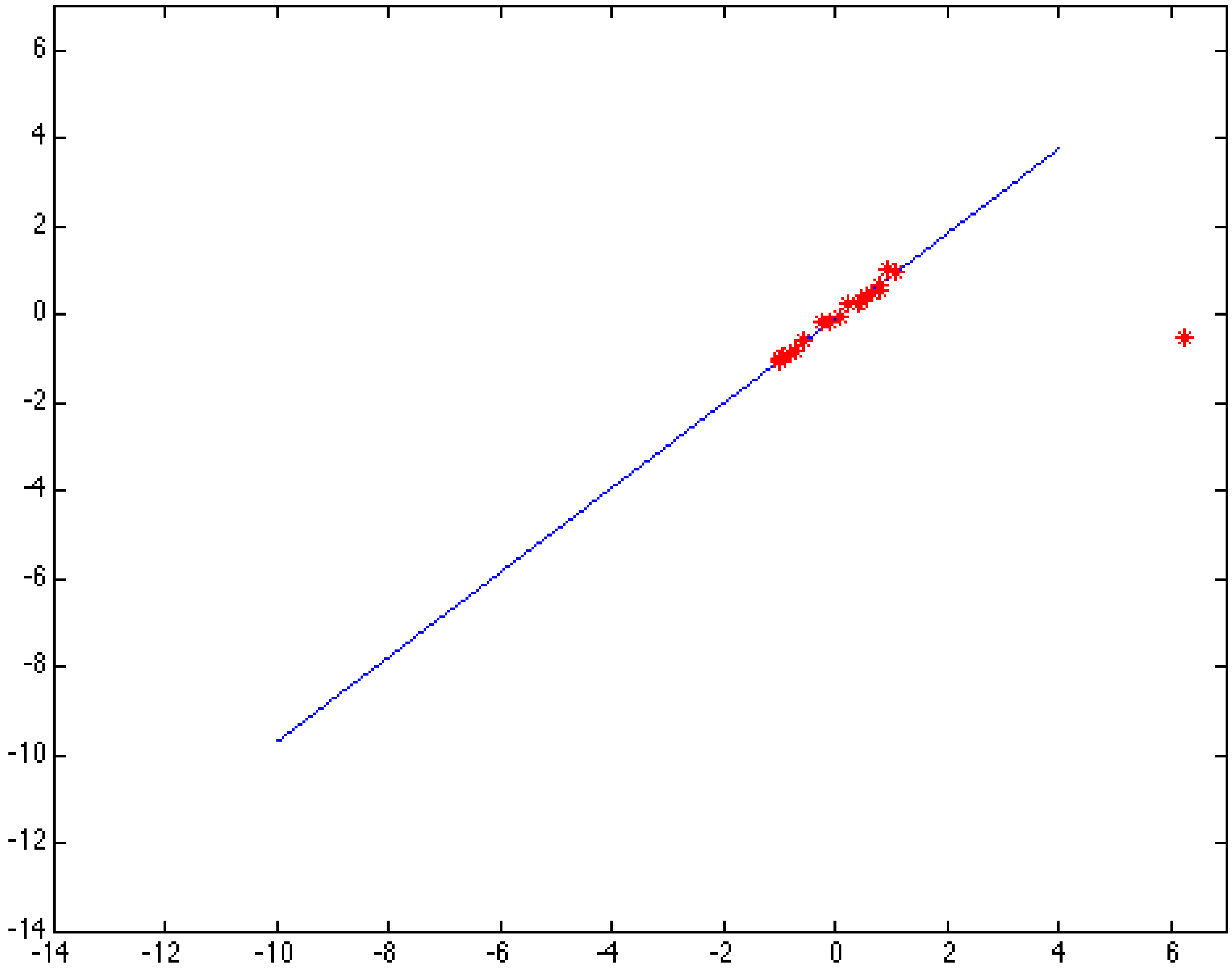
- General approach: minimize $\sum_i \rho\left(r_i\left(x_i, \theta\right); \sigma\right)$

