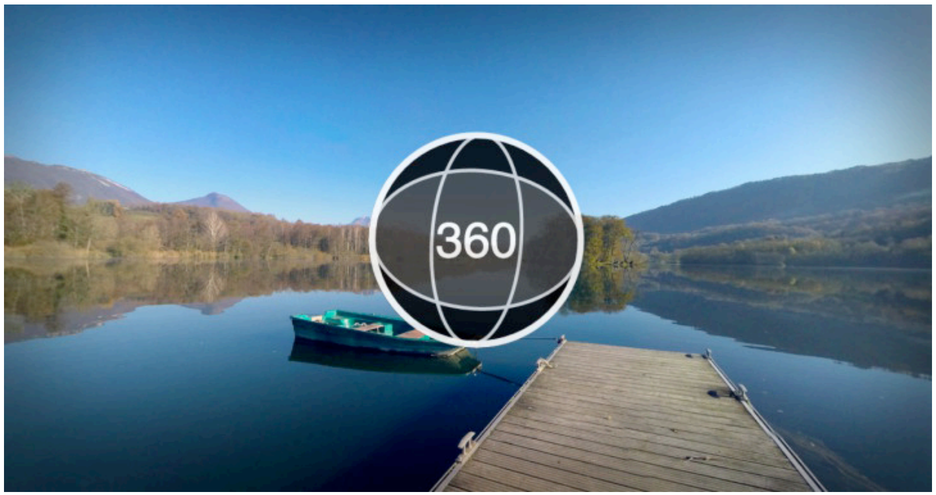
Fitting & Matching
Region Representation
Image Alignment, Optical Flow

Lectures 5 & 6 – Prof. Fergus

Slides from: S. Lazebnik, S. Seitz, M. Pollefeys, A. Effros.

Panoramas



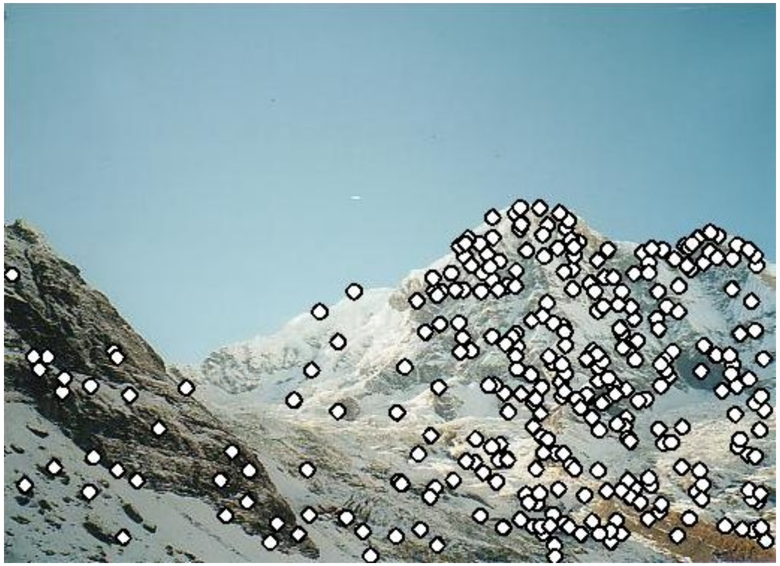
Facebook 360 photos

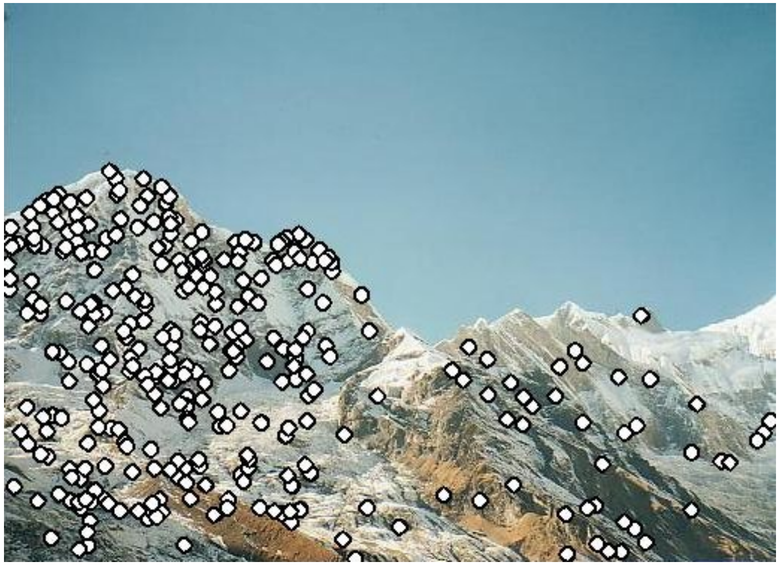
How do we build panorama?

- We need to match (align) images

Matching with Features

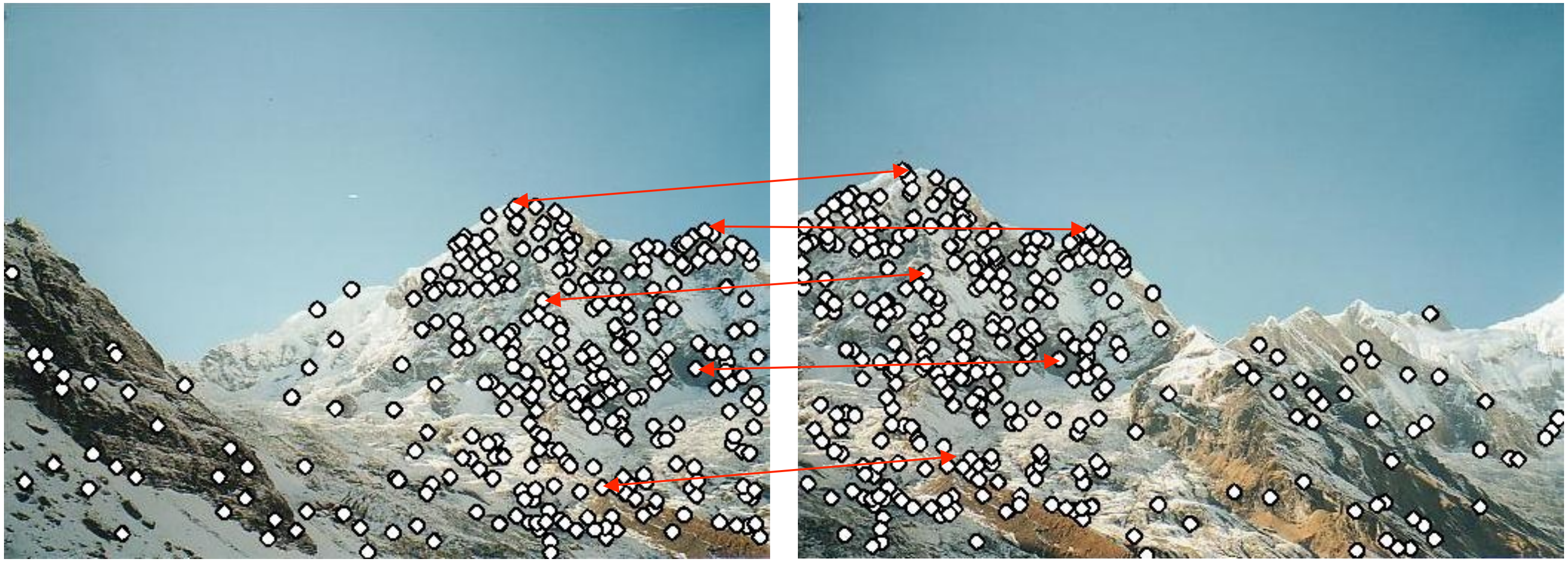
- Detect feature points in both images





Matching with Features

- Detect feature points in both images
- Find corresponding pairs



Matching with Features

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

Matching with Features

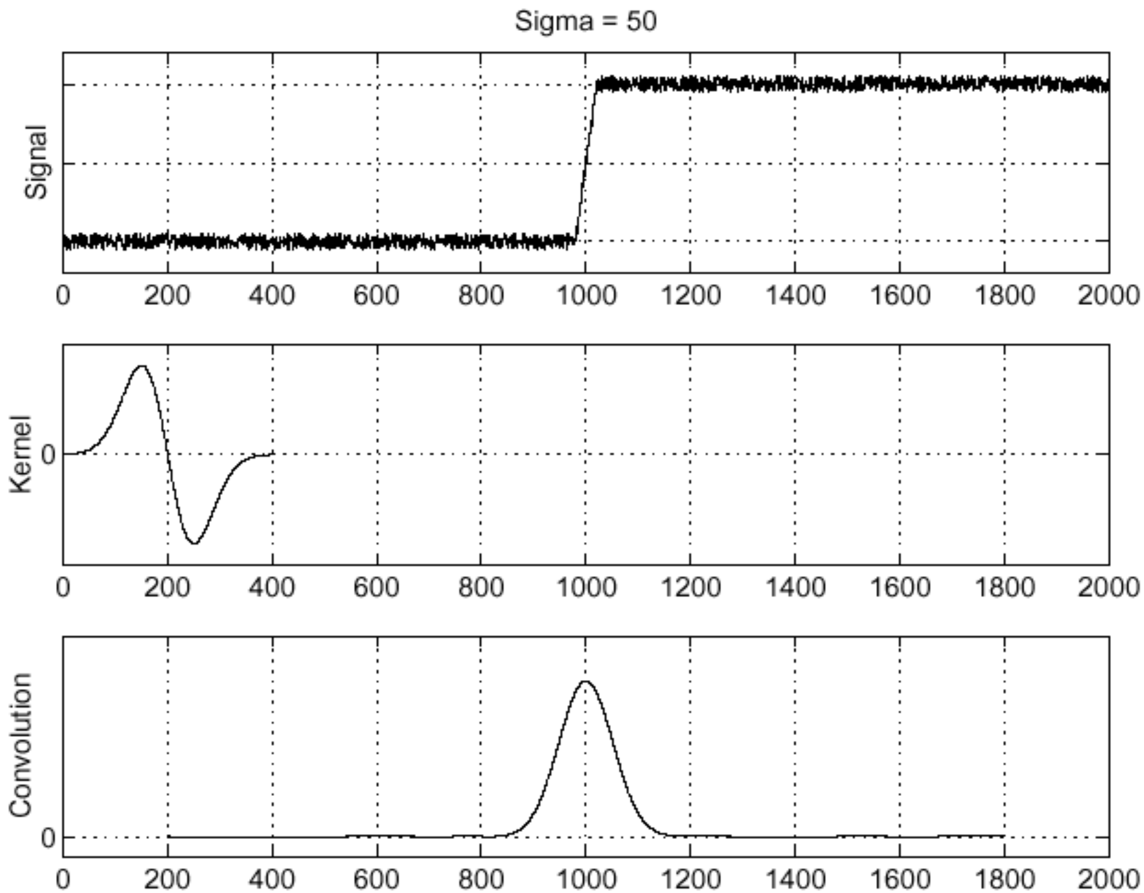
- Detect feature points in both images
- Find corresponding pairs
- Use these pairs to align images

Recall: Edge detection

f

$\frac{d}{dx} g$

$f * \frac{d}{dx} g$



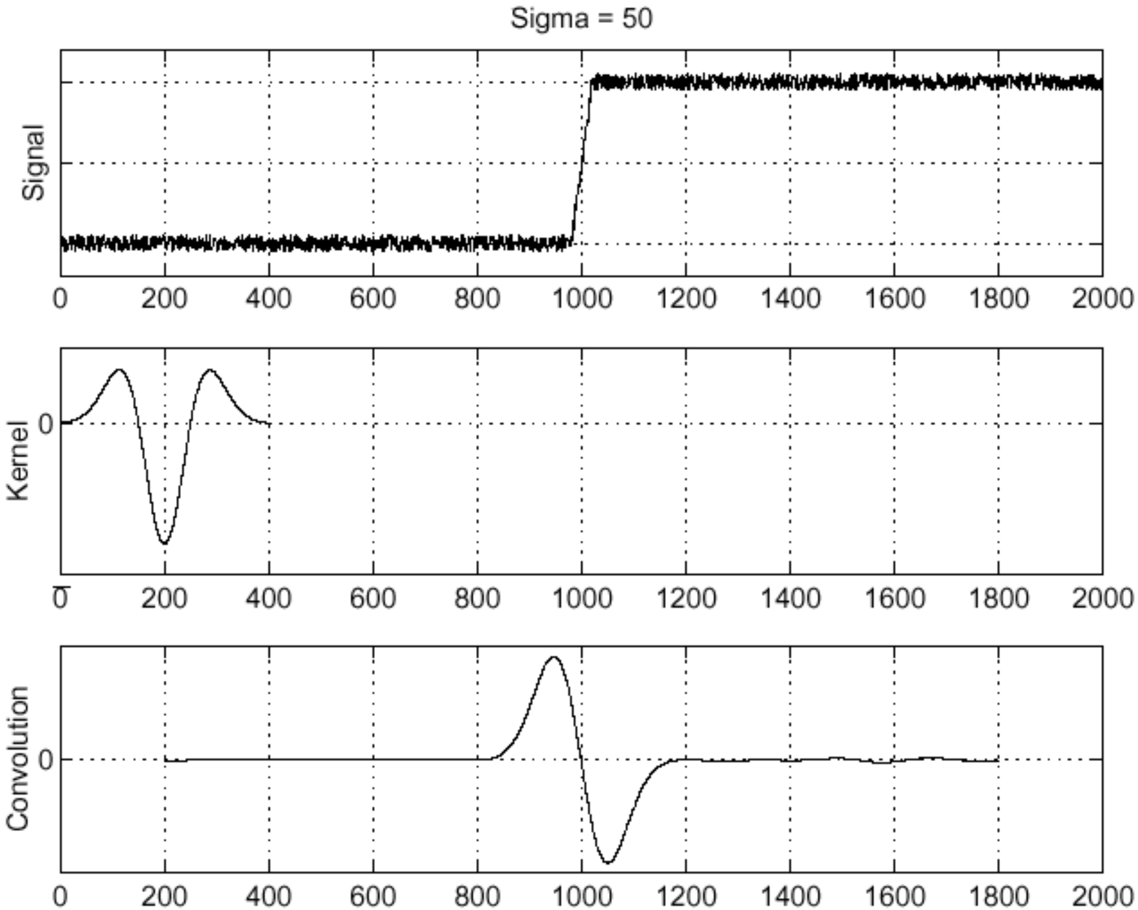
Edge

Derivative of Gaussian

Edge = maximum of derivative

Source: S. Seitz

Edge detection, Take 2



f — Edge

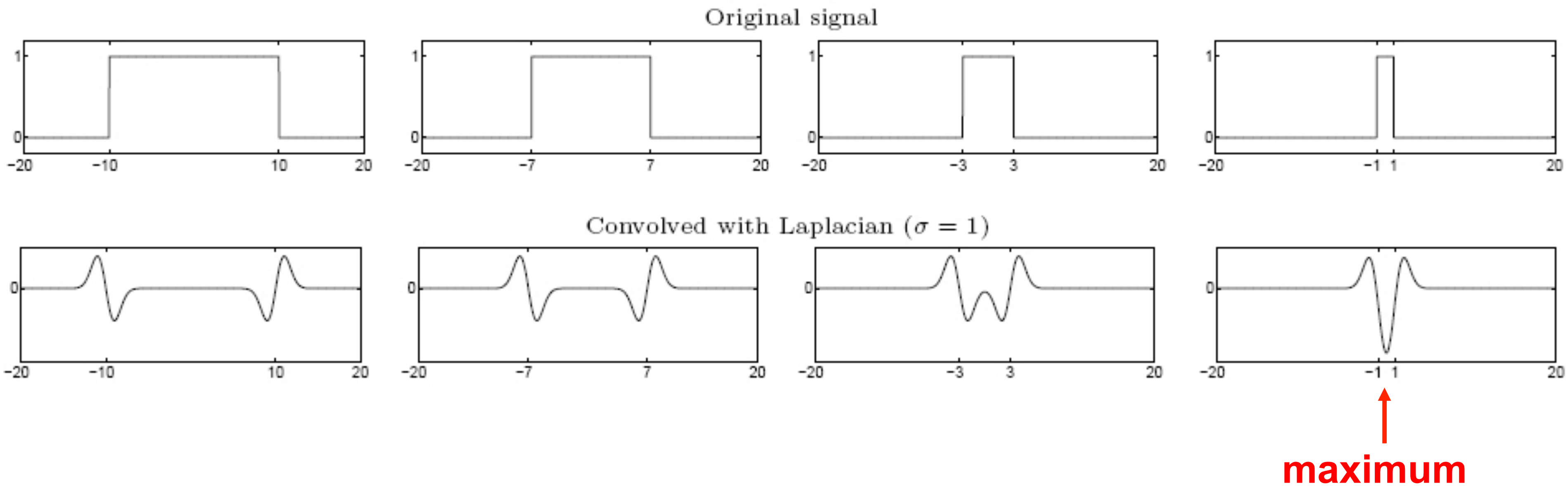
$\frac{d^2}{dx^2} g$ — Second derivative of Gaussian (Laplacian)

$f * \frac{d^2}{dx^2} g$ — Edge = zero crossing of second derivative

Source: S. Seitz

From edges to blobs

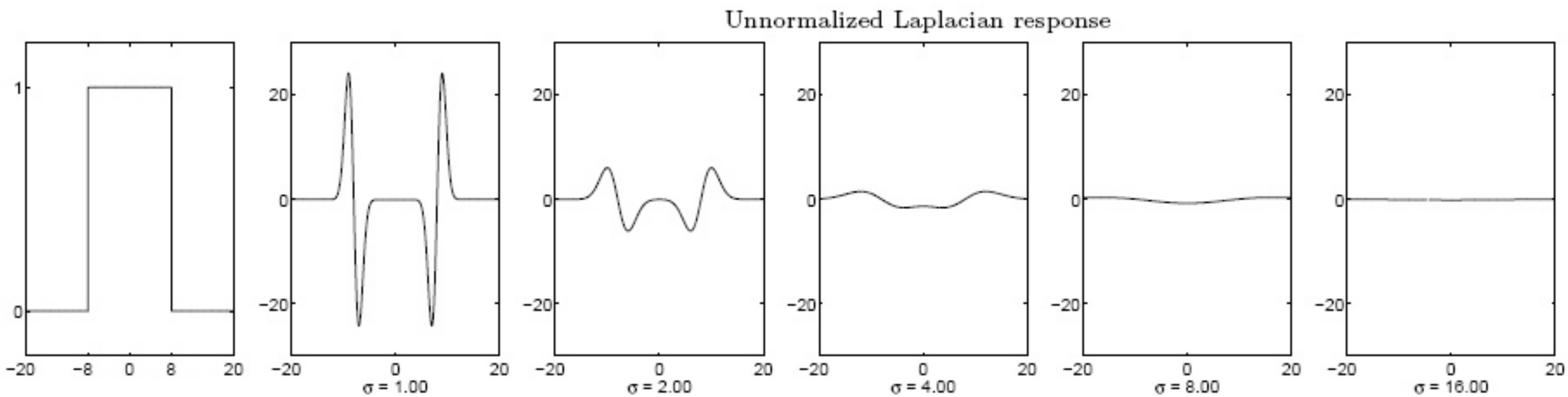
- Edge = ripple
- Blob = superposition of two ripples



Spatial selection: the magnitude of the Laplacian response will achieve a maximum at the center of the blob, provided the scale of the Laplacian is "matched" to the scale of the blob

Scale selection

- We want to find the characteristic scale of the blob by convolving it with Laplacians at several scales and looking for the maximum response
- However, Laplacian response decays as scale increases:



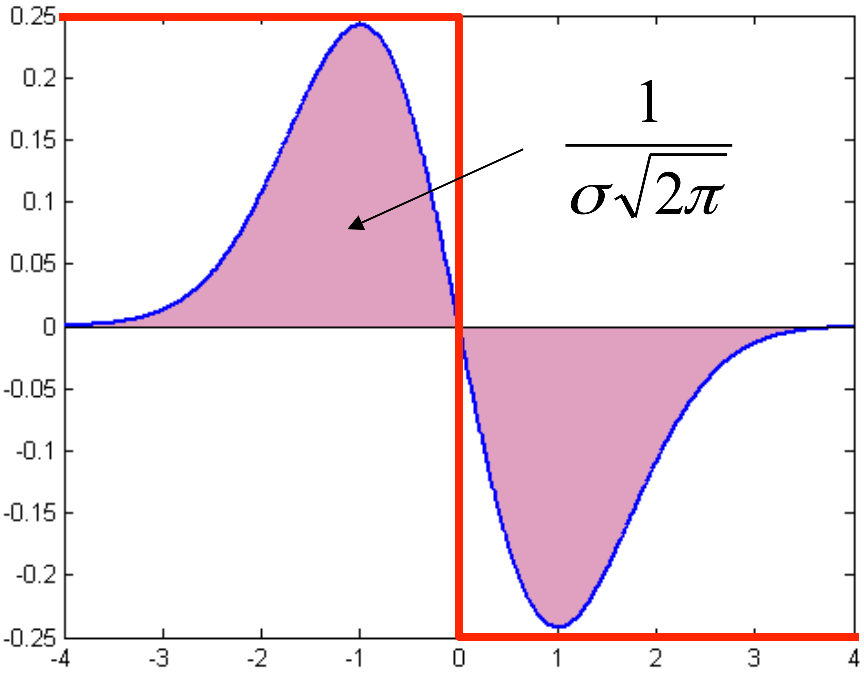
original signal (radius=8)

increasing σ

Why does this happen?