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



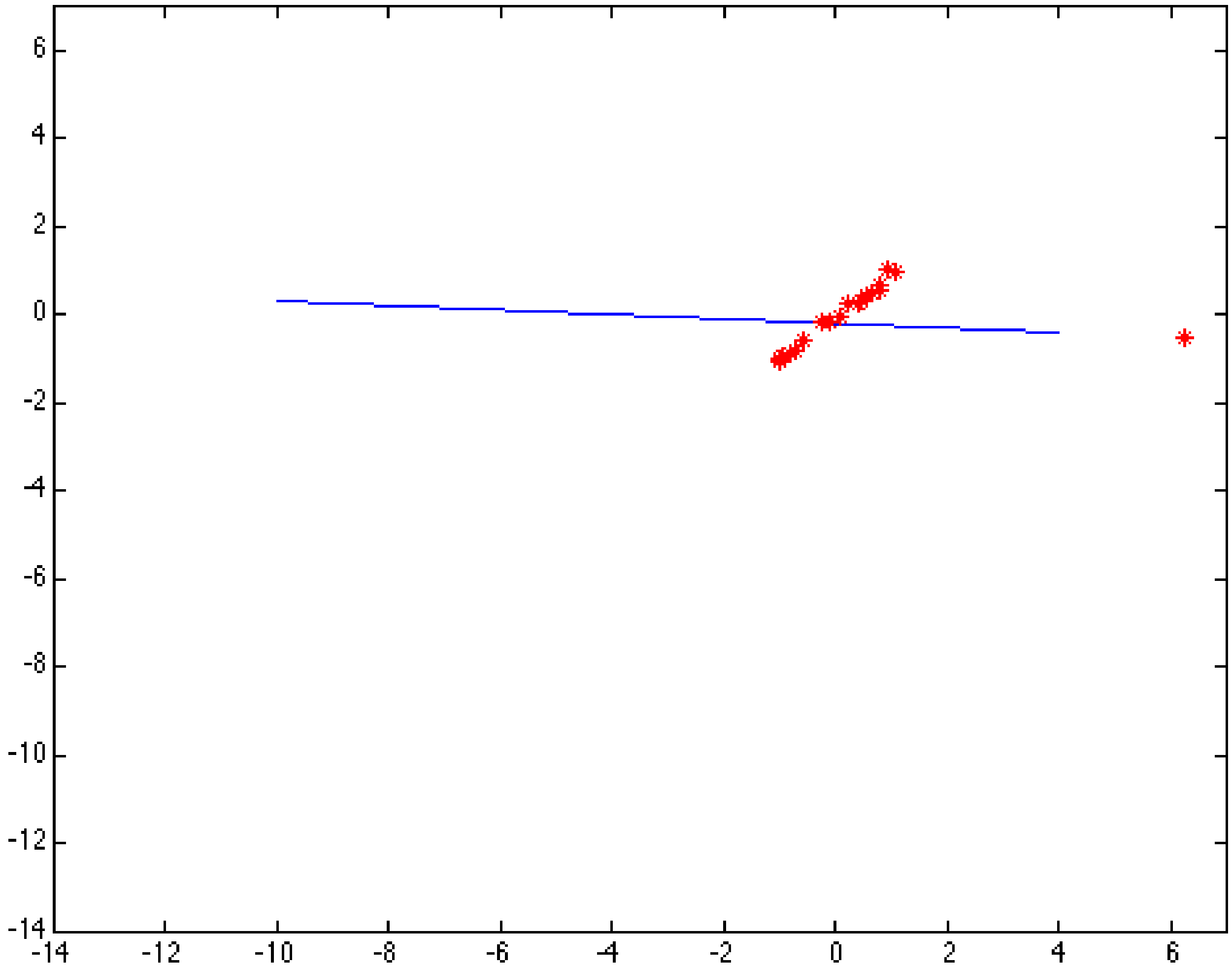
The robust function ρ behaves like squared distance for small values of the residual u but saturates for larger values of u