Scale normalization

- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases



Scale normalization

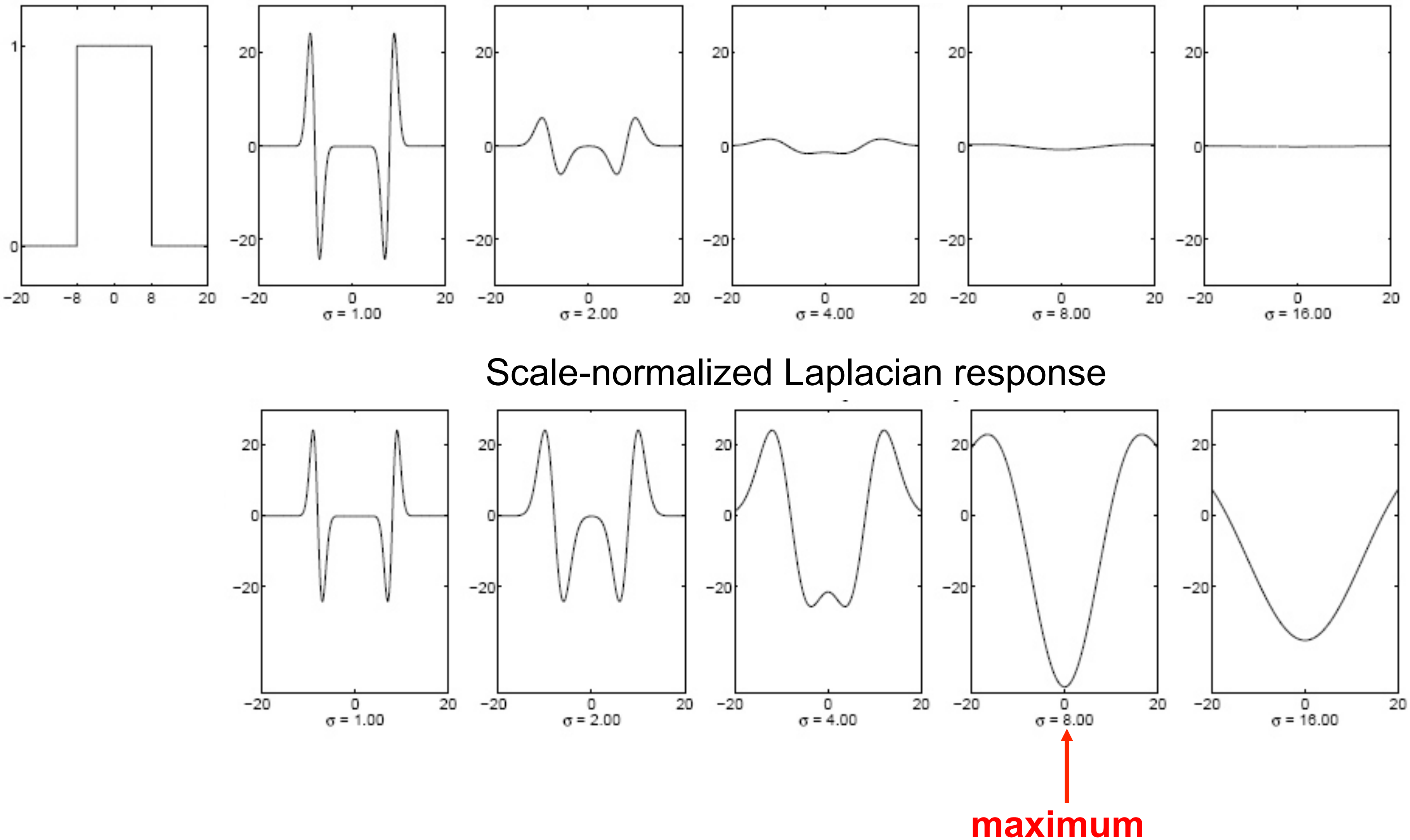
- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases
- To keep response the same (scale-invariant), must multiply Gaussian derivative by σ
- Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2

Effect of scale normalization

Original signal

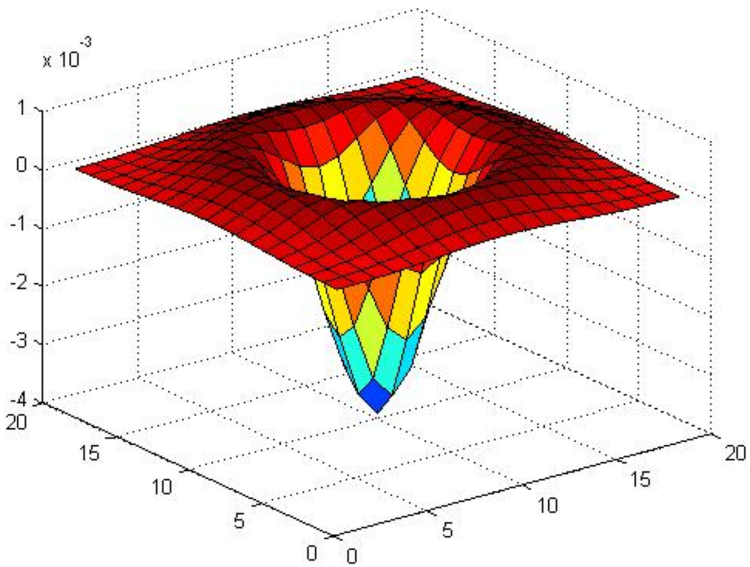
Unnormalized Laplacian response

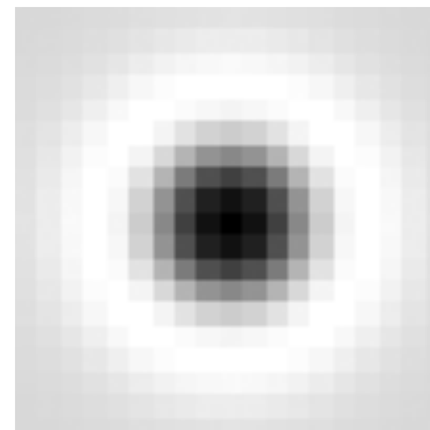
Scale-normalized Laplacian response



Blob detection in 2D

Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D

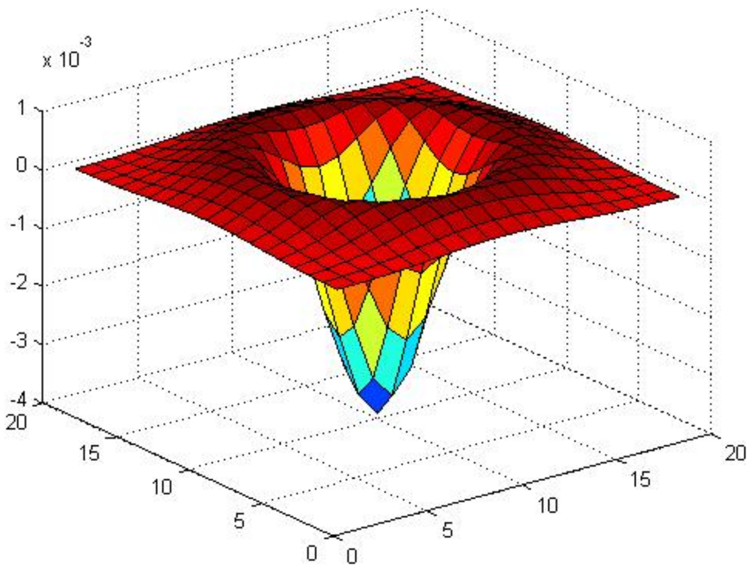


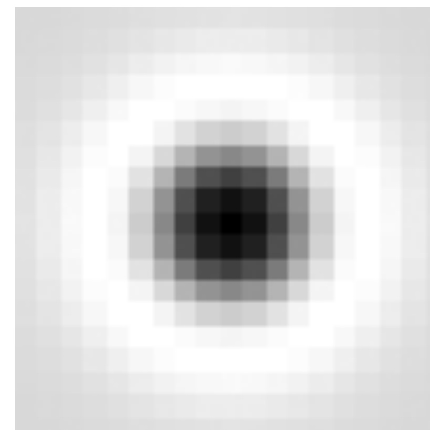


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} = \frac{1}{\pi\sigma^4}\left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Blob detection in 2D

Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D

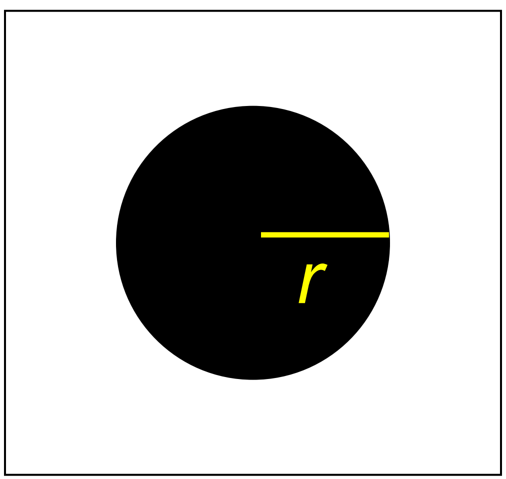




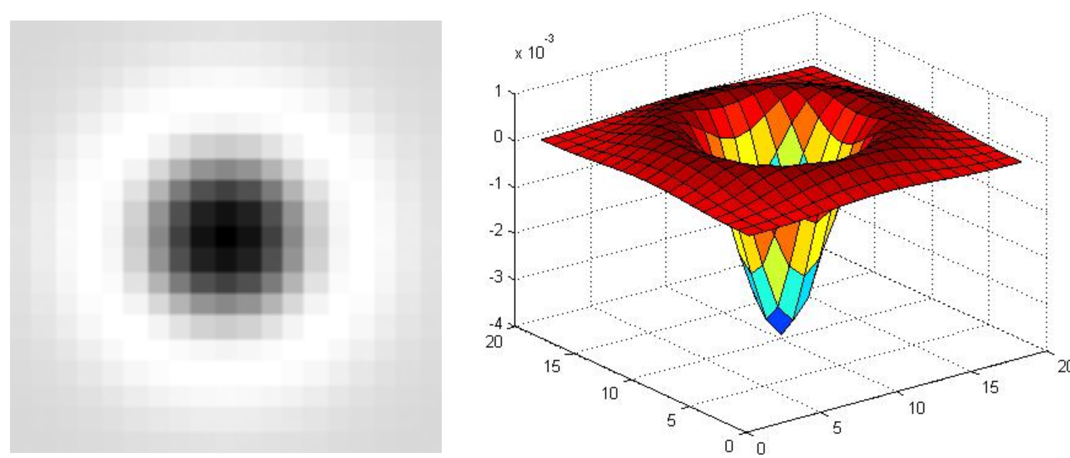
Scale-normalized: $$\nabla^2_{\text{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

Scale selection

- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r?



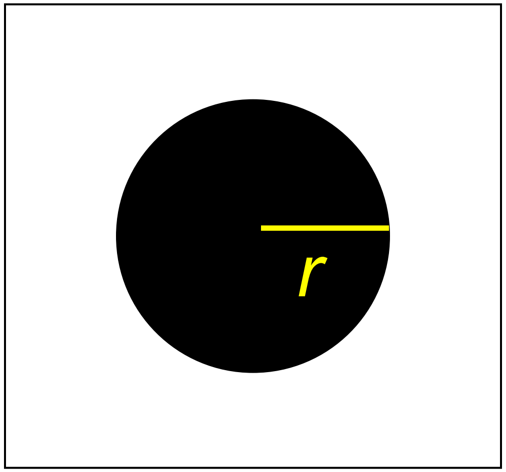
image



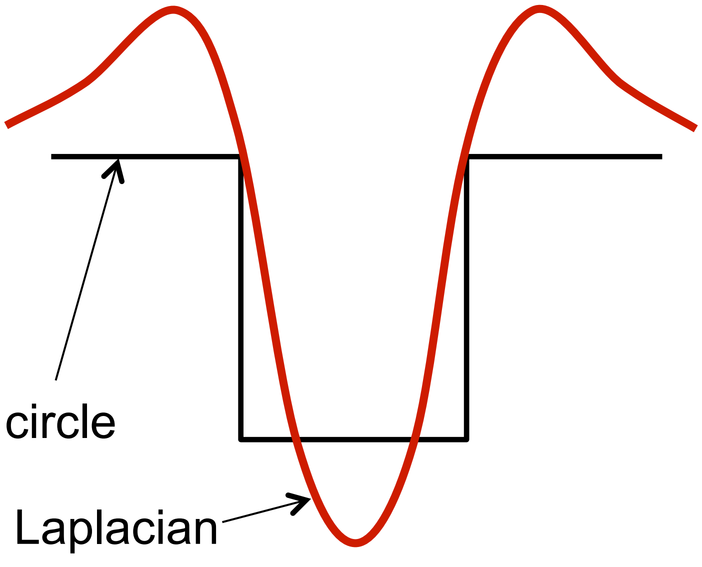
Laplacian

Scale selection

- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r?
- To get maximum response, the zeros of the Laplacian have to be aligned with the circle
- Zeros of Laplacian is given by (up to scale): $\left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) = 0$
- Therefore, the maximum response occurs at $\sigma = r / \sqrt{2}.$

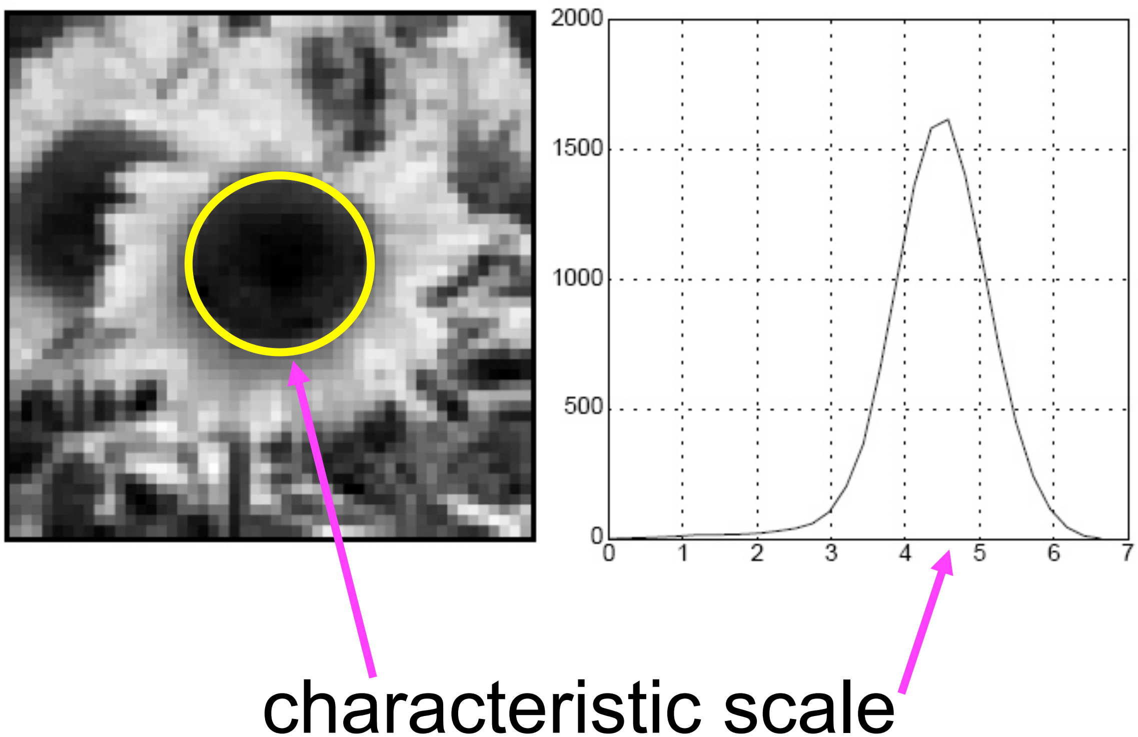


image



Characteristic scale

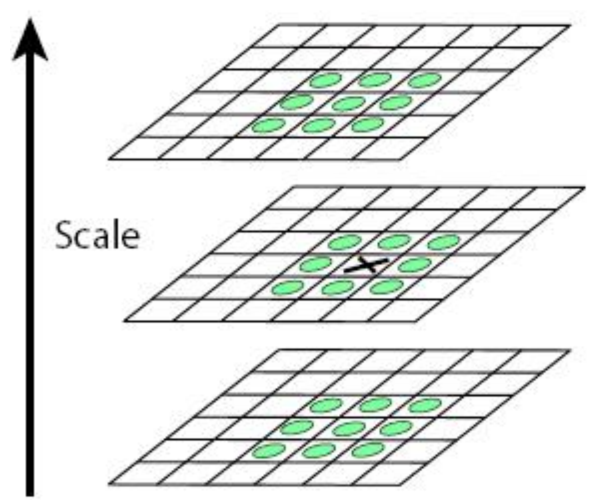
- We define the characteristic scale of a blob as the scale that produces peak of Laplacian response in the blob center



T. Lindeberg (1998). ["Feature detection with automatic scale selection."]() *International Journal of Computer Vision* **30** (2): pp 77--116.

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space

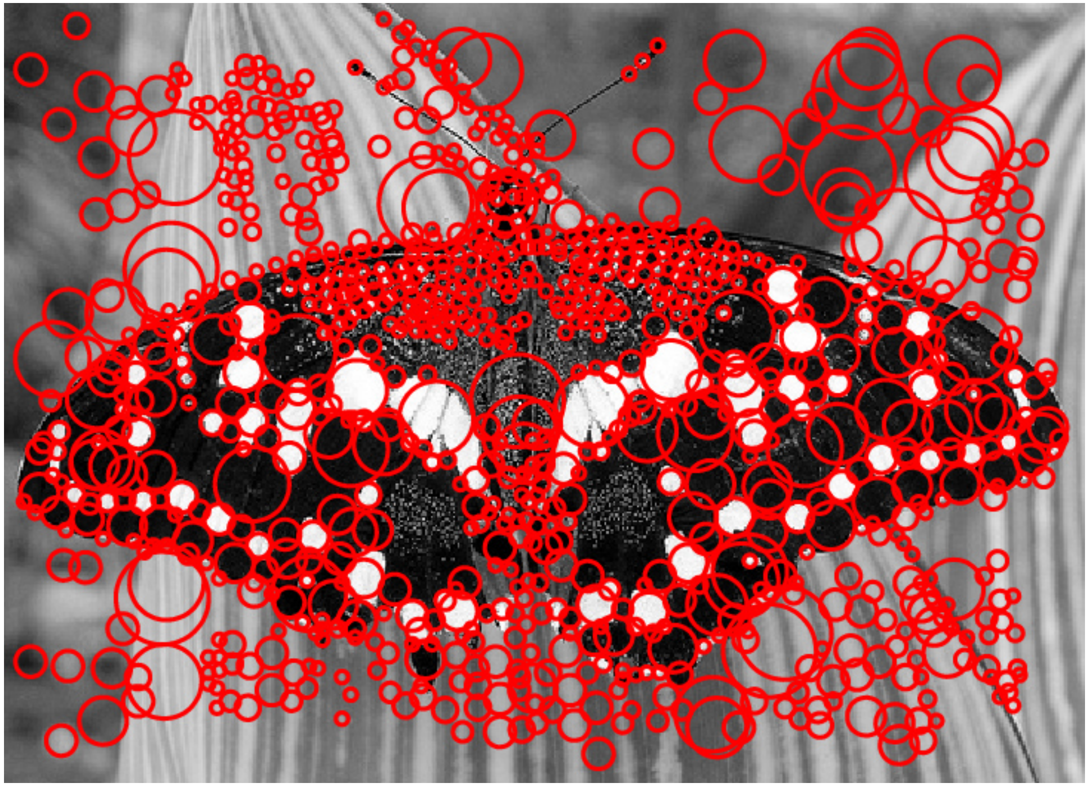


Scale-space blob detector: Example

Scale-space blob detector: Example

sigma = 11.9912

Scale-space blob detector: Example



Matching with Features

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

Scale Invariant Feature Transform

David Lowe IJCV 2004

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

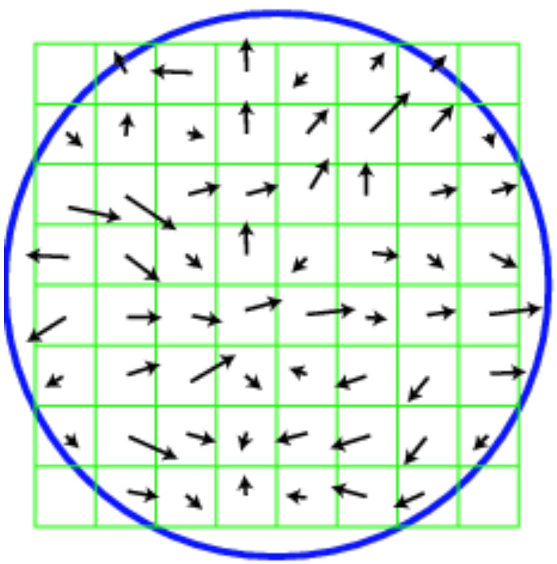
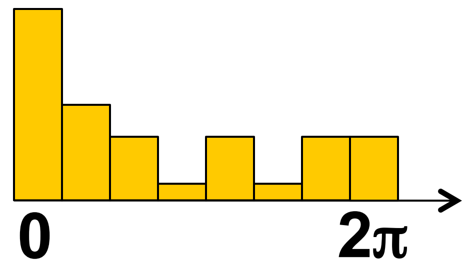
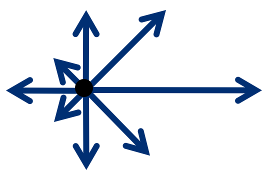


Image gradients



angle histogram

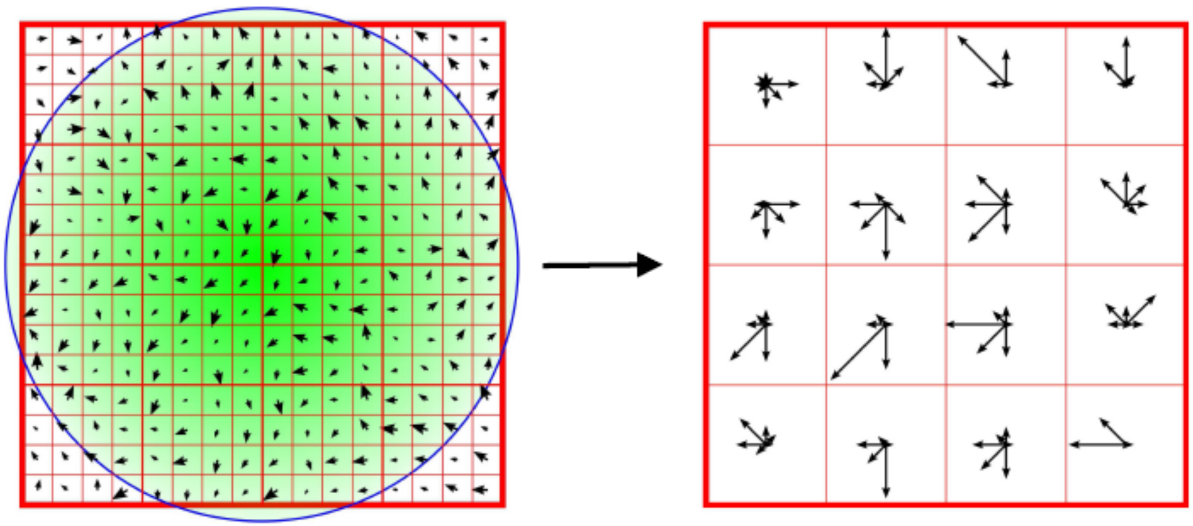


Adapted from slide by David Lowe

Former NYU faculty & Prof. Ken Perlin's advisor

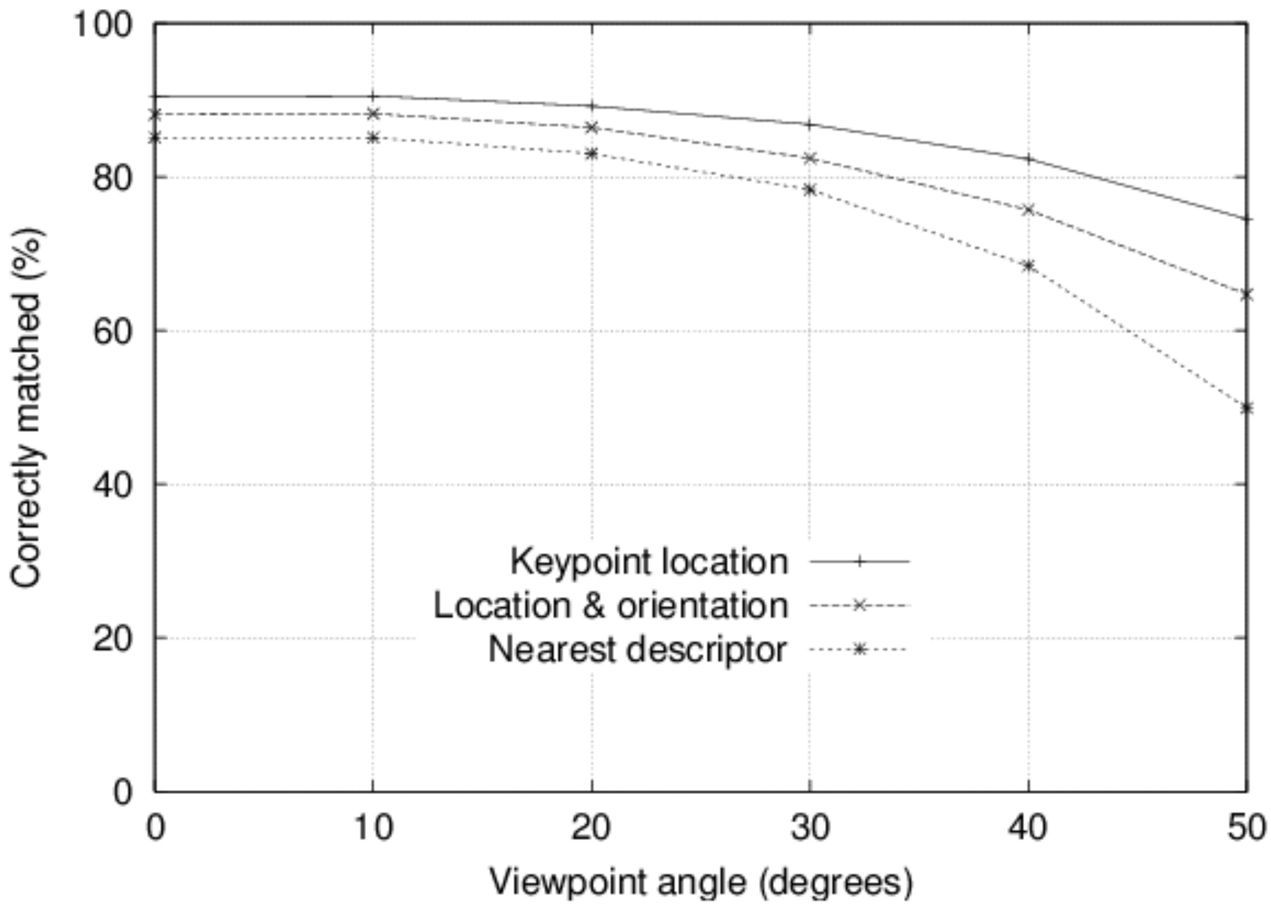
Orientation Histogram

- 4x4 spatial bins (16 bins total)
- Gaussian center-weighting
- 8-bin orientation histogram per bin
- 8 x 16 = 128 dimensions total
- Normalized to unit norm



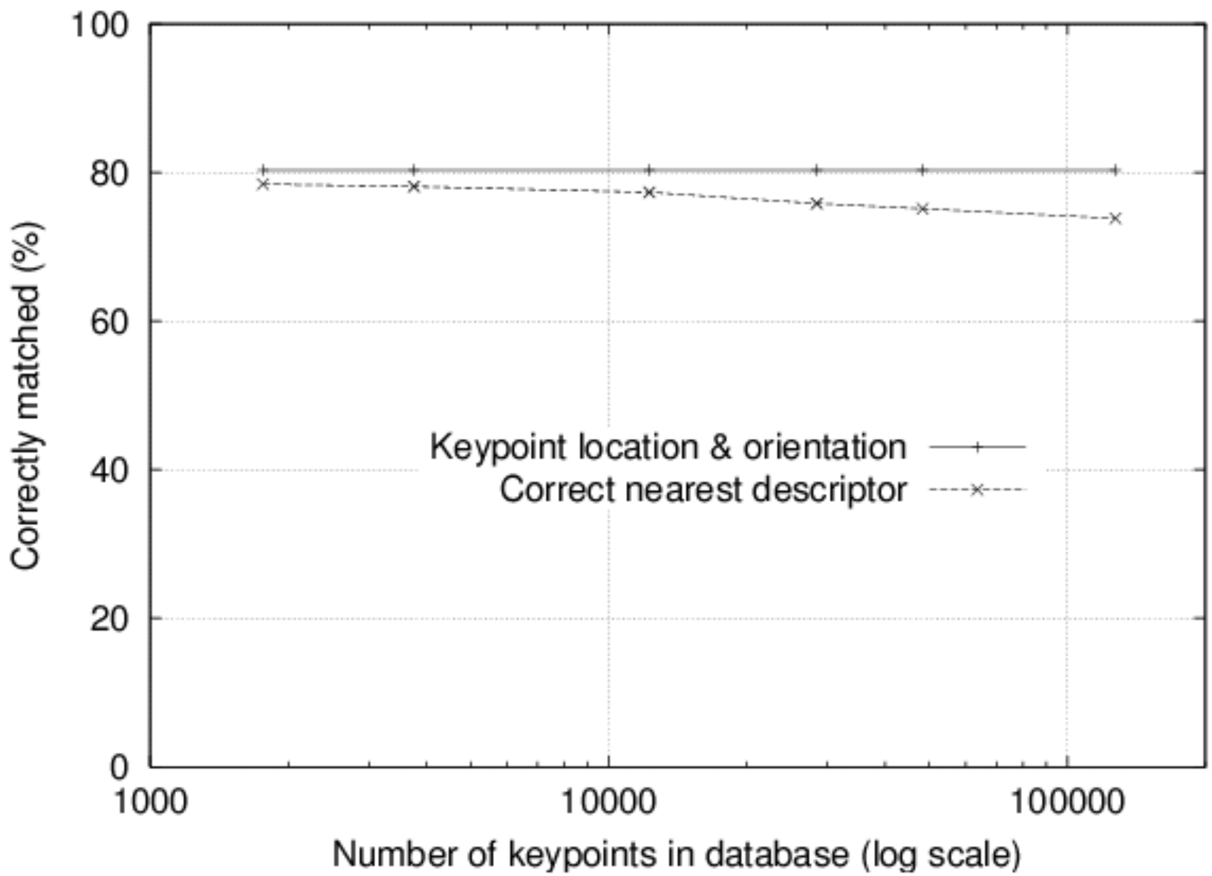
Feature stability to affine change

- Match features after random change in image scale & orientation, with 2% image noise, and affine distortion
- Find nearest neighbor in database of 30,000 features



Distinctiveness of features

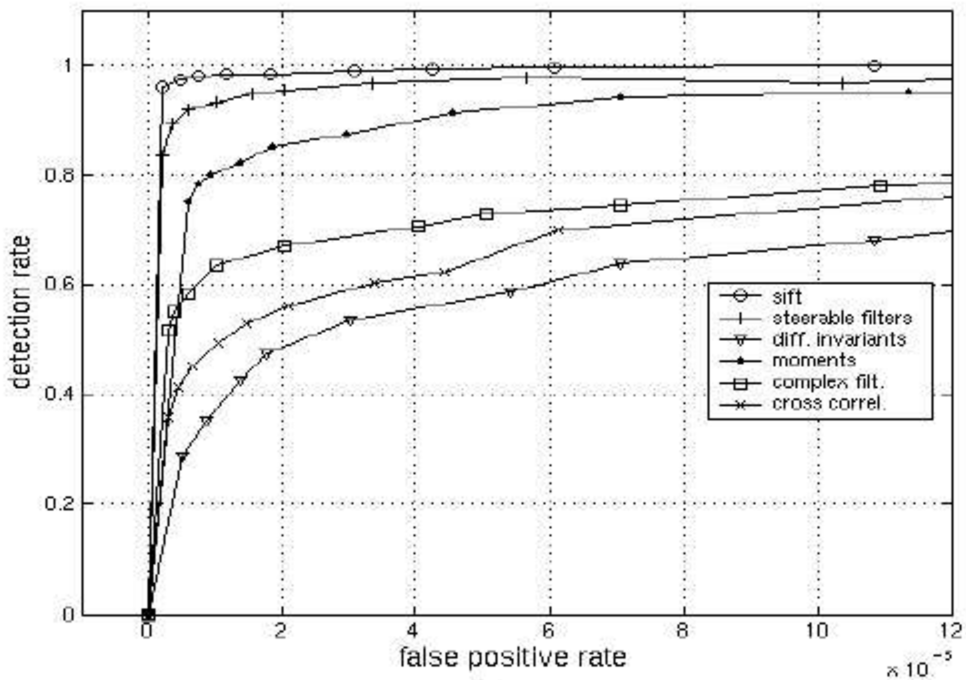
- Vary size of database of features, with 30 degree affine change, 2% image noise
- Measure % correct for single nearest neighbor match



SIFT – Scale Invariant Feature Transform[1]

- Empirically found[2] to show very good performance, invariant to *image rotation*, *scale*, *intensity change*, and to moderate *affine* transformations

Scale = 2.5
Rotation = 45^0



[1] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Accepted to IJCV 2004
[2] K.Mikolajczyk, C.Schmid. "A Performance Evaluation of Local Descriptors". CVPR 2003

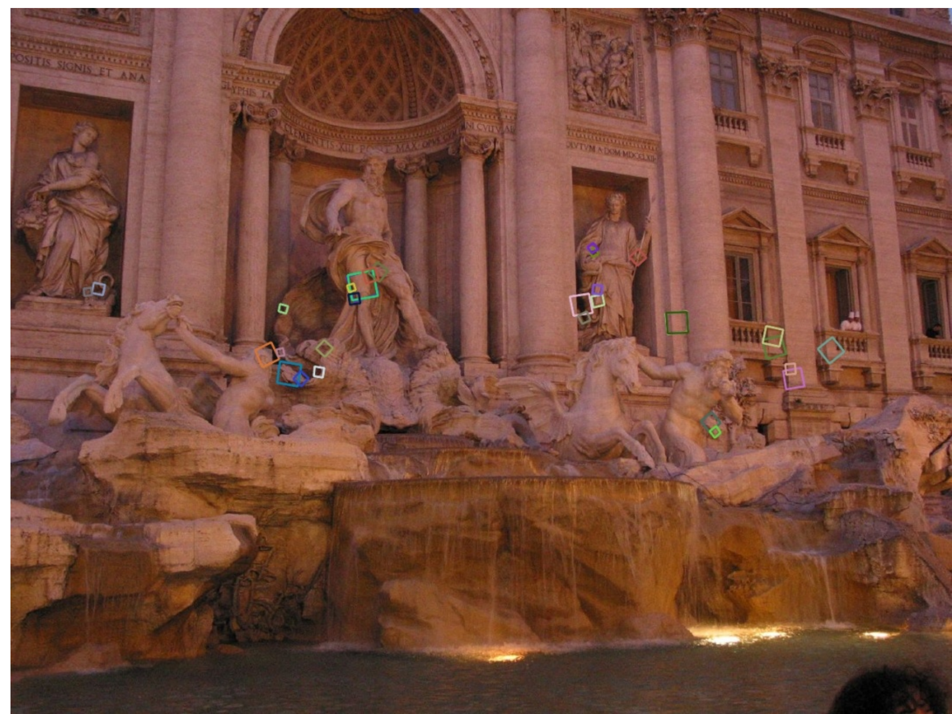
SIFT invariances

- Spatial binning gives tolerance to small shifts in location and scale
- Explicit orientation normalization
- Photometric normalization by making all vectors unit norm
- Orientation histogram gives robustness to small local deformations