Choosing the scale: Just right



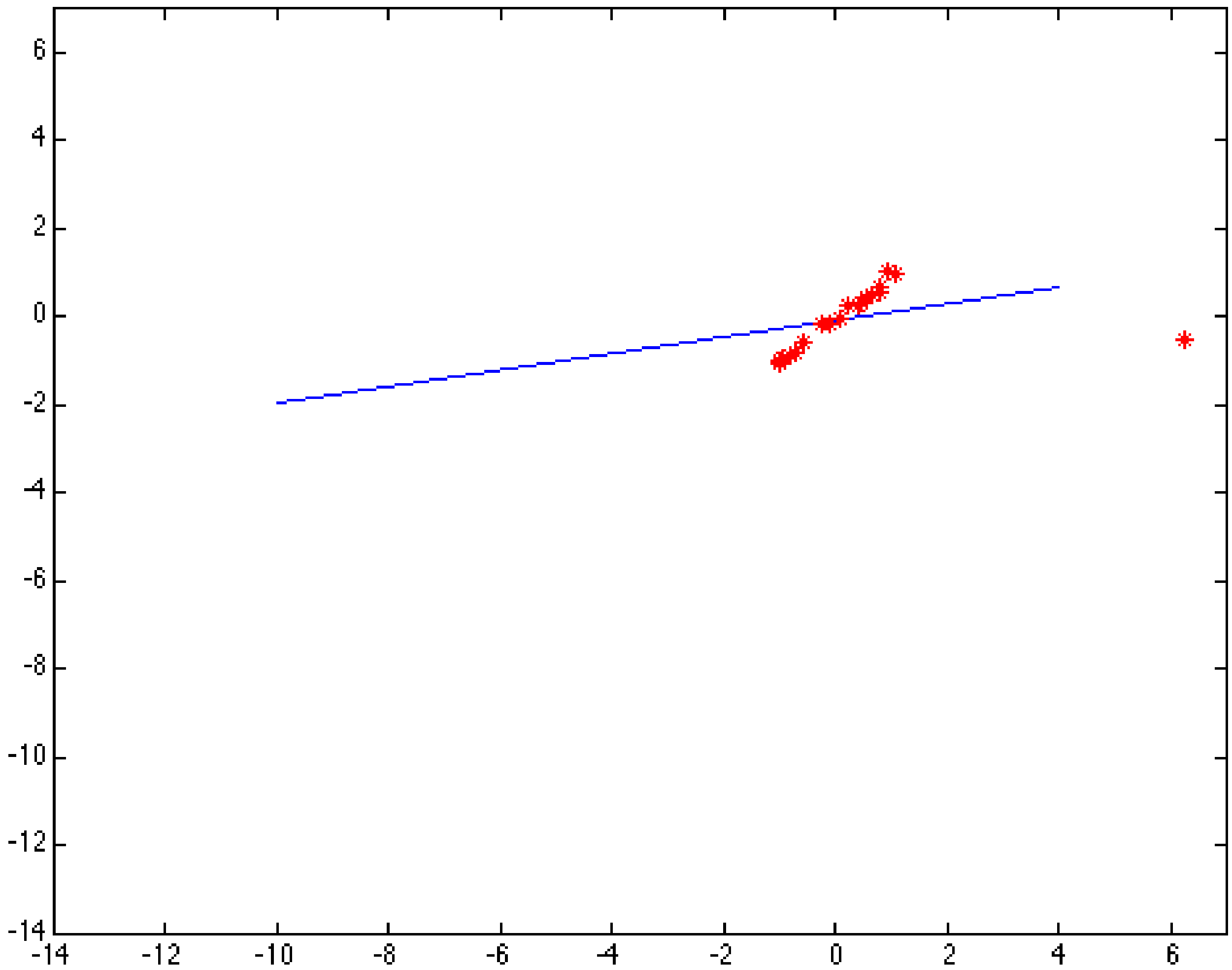
The effect of the outlier is minimized

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



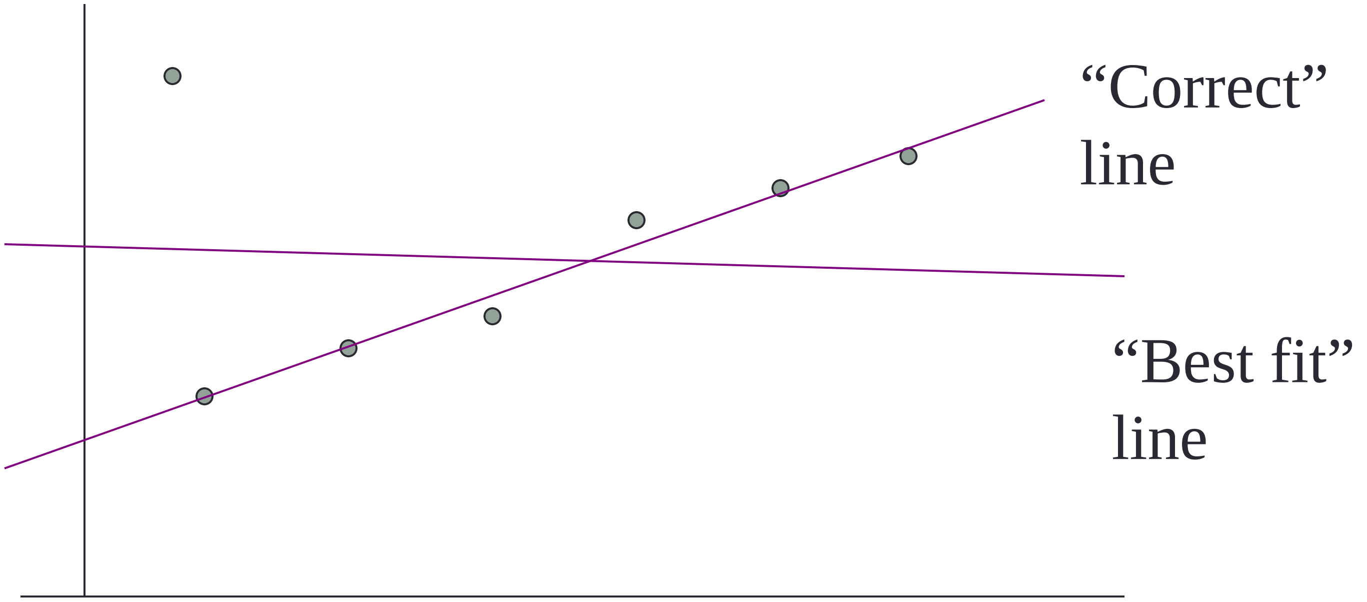
Behaves much the same as least squares

“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

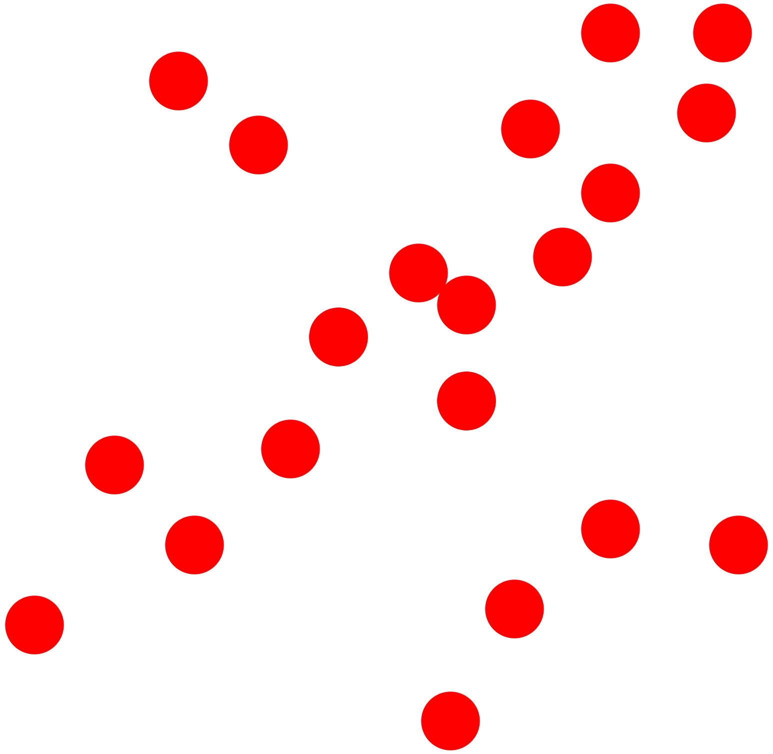
- No point with d>t
- RANSAC:
  - RANdom SAmple Consensus
  - Fischler & Bolles 1981
  - Copes with a large proportion of outliers

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:

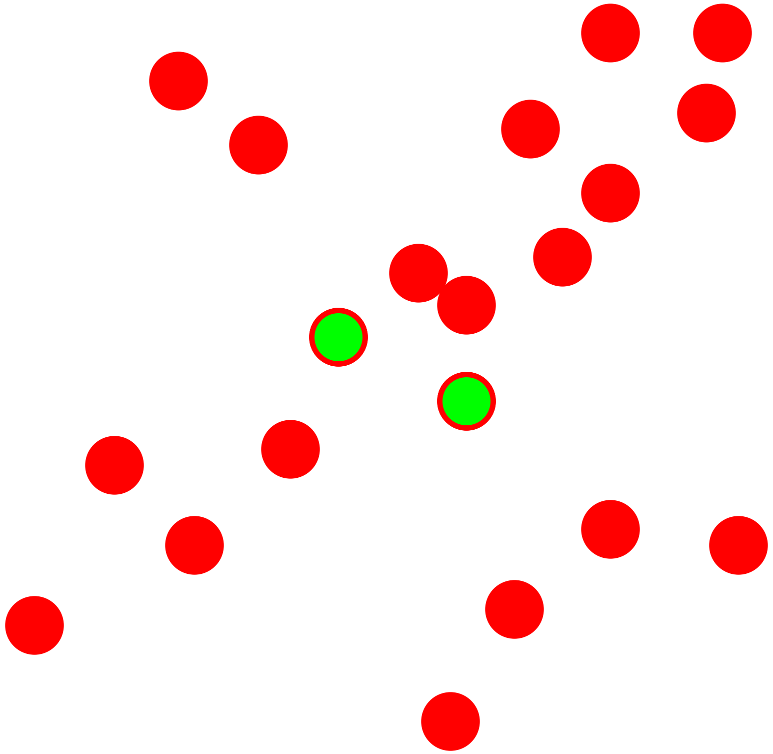
1. **Sample** (randomly) the number of points required to fit the model
2. **Solve** for model parameters using sample
3. **Score** by the fraction of ***inliers*** within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