Summary of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Matching with Features

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

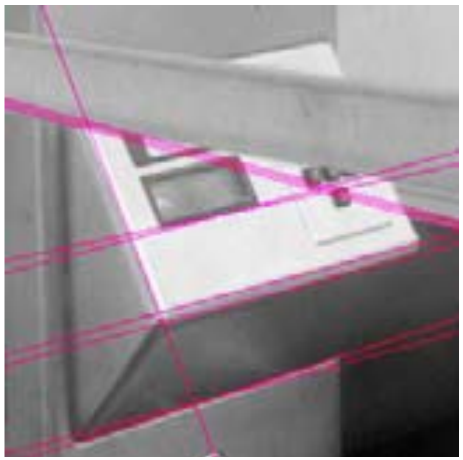
Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

Fitting

- Choose a parametric model to represent a set of features



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

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

Slide: S. Lazebnik

Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

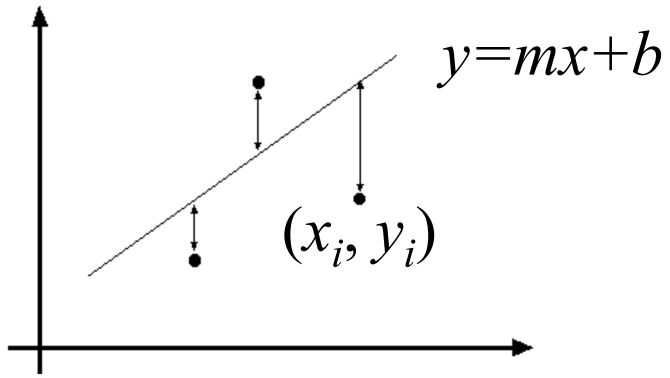
Least squares line fitting

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

Line equation: $y_i = m x_i + b$

Find (m, b) to minimize

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



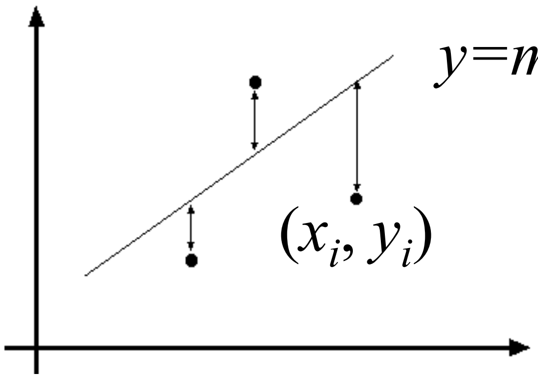
Slide: S. Lazebnik

Least squares line fitting

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

Line equation: $y_i = m x_i + b$

Find (m, b) to minimize



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

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

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

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

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

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

Slide: S. Lazebnik

Matlab Demo

```
%%%% let's make some points
n = 10;
true_grad = 2;
true_intercept = 3;
noise_level = 0.04;

x = rand(1,n);
y = true_grad*x + true_intercept + randn(1,n)*noise_level;

figure; plot(x,y,'rx');
hold on;

%%% make matrix for linear system
X = [x(:) ones(n,1)];

%%% Solve system of equations
p = inv(X'*X)*X'*y(:); % Pseudo-inverse
p = pinv(X) * y(:); % Pseduo-inverse
p = X \ y(:); % Matlab's \ operator

est_grad = p(1);
est_intercept = p(2);

plot(x,est_grad*x+est_intercept,'b-');

fprintf('True gradient: %f, estimated gradient: %f\n',true_grad,est_grad);
fprintf('True intercept: %f, estimated intercept: %f\n',true_intercept,est_intercept);
```

Problem with “vertical” least squares

- Not rotation-invariant
- Fails completely for vertical lines

Slide: S. Lazebnik

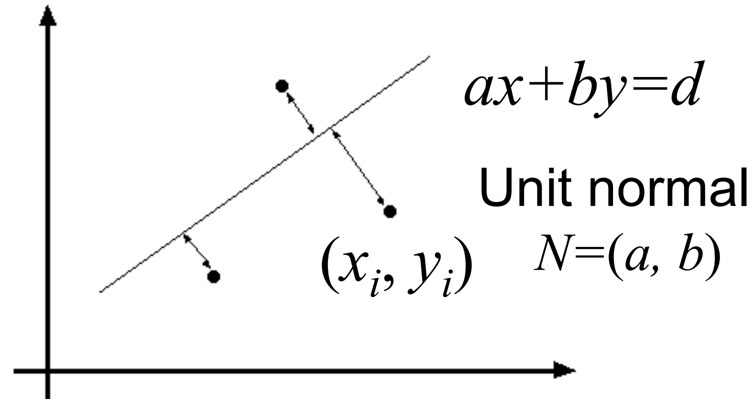
Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

Total least squares

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



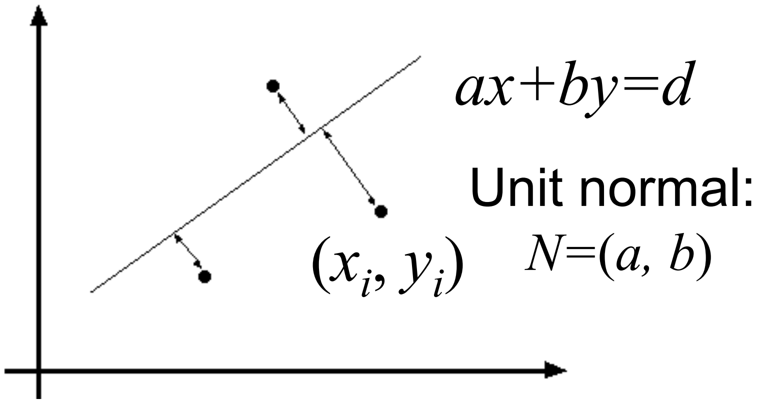
Slide: S. Lazebnik

Total least squares

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

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

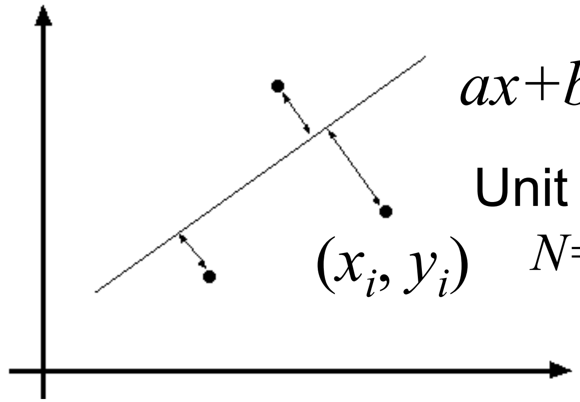
$$E = \sum_{i=1}^{n} (ax_i + by_i - d)^2$$



Total least squares

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

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



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

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

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

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

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

Slide: S. Lazebnik

Total least squares

$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \qquad U^T U = \begin{bmatrix} \sum_{i=1}^{n} (x_i - \bar{x})^2 & \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n} (y_i - \bar{y})^2 \end{bmatrix}$$

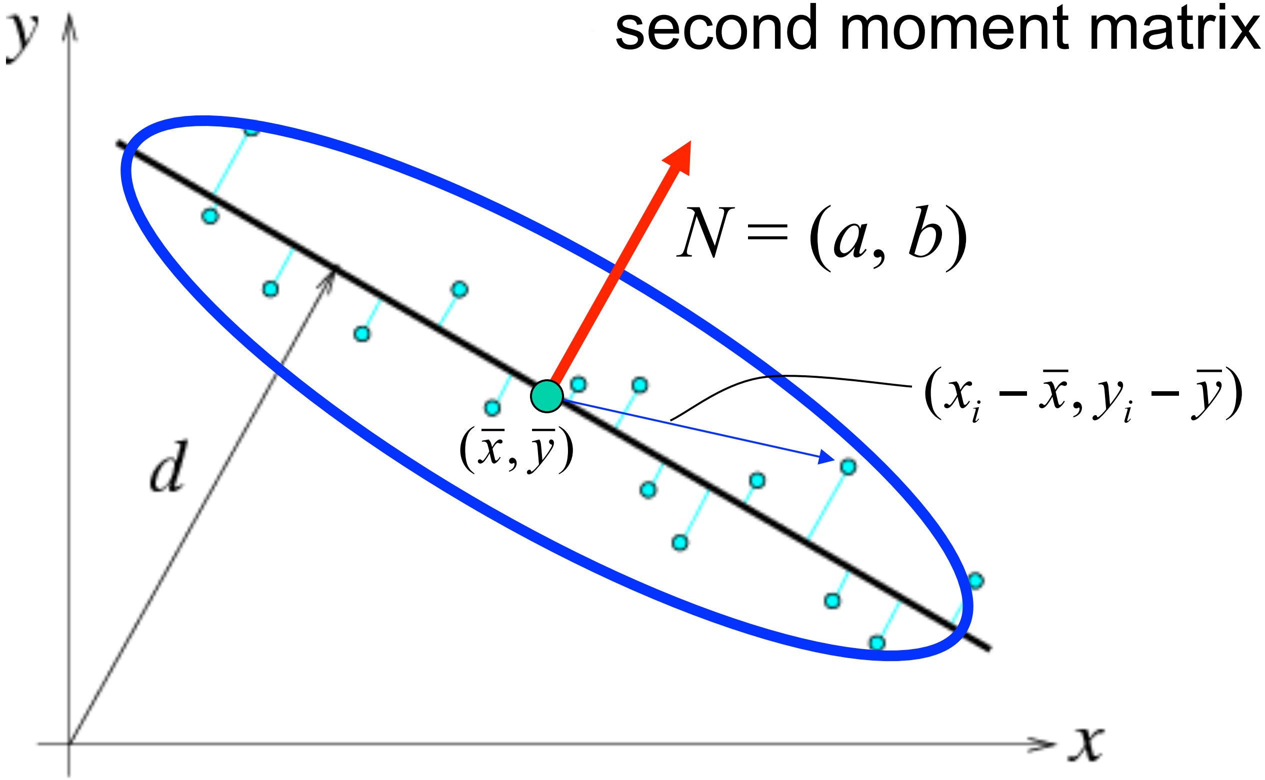
second moment matrix

Slide: S. Lazebnik

Total least squares

$$U = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix} \qquad U^T U = \begin{bmatrix} \sum_{i=1}^{n} (x_i - \bar{x})^2 & \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \\ \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) & \sum_{i=1}^{n} (y_i - \bar{y})^2 \end{bmatrix}$$

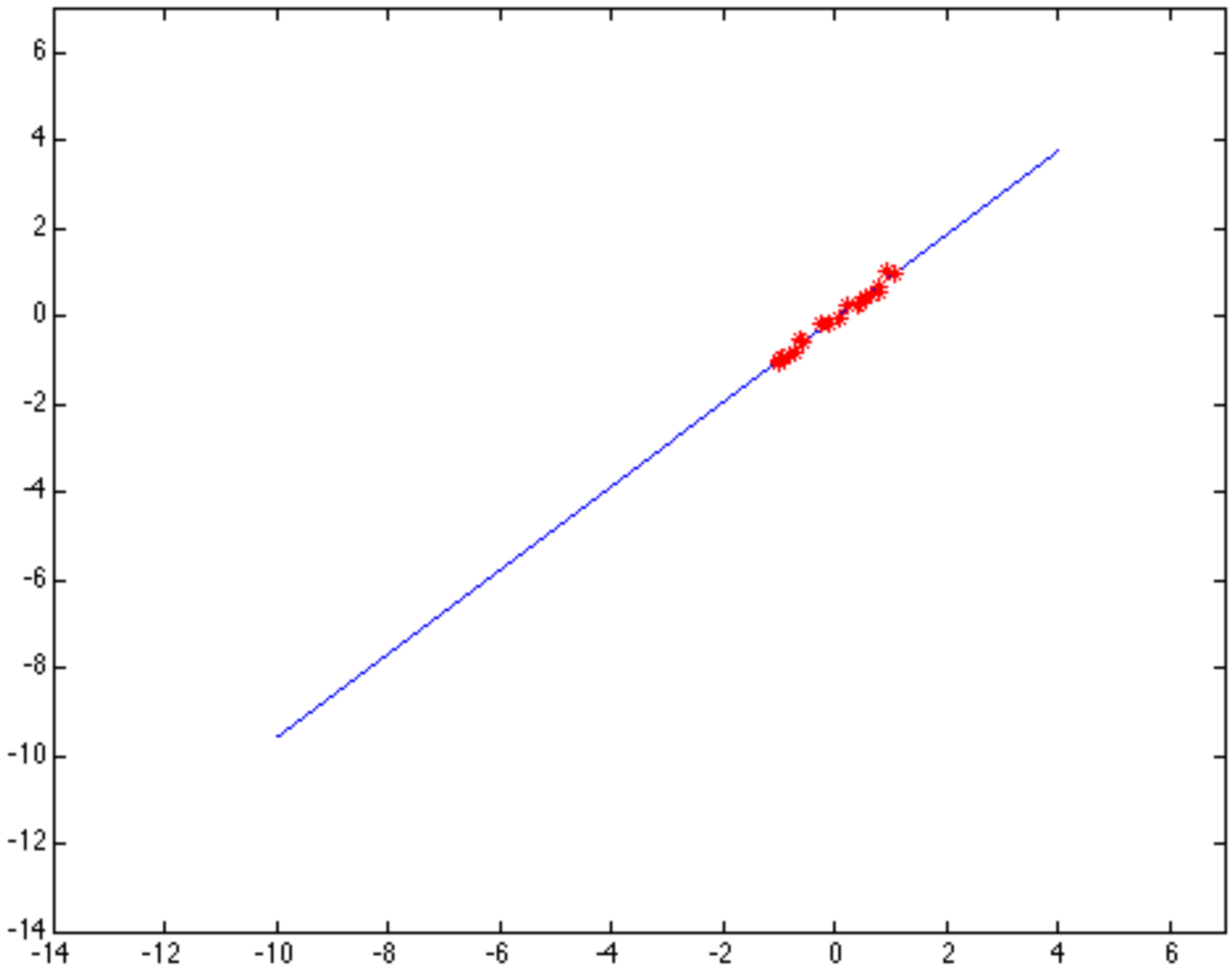
second moment matrix



Slide: S. Lazebnik

Least squares: Robustness to noise

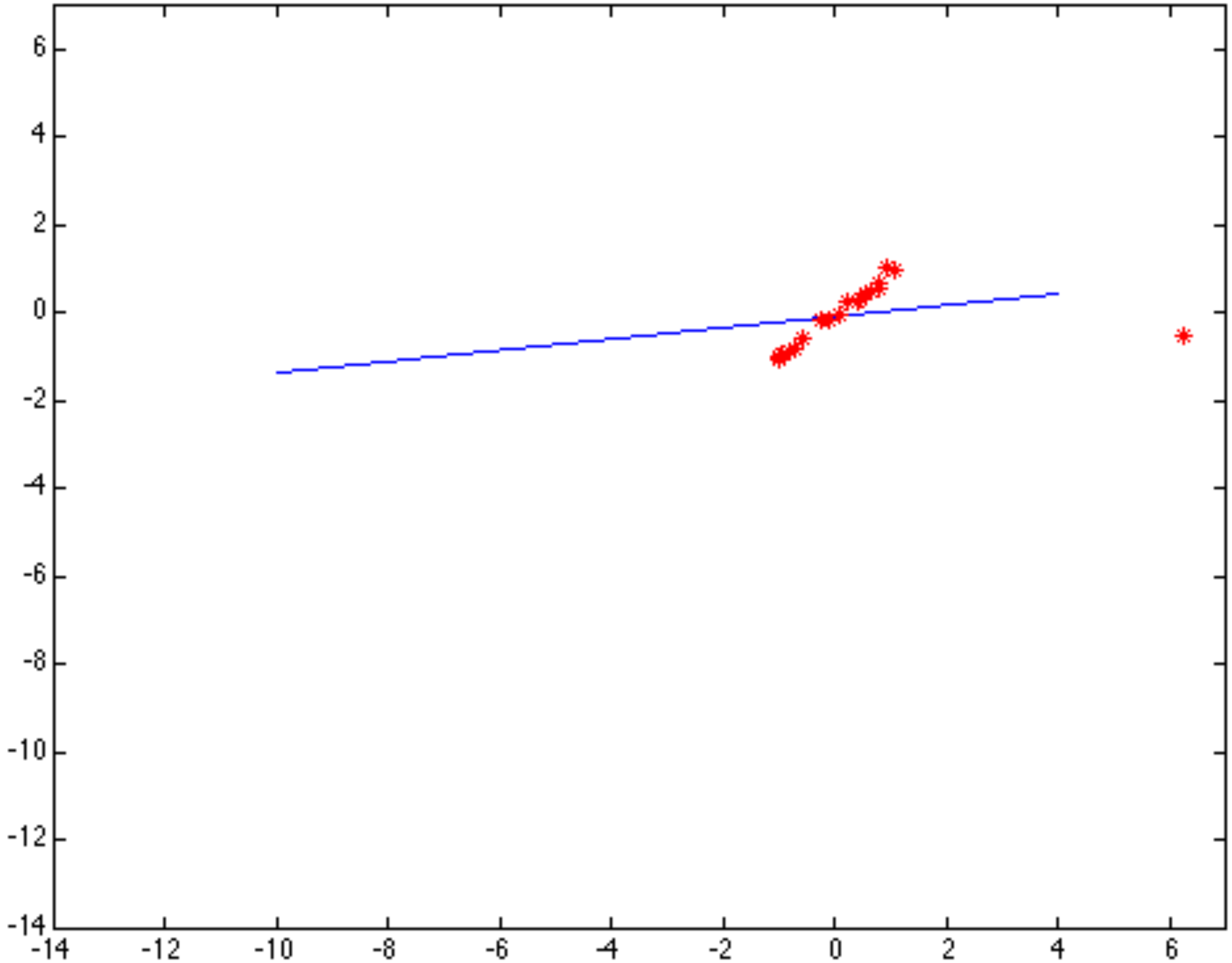
Least squares fit to the red points:



Slide: S. Lazebnik

Least squares: Robustness to noise

Least squares fit with an outlier:



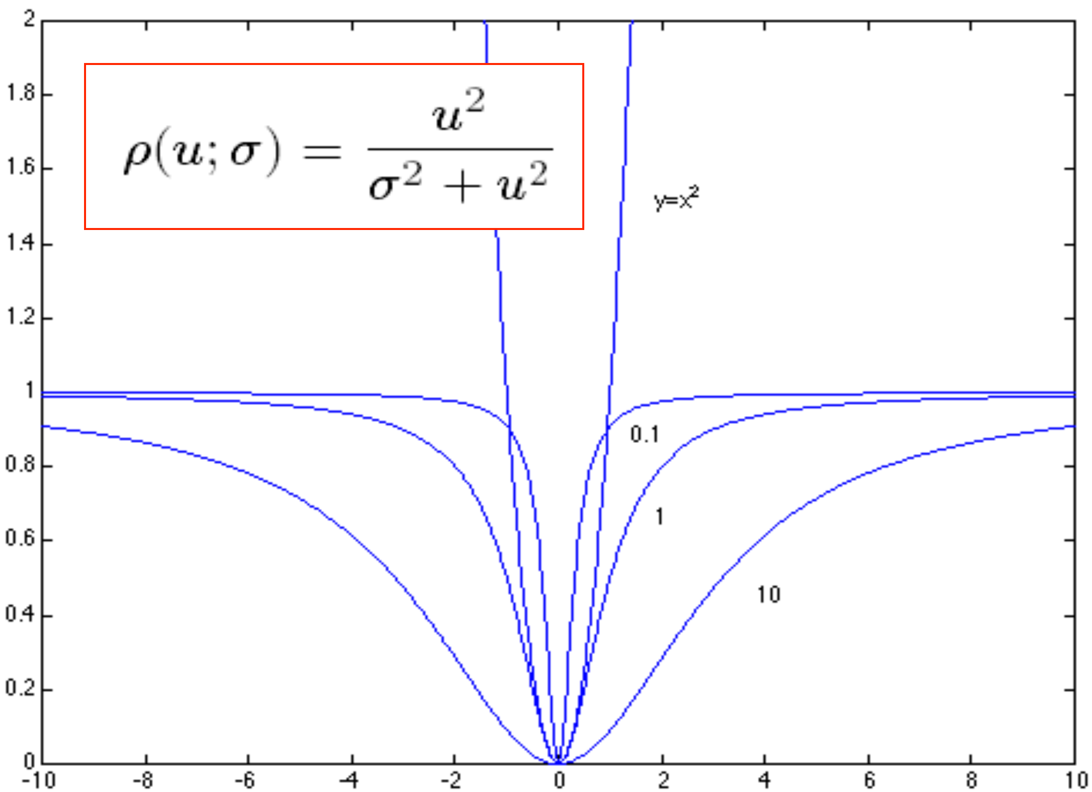
Problem: squared error heavily penalizes outliers

Slide: S. Lazebnik

Robust estimators

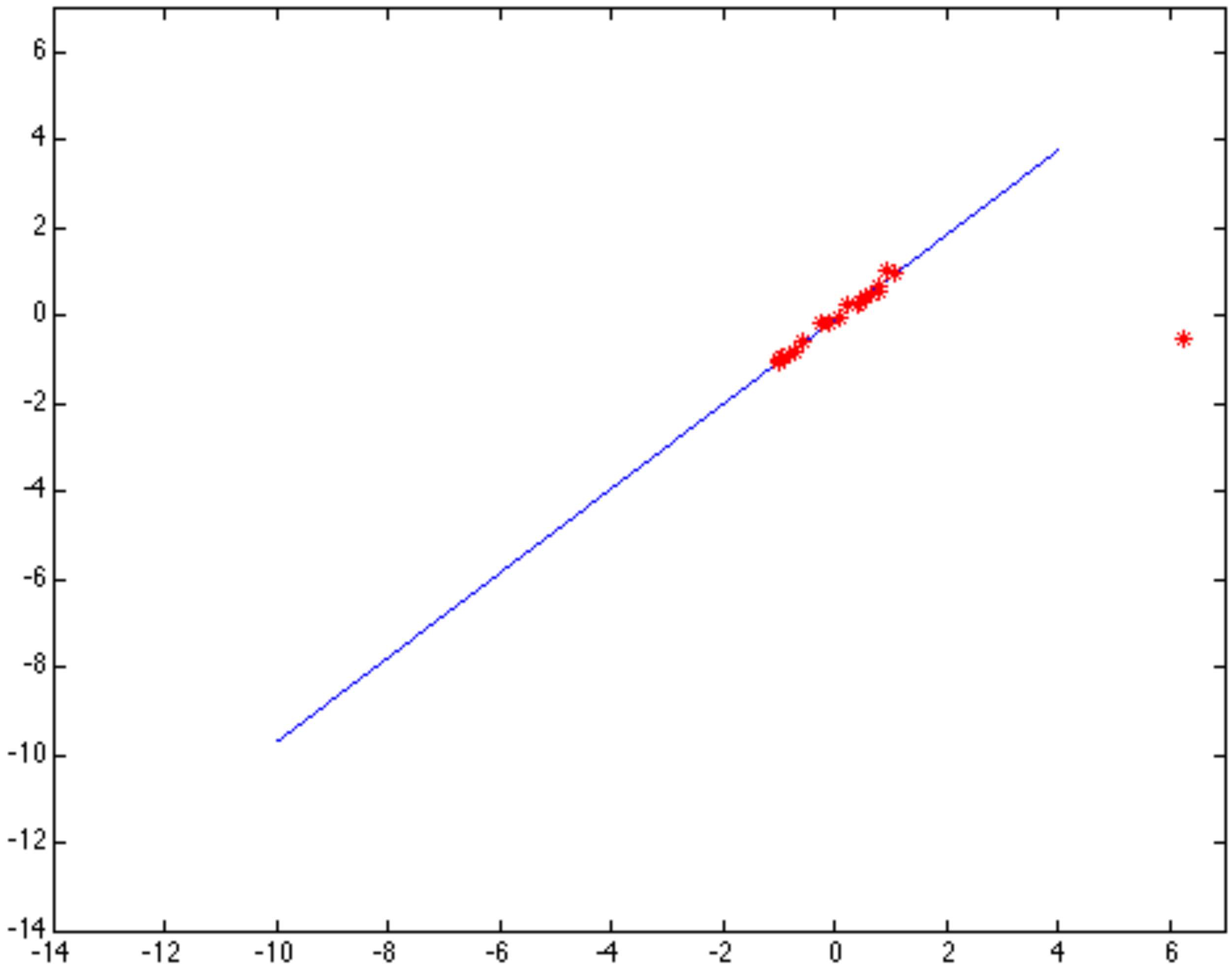
- General approach: minimize $\sum_i \rho\left(r_i\left(x_i,\theta\right);\sigma\right)$

$r_i(x_i, \theta)$ – 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*

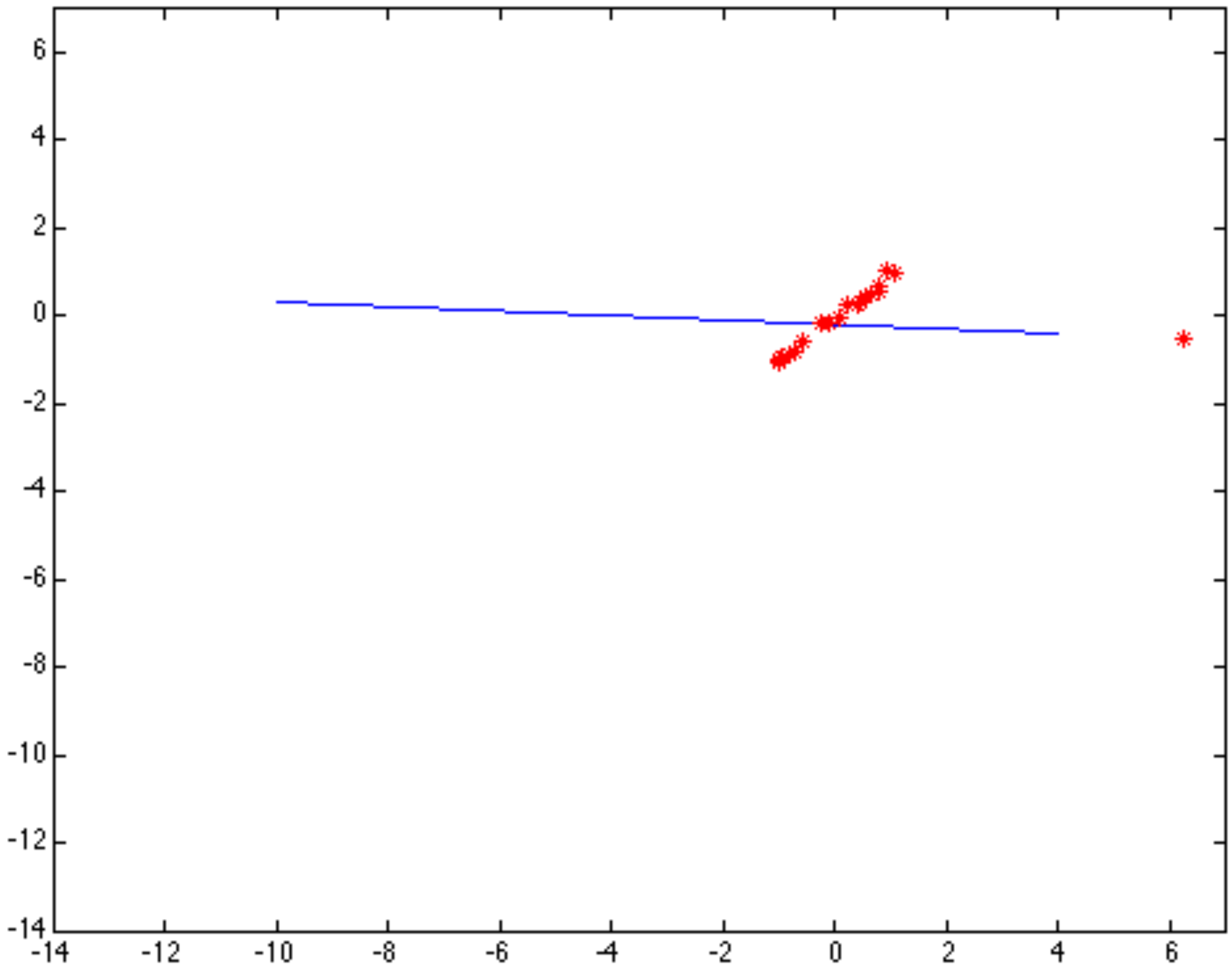
Choosing the scale: Just right



The effect of the outlier is minimized

Slide: S. Lazebnik

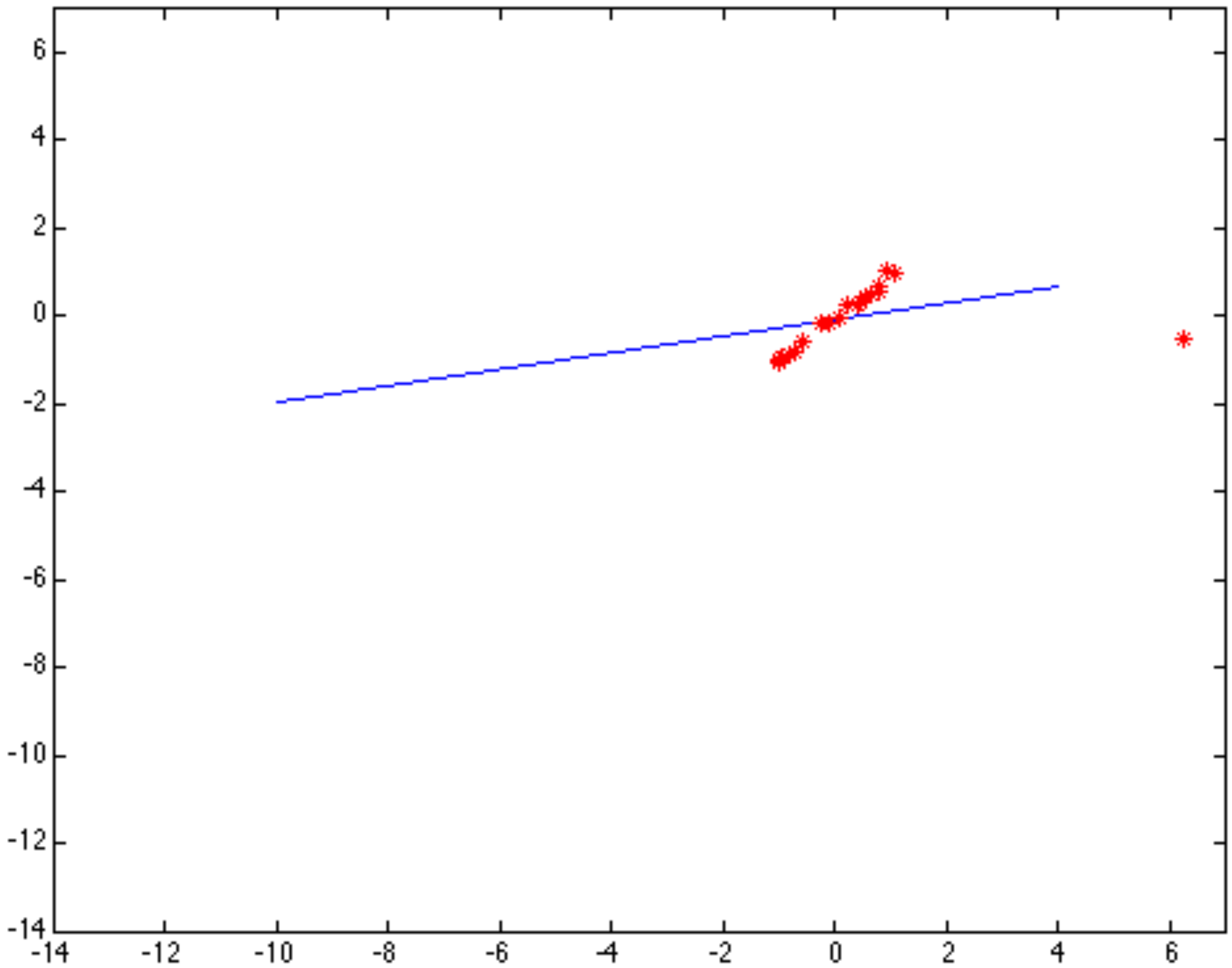
Choosing the scale: Too small



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

Slide: S. Lazebnik

Choosing the scale: Too large



Behaves much the same as least squares

Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

RANSAC

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

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

Slide: S. Lazebnik

RANSAC for line fitting

Repeat ***N*** times:

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

Source: M. Pollefeys

Choosing the parameters

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

Source: M. Pollefeys

Choosing the parameters

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

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

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

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

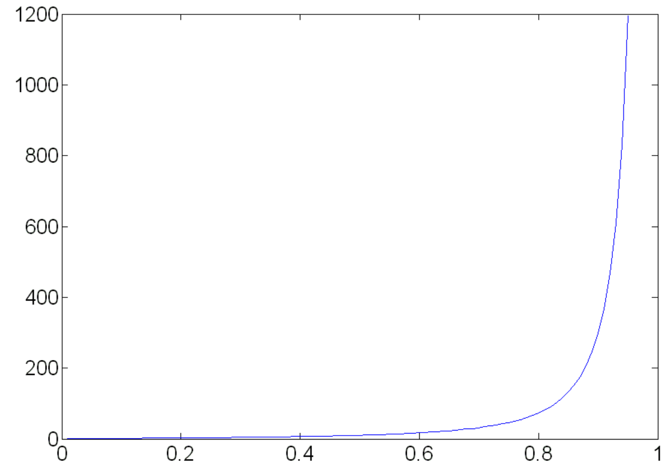
Source: M. Pollefeys

Choosing the parameters

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

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

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



Source: M. Pollefeys

Choosing the parameters

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

Source: M. Pollefeys

Adaptively determining the number of samples

- 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 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

Source: M. Pollefeys

Voting schemes

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

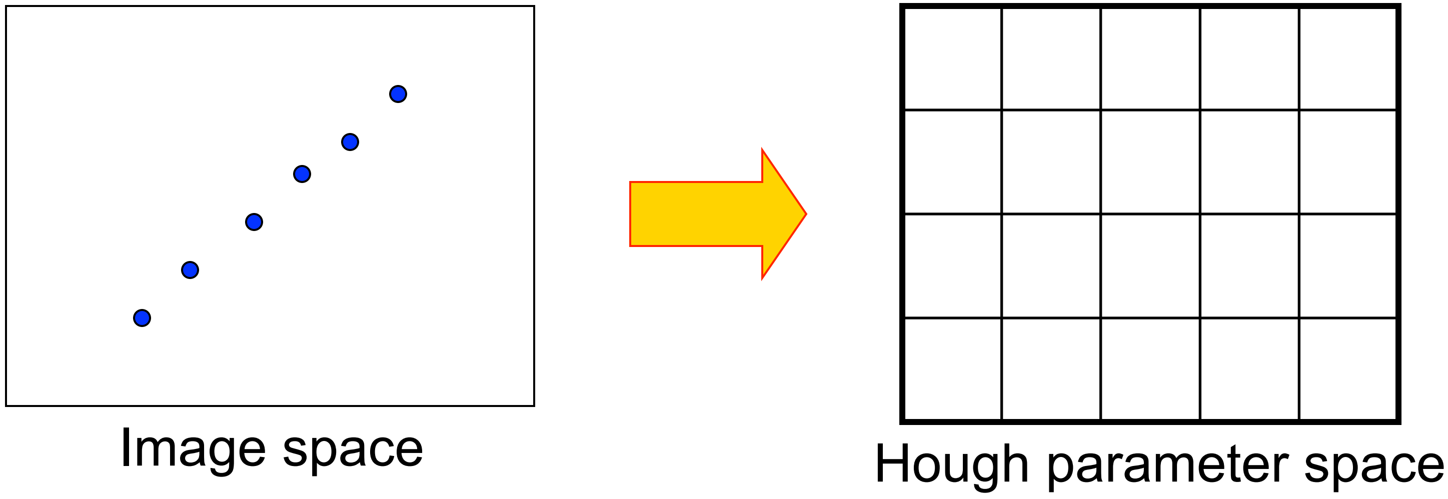
Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

Hough transform

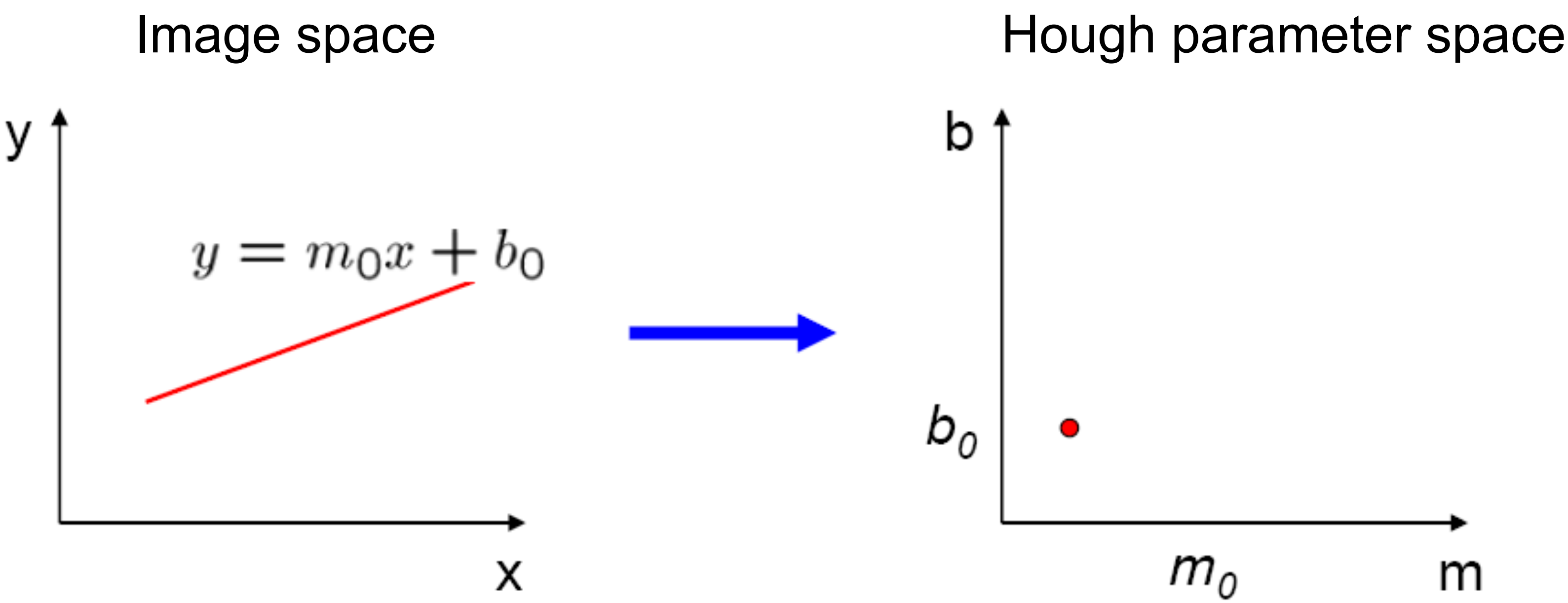
- An early type of voting scheme
- General outline:
  - Discretize parameter space into bins
  - For each feature point in the image, put a vote in every bin in the parameter space that could have generated this point
  - Find bins that have the most votes



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

Parameter space representation

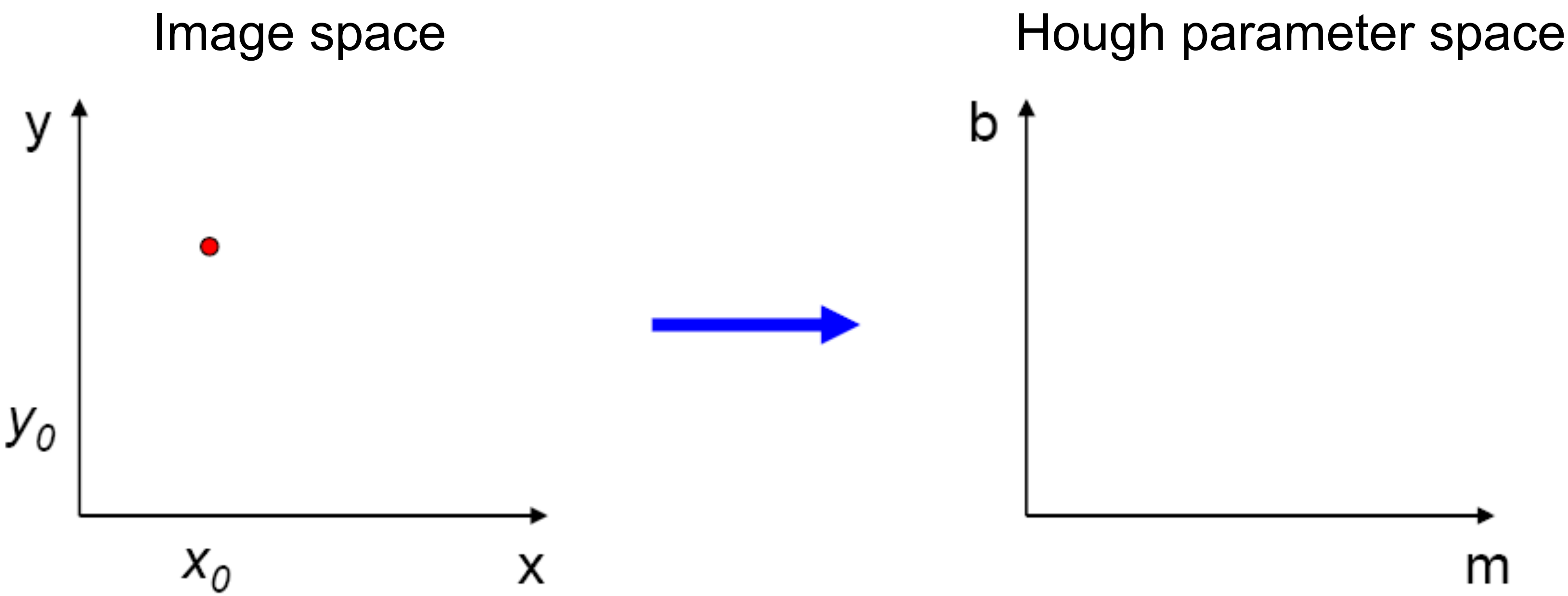
- A line in the image corresponds to a point in Hough space



Source: S. Seitz

Parameter space representation

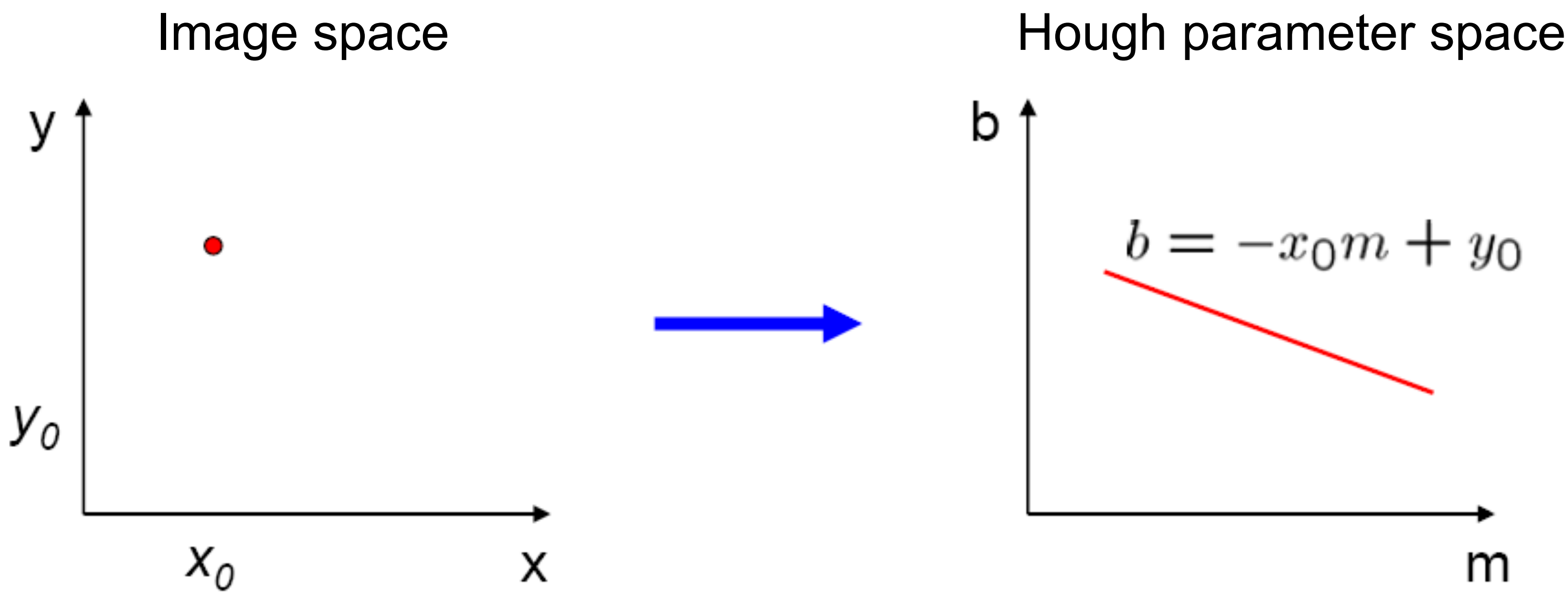
- What does a point (x_0, y_0) in the image space map to in the Hough space?



Source: S. Seitz

Parameter space representation

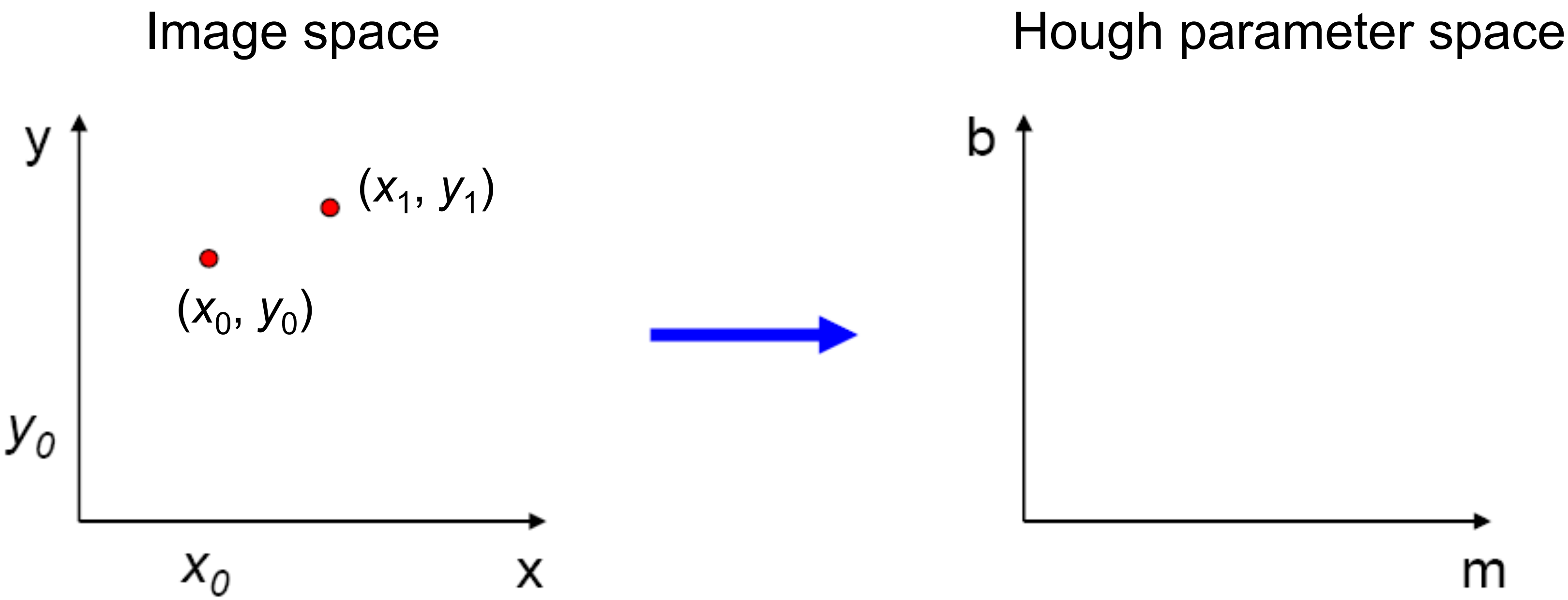
- What does a point (x_0, y_0) in the image space map to in the Hough space?
  - Answer: the solutions of $b = -x_0 m + y_0$
  - This is a line in Hough space



Source: S. Seitz

Parameter space representation

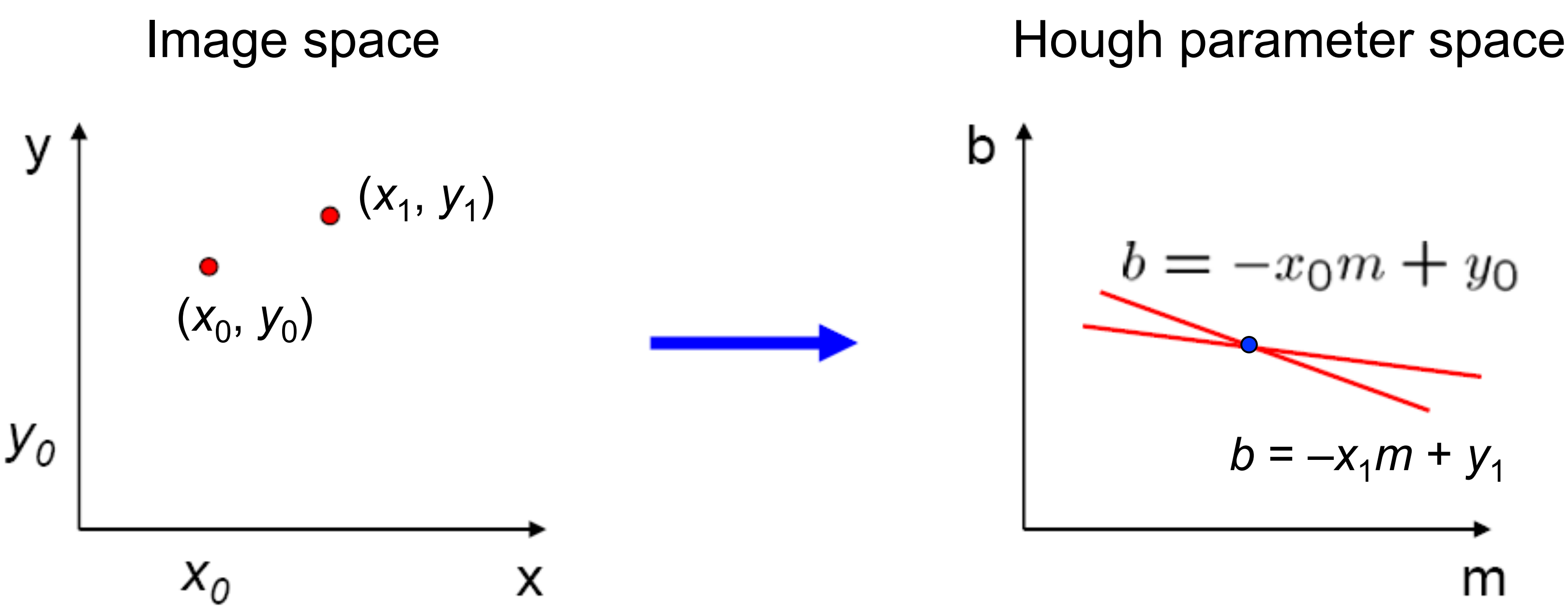
- Where is the line that contains both (x_0, y_0) and (x_1, y_1)?



Source: S. Seitz

Parameter space representation

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



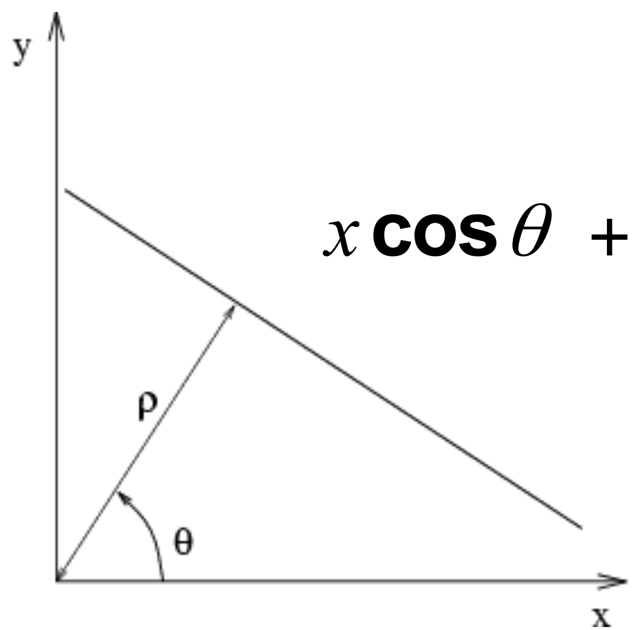
Source: S. Seitz

Parameter space representation

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

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

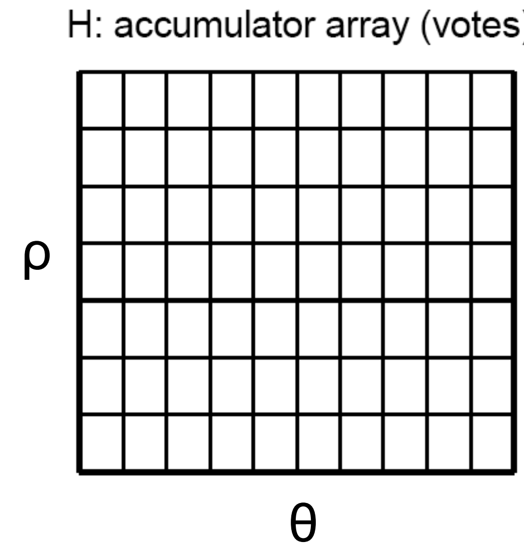
  For $\theta = 0$ to 180

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

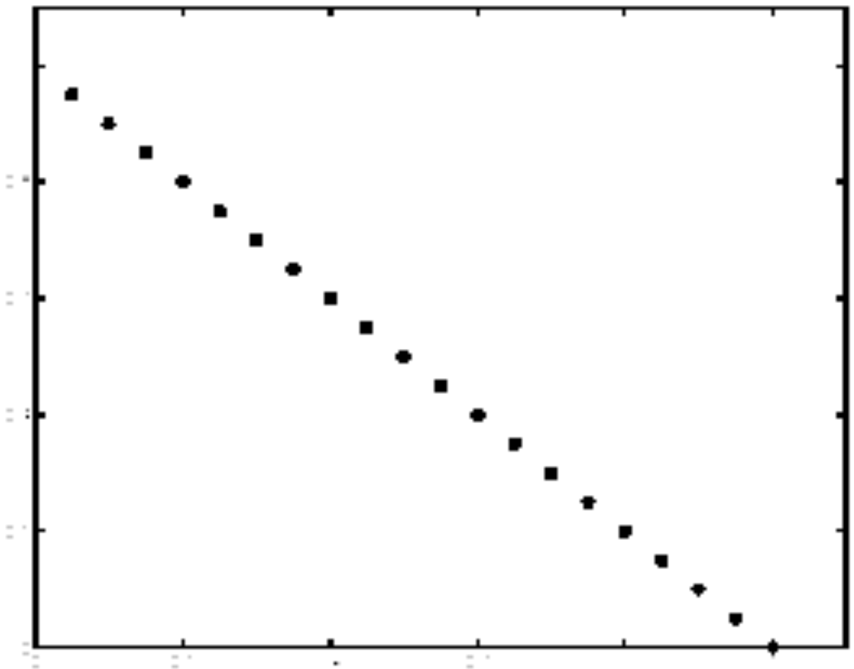
  $H(\theta, \rho) = H(\theta, \rho) + 1$