Algorithm:

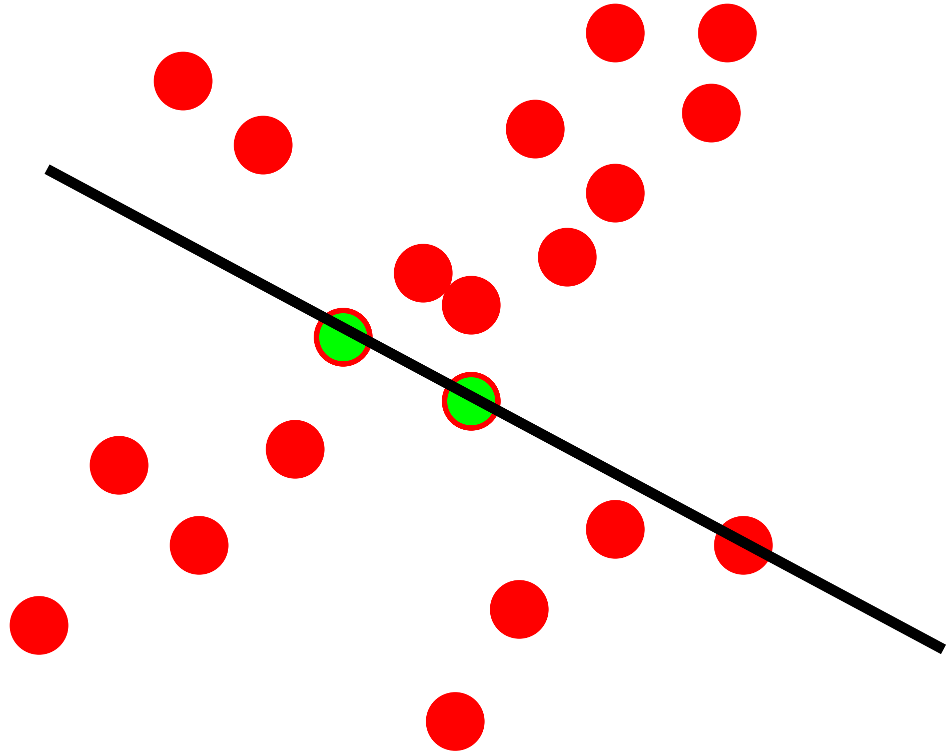
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using sample
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



Algorithm:

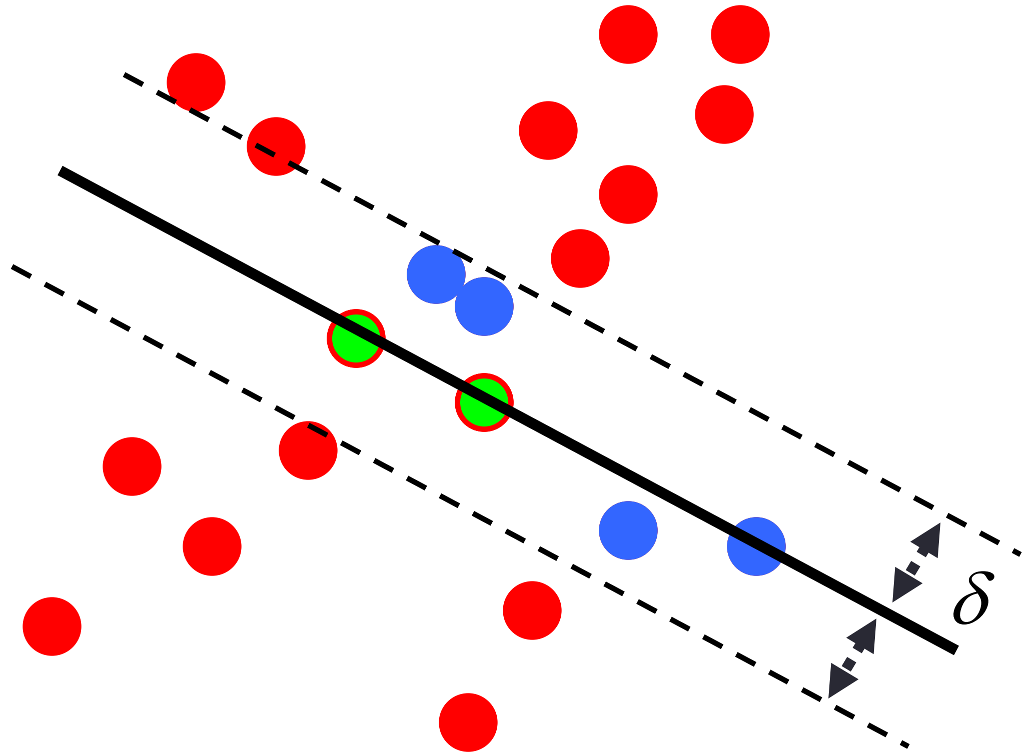
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using sample
3. **Score** by the fraction of inliers within a preset threshold of the model