  end

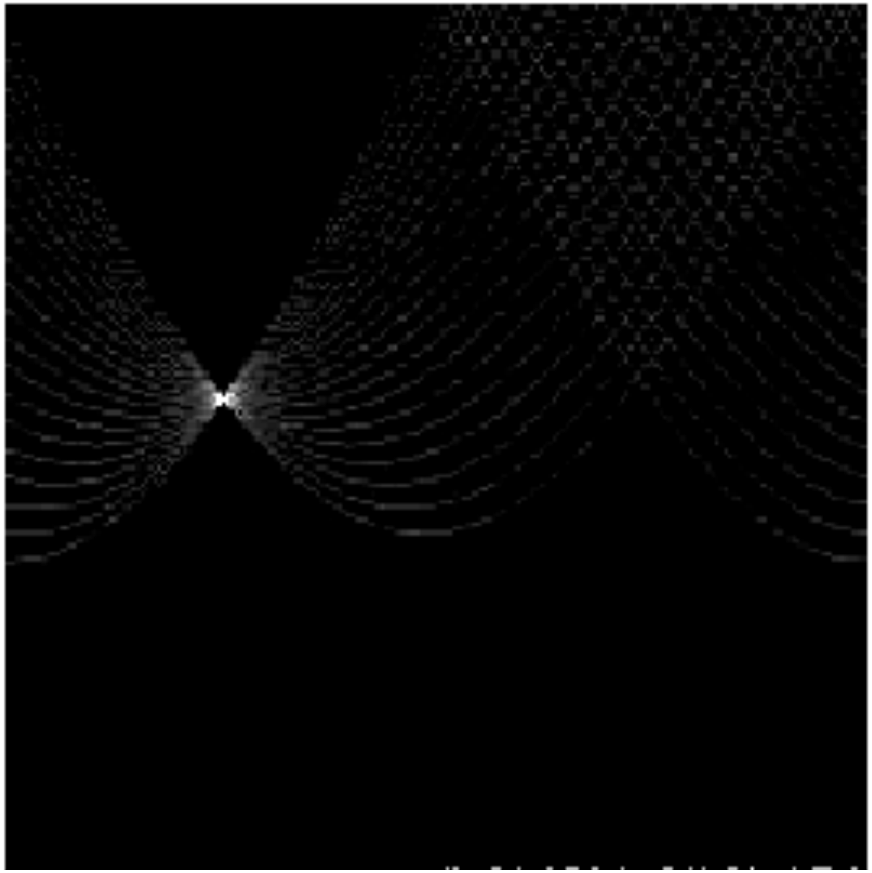
  end
- Find the value(s) of (θ, ρ) where $H(\theta, \rho)$ is a local maximum
  - The detected line in the image is given by $\rho = x \cos\theta + y \sin\theta$



Basic illustration



features

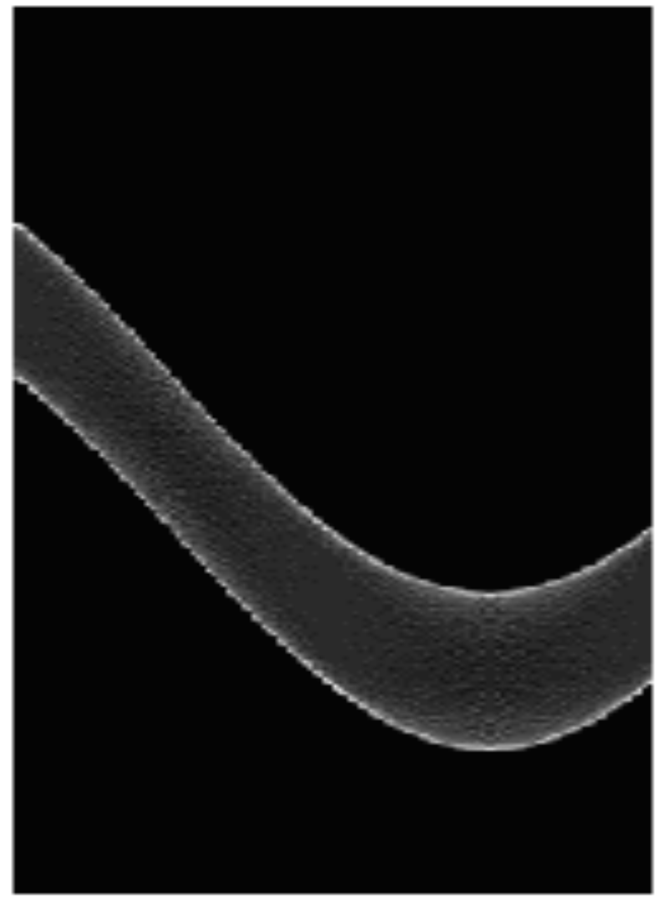


votes

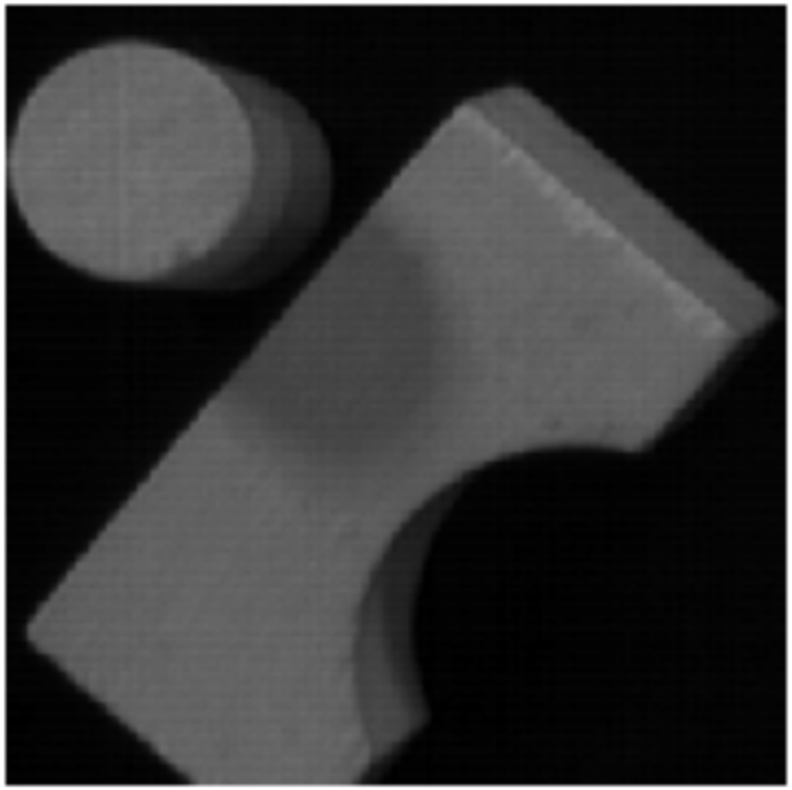
Other shapes

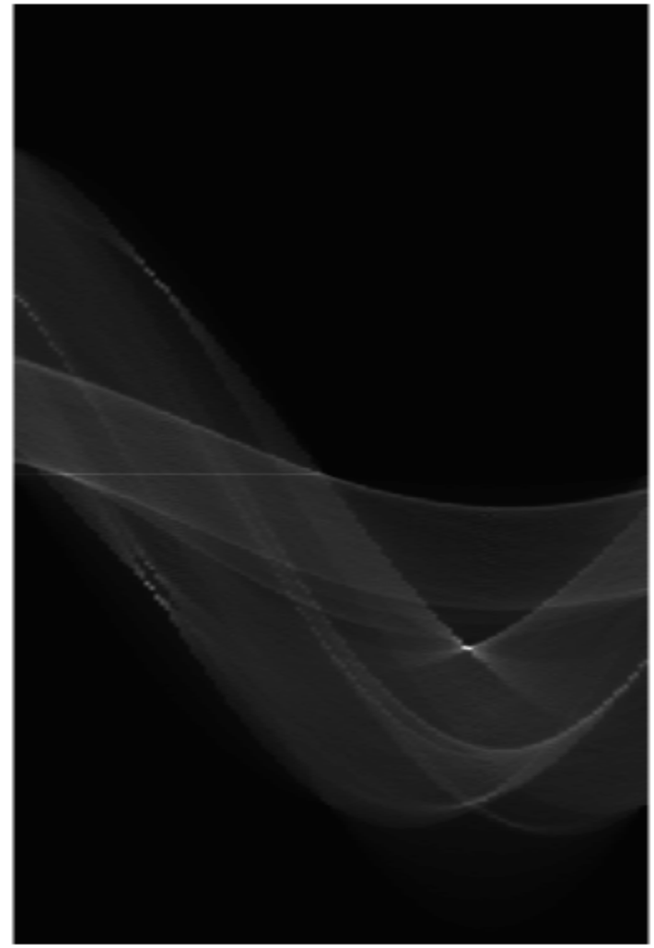
Square

Circle

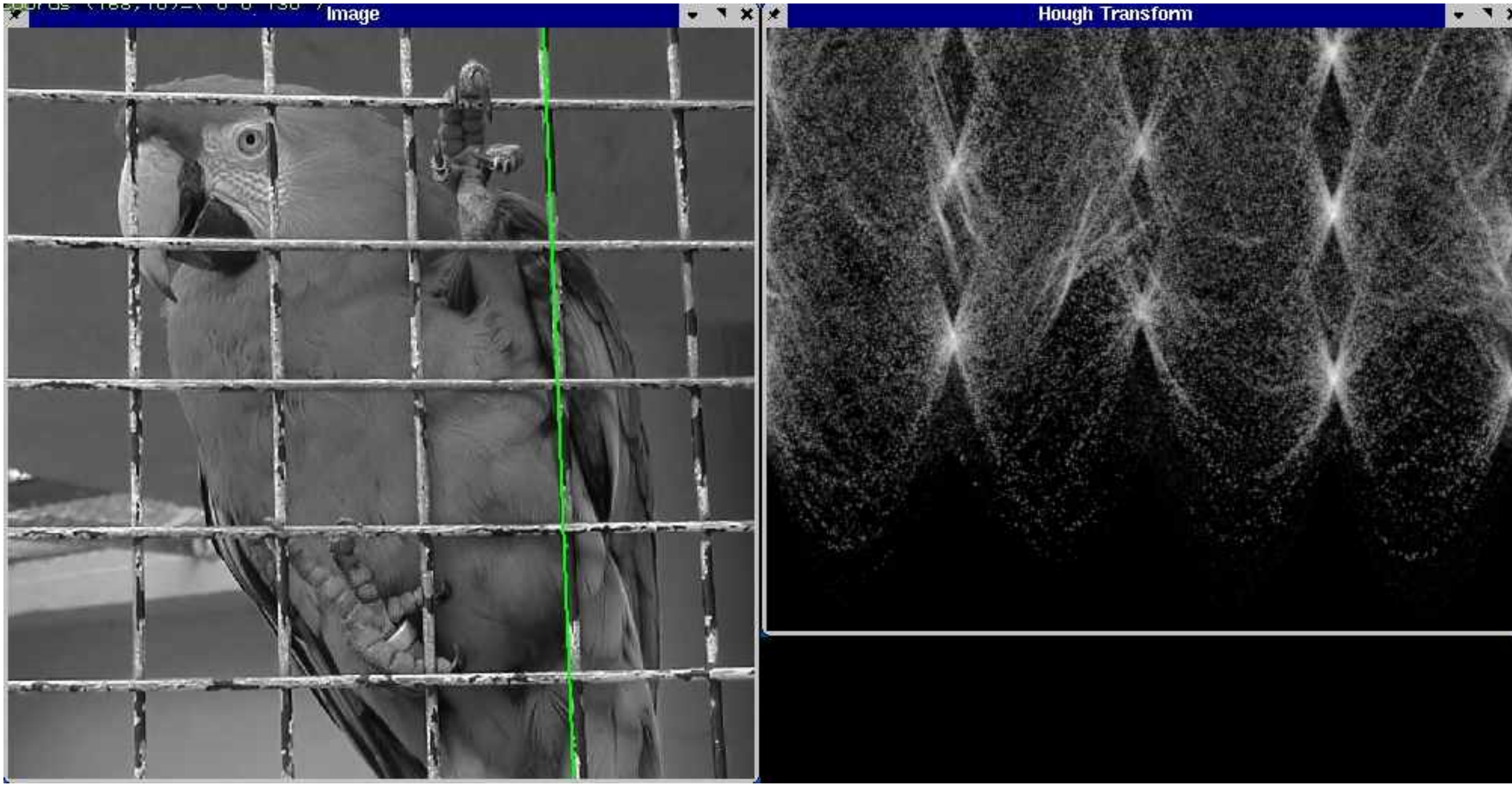


Several lines



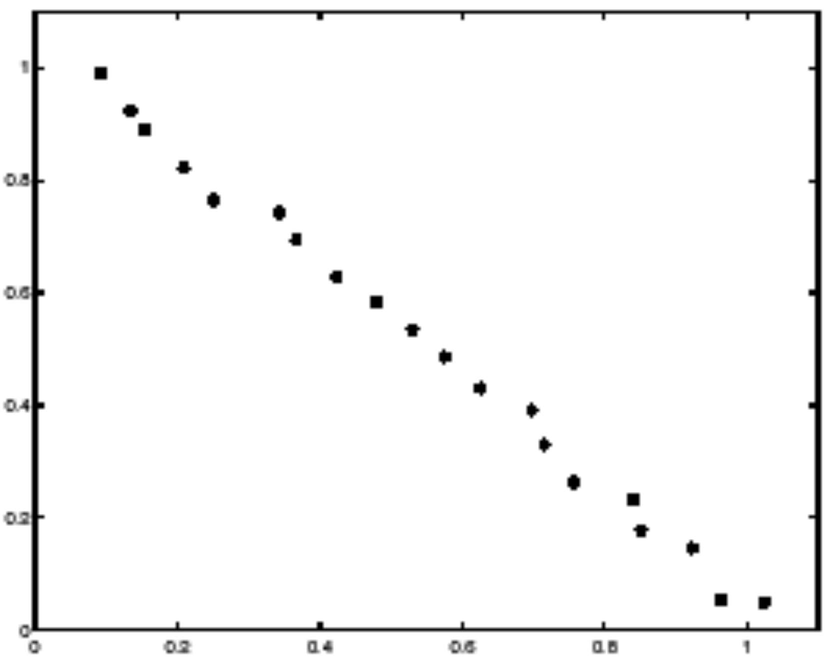


A more complicated image



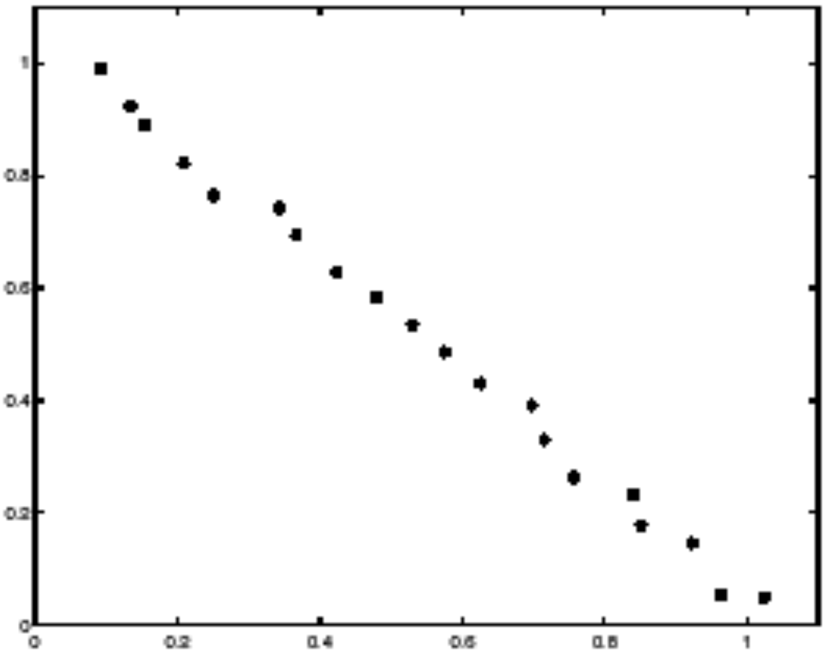
http://ostatic.com/files/images/ss_hough.jpg