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

Illustration by Savarese

RANSAC

Line fitting example



$$N_I = 6$$

Algorithm:

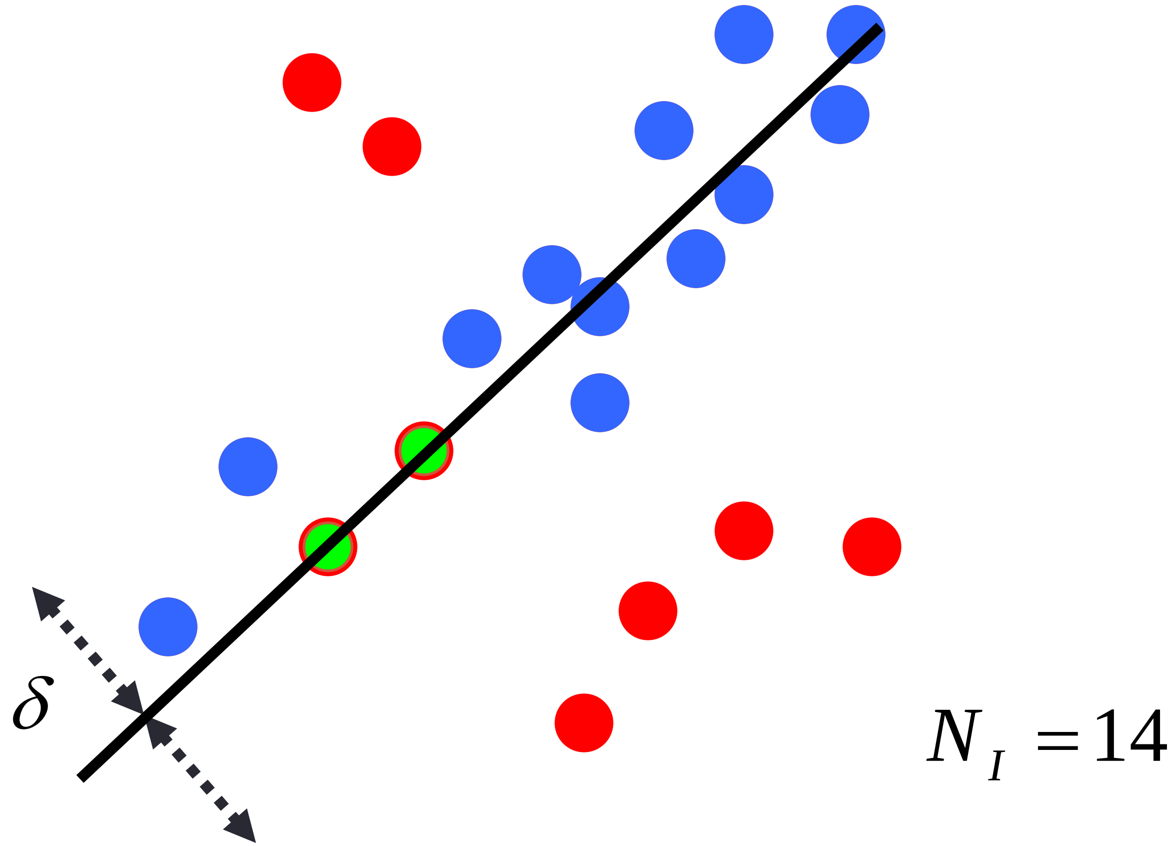
1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using the sample
3. **Score** by the fraction of inliers within a preset threshold of the model

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 sample
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

Best Line has most support

- More support -> better fit

RANSA for general model

- A given model has a *minimal set* – the smallest number of samples from which the model can be computed.
  - Line: 2 points
- Image transformations are models. Minimal set of *s* of point pairs/matches:
  - Translation: pick one point pair
  - Homography (for plane) – pick 4 point pairs
  - Fundamental matrix – pick 8 point pairs (really 7 but lets not go there)
- Algorithm
  - Randomly select s points (or point pairs) to form a sample
  - 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

- If **Location**: Gaussian noise with $\sigma^2 = 1$
- Then **Distance** d has ***Chi*** distribution with k degrees of freedoms
- If one dimension, e.g.distance off a line, then 1DOF

$$f(t) = \frac{\sqrt{2}e^{-\frac{t^2}{2}}}{\sqrt{\pi}}, \mathrm{t} \geq 0$$

  - For 95% cumulative threshold when Gaussian with σ^2:

$$t^2 = 3.84\sigma^2$$

  - That is: if $t^2 = 3.84\sigma^2$ then 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 high (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 $P(all\ N\ samples\ have\ outlier) < (1 - p)$
- So $(1 - (1 - e)^s)^N < 1 - p$

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

Samples required for inliers only in a sample

- Set p=0.99 – chance of getting good sample

$s = 2, e = 5\%$ => N=2

$s = 2, e = 50\%$ => N=17

$s = 4, e = 5\%$ => N=3

$s = 4, e = 50\%$ => N=72

$s = 8, e = 5\%$ => N=5

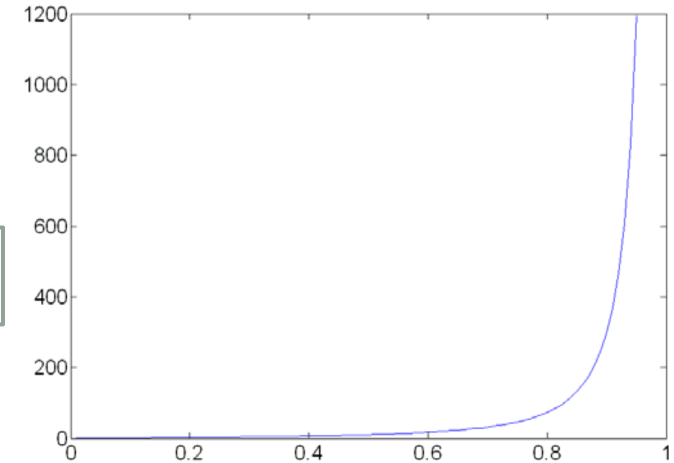
$s = 8, e = 50\%$ => N=1177

	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

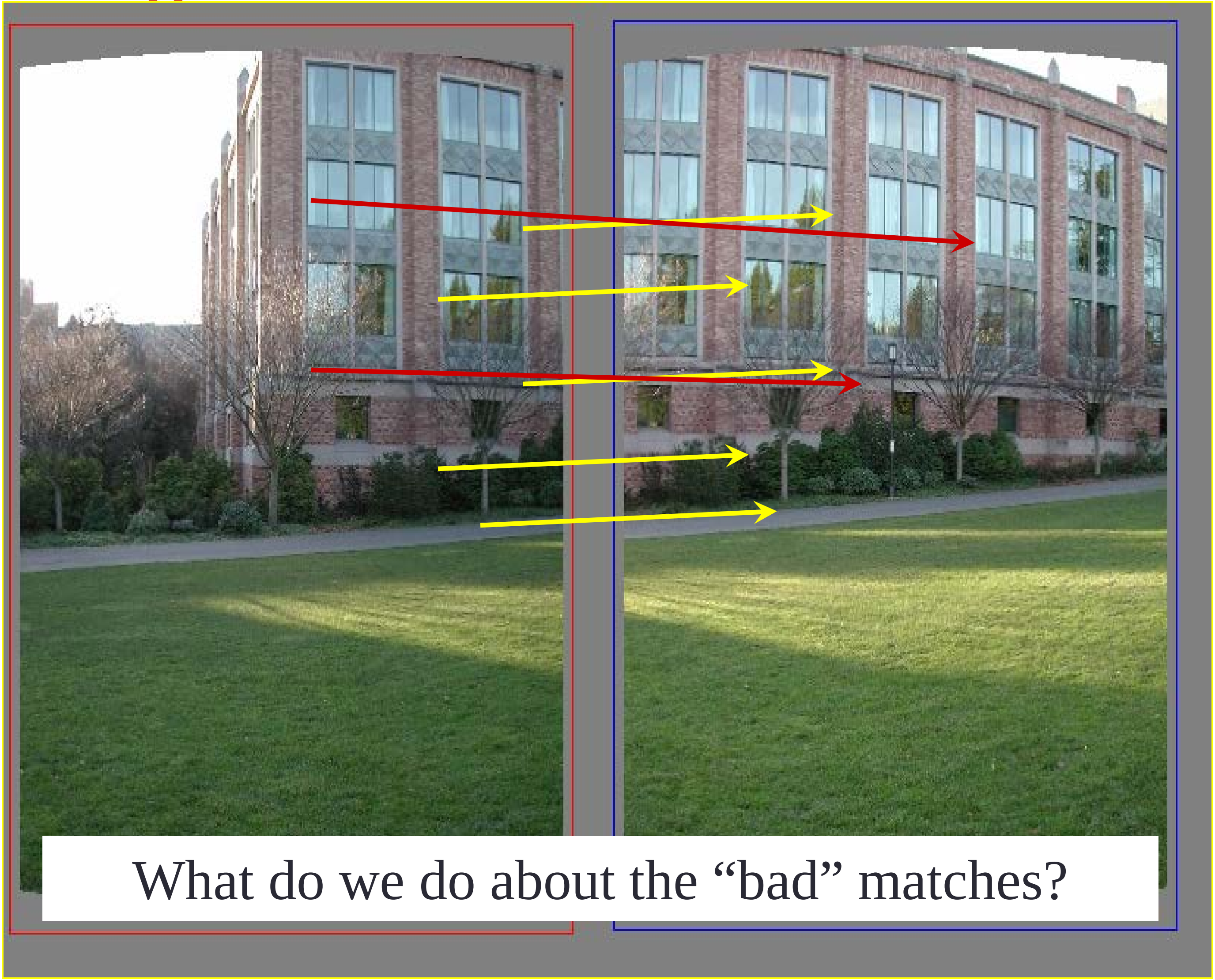
- N increases steeply with s

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

$$N = f(e),$$ ***not the number of points!***

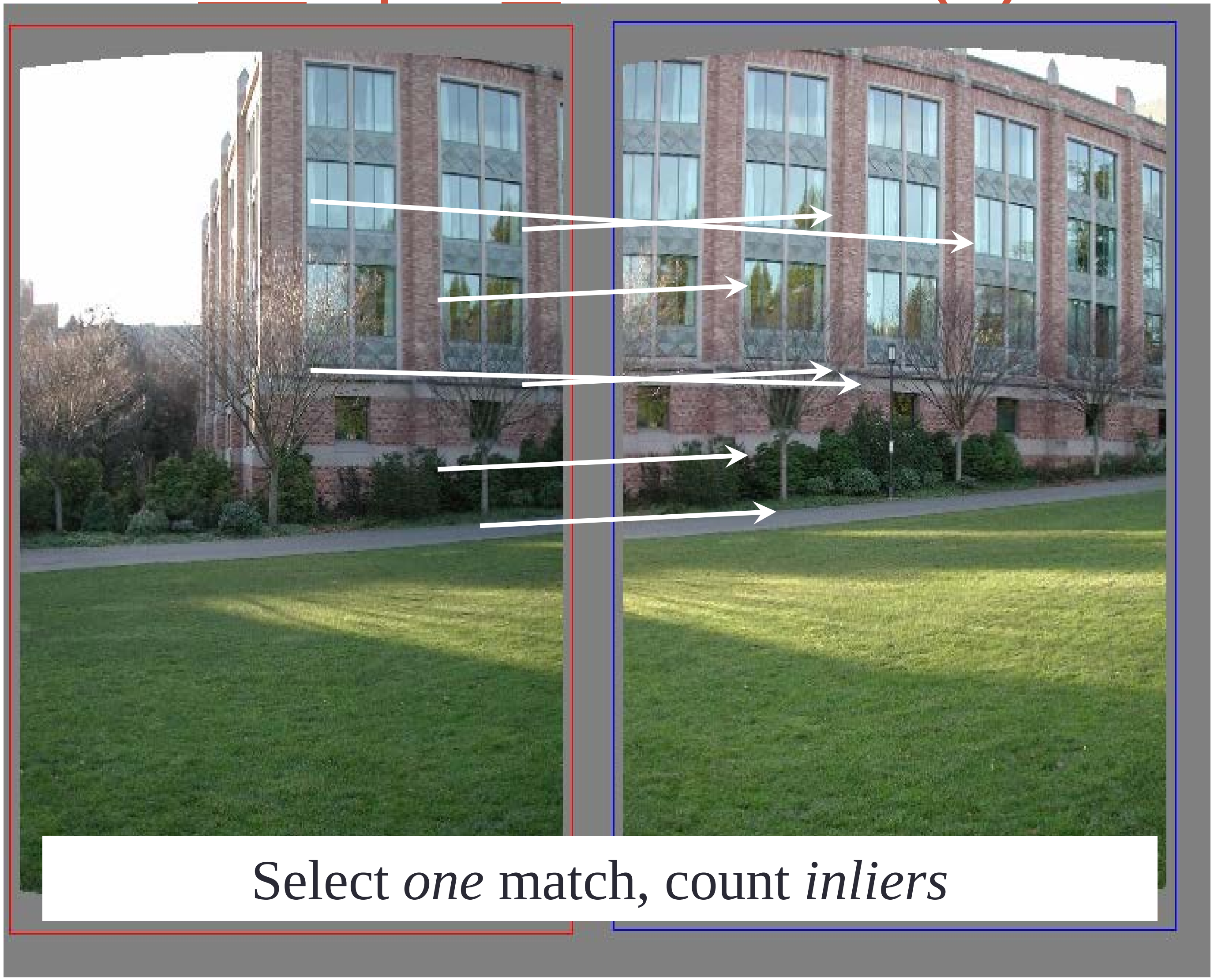


Matching features



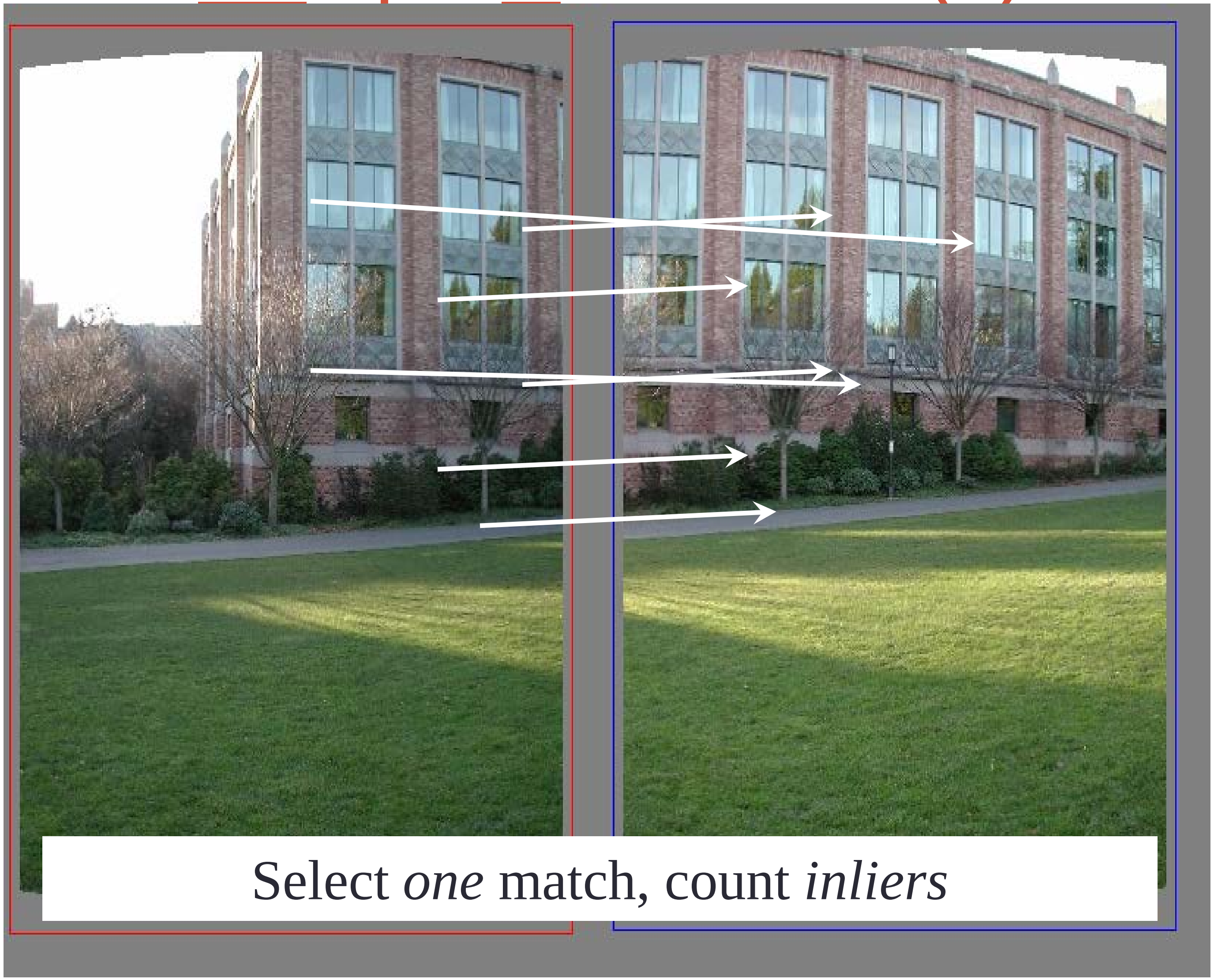
What do we do about the “bad” matches?

RAndom SAmple Consensus (1)

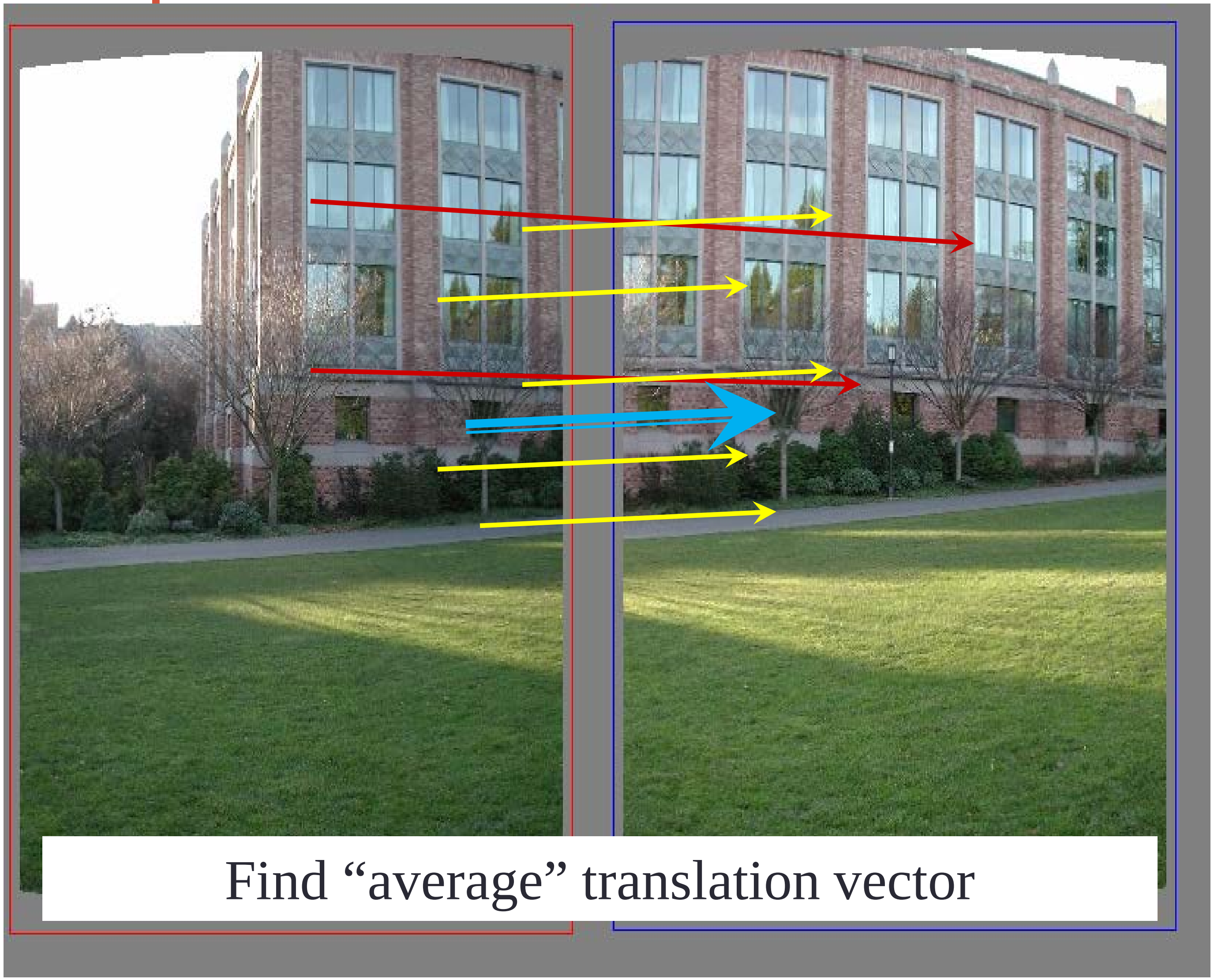


Select *one* match, count *inliers*

RAndom SAmple Consensus (2)



Least squares fit



Find "average" translation vector

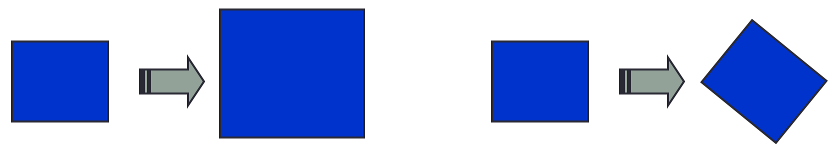
RANSAC for estimating homography

- RANSAC loop:

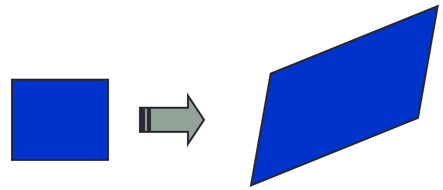
1. Select four feature pairs (at random)
2. Compute homography $\boldsymbol{H}$ (exact)
3. Compute *inliers* where $SSD(pi', \boldsymbol{H}\, p_i) < \varepsilon$
4. Keep largest set of inliers
5. Re-compute least-squares $\boldsymbol{H}$ estimate on all of the inliers

2D transformation models

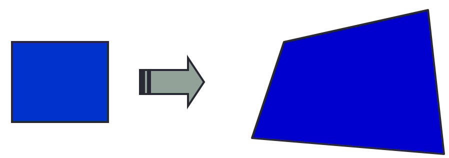
- Similarity (translation, scale, rotation)



- 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, $e = 1.0$
  - While N >sample_count
    - Choose a sample and count the number of inliers
    - Set $e_0 = 1 - (number\ of\ inliers)/(total\ number\ of\ points)$
    - If $e_0 < e$ Set $e = e_0$ and 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 number of parameters
- Not as good for getting multiple fits
- Really not good for approximate models

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