Effect of noise

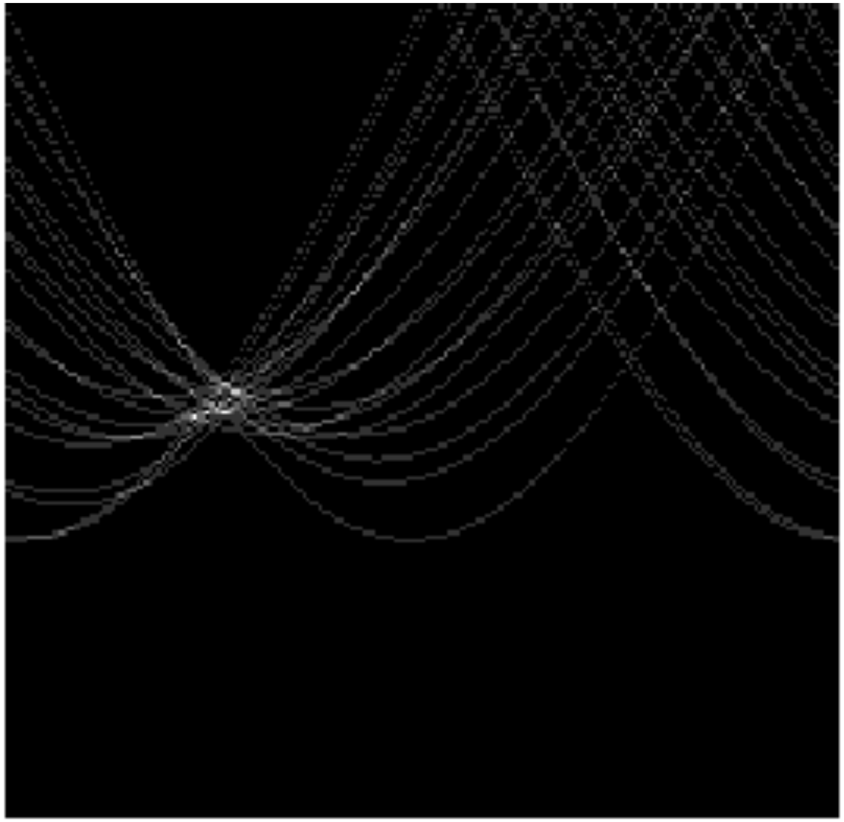


features

Effect of noise



features

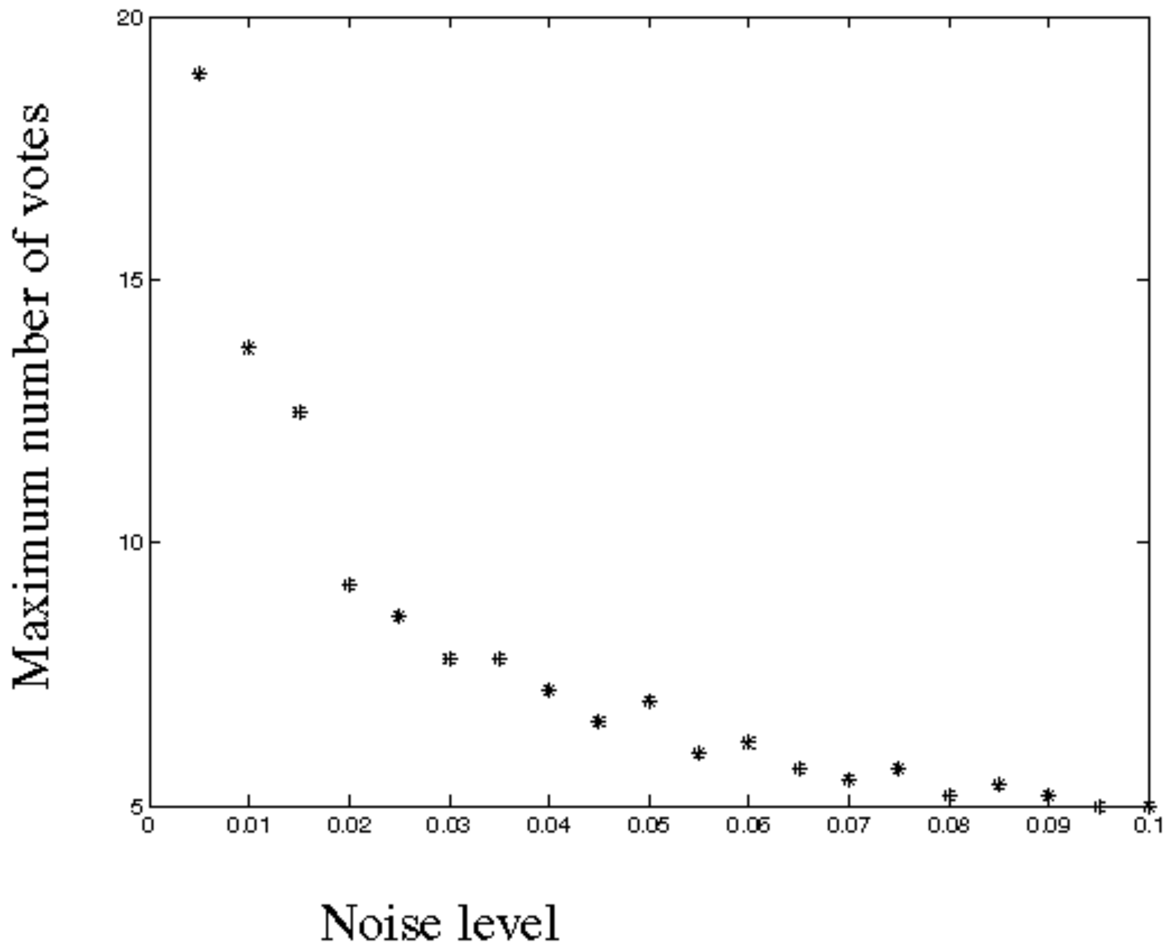


votes

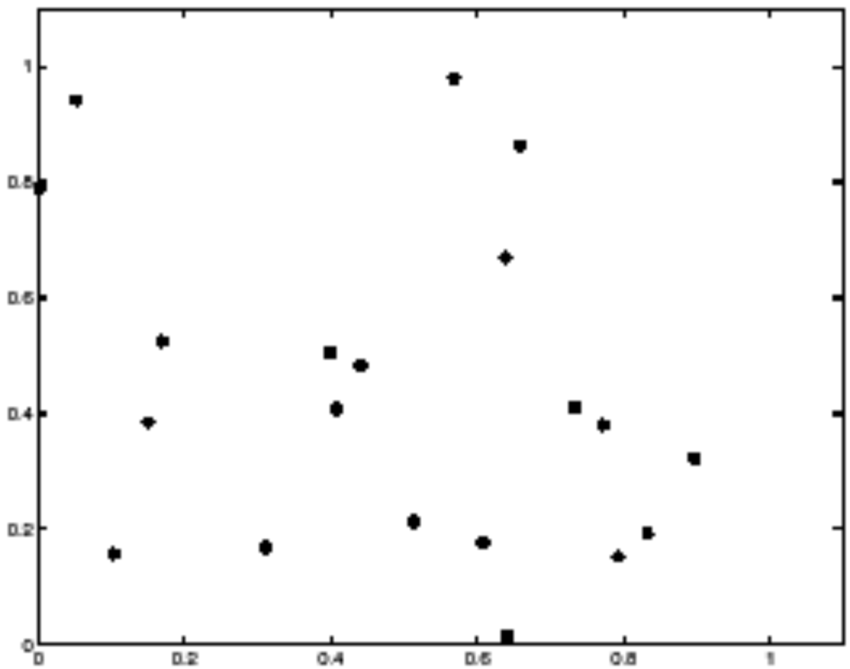
Peak gets fuzzy and hard to locate

Effect of noise

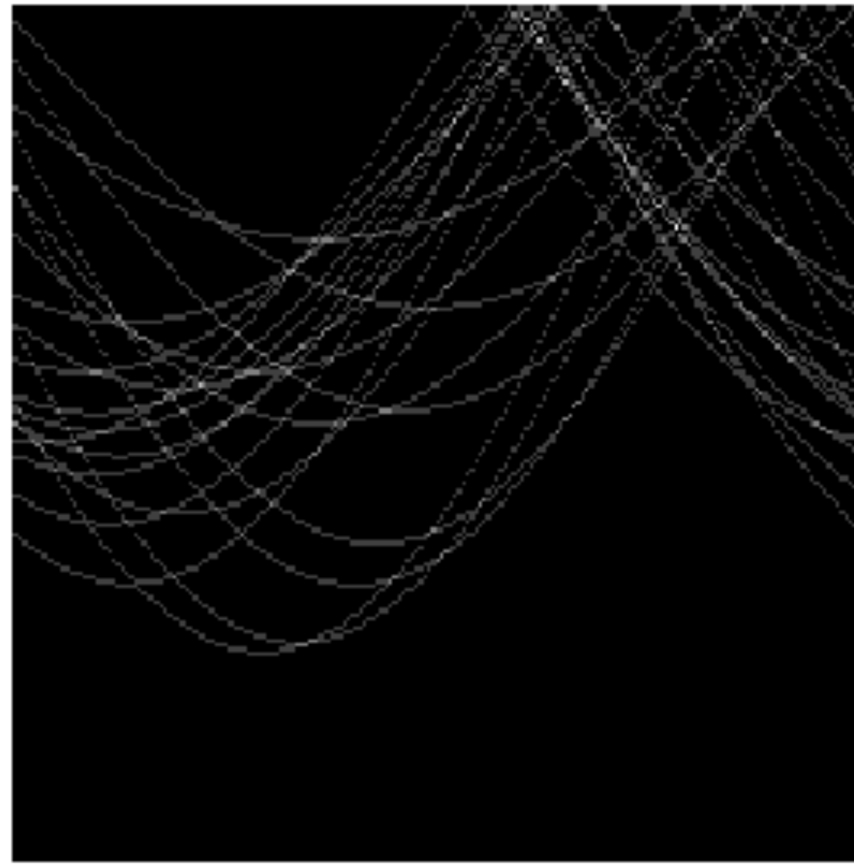
- Number of votes for a line of 20 points with increasing noise:



Random points



features

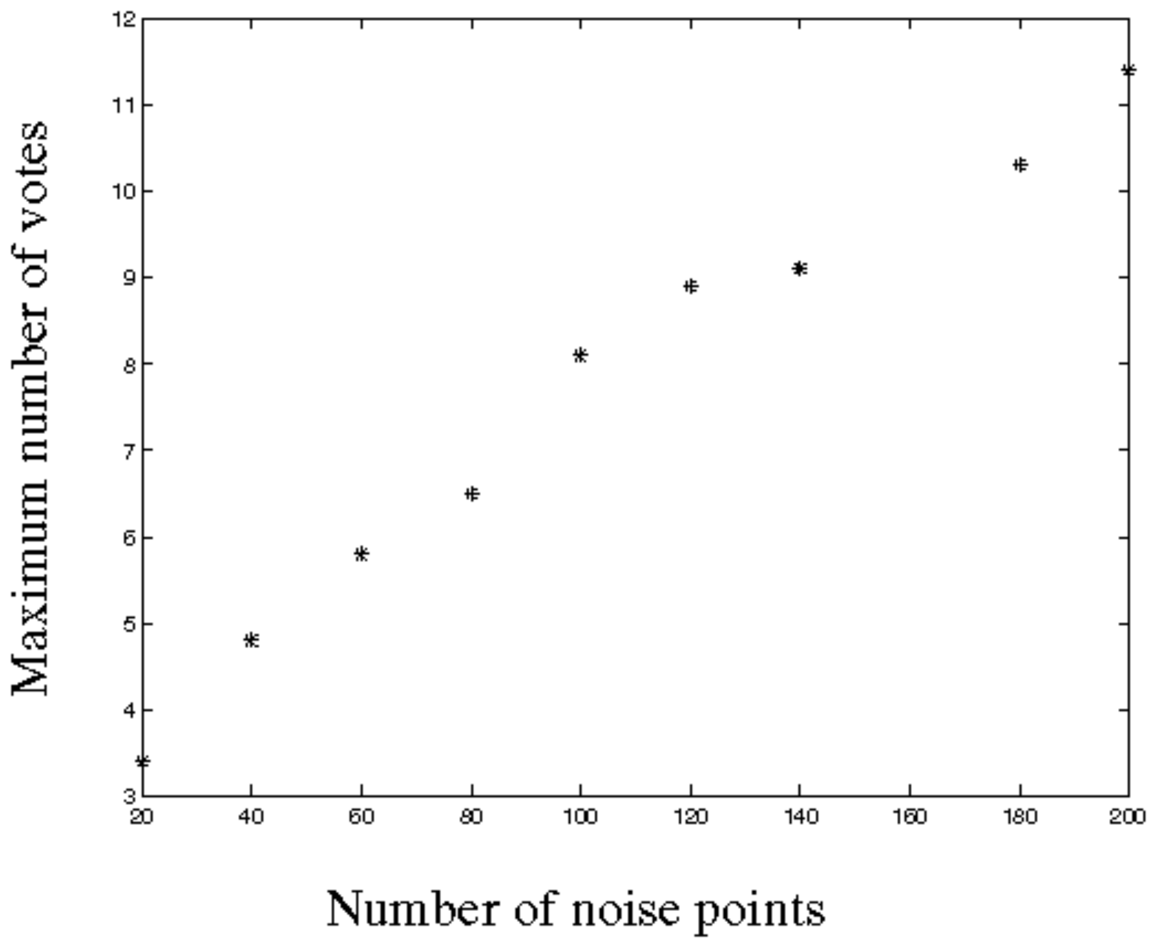


votes

Uniform noise can lead to spurious peaks in the array

Random points

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



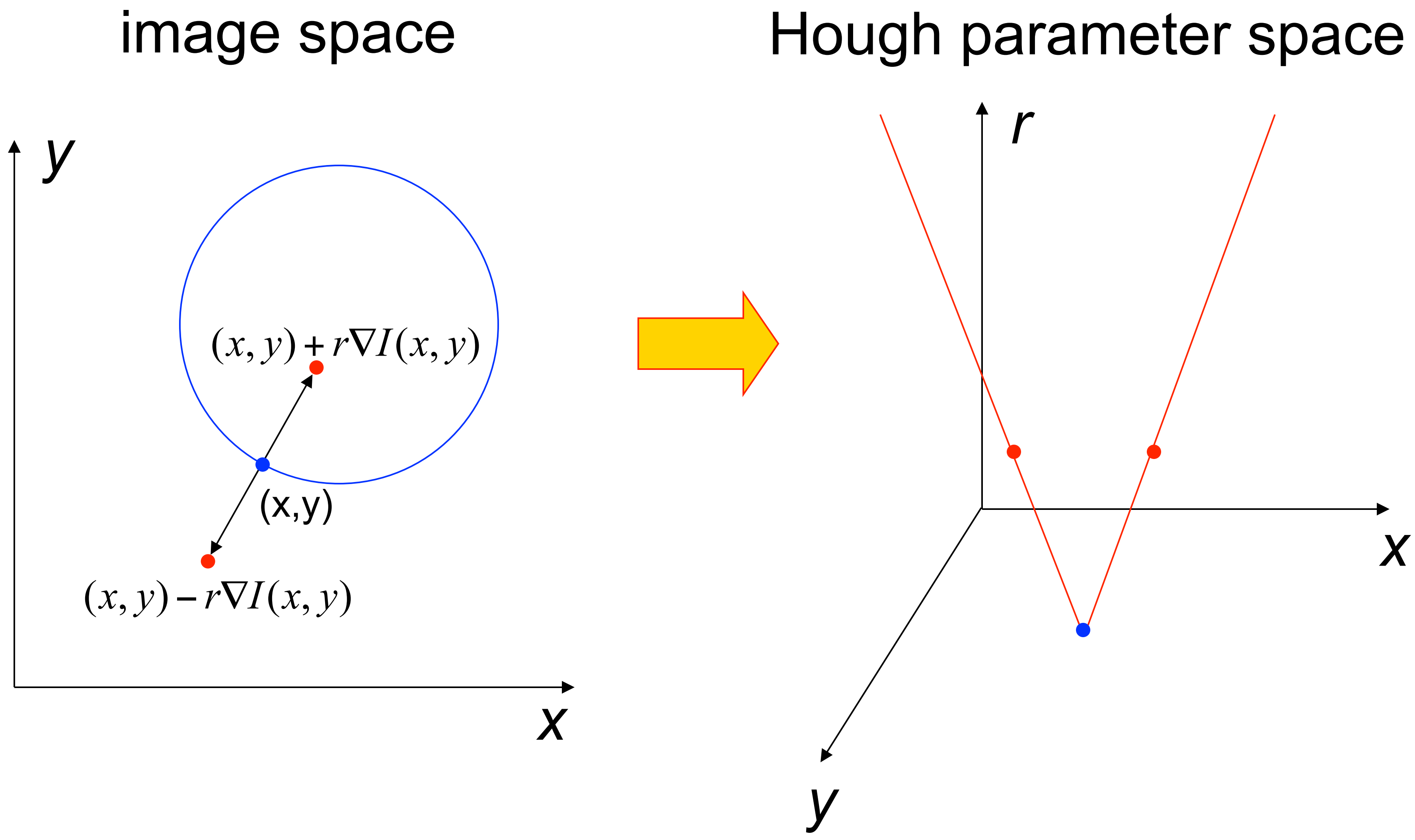
Dealing with noise

- Choose a good grid / discretization
  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
- Increment neighboring bins (smoothing in accumulator array)
- Try to get rid of irrelevant features
  - Take only edge points with significant gradient magnitude

Hough transform for circles

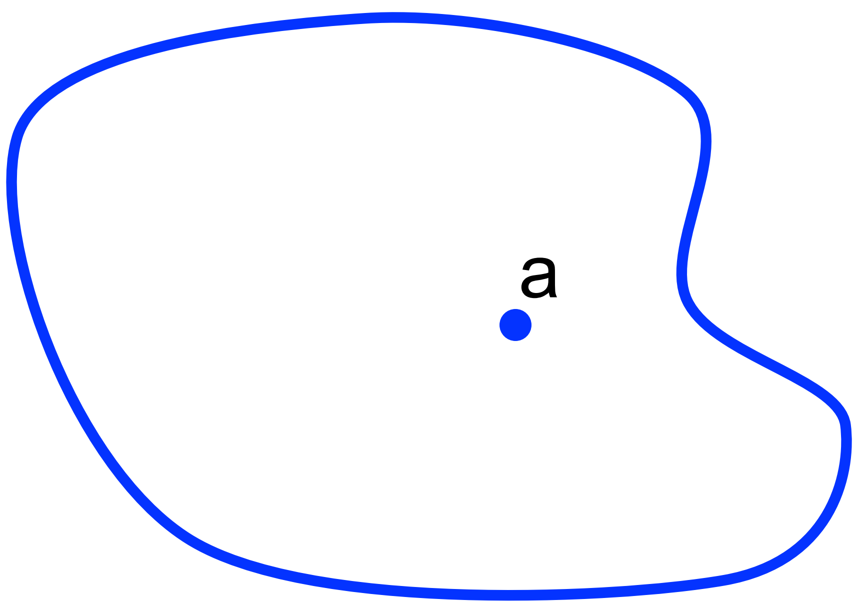
- How many dimensions will the parameter space have?
- Given an oriented edge point, what are all possible bins that it can vote for?

Hough transform for circles



Generalized Hough transform

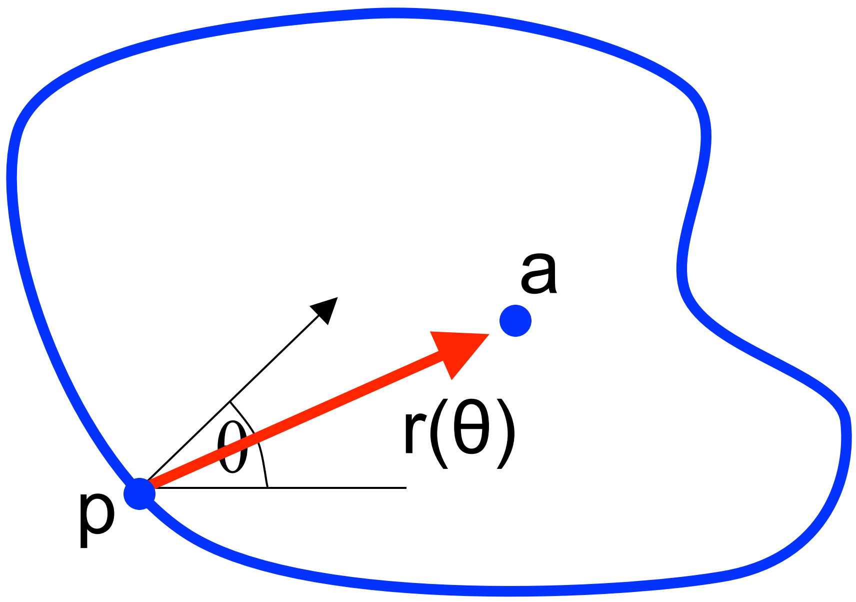
- We want to find a shape defined by its boundary points and a reference point



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

Generalized Hough transform

- We want to find a shape 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



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

Generalized Hough transform

- For model shape: construct a table indexed by θ storing displacement vectors r as function of gradient direction
- Detection: 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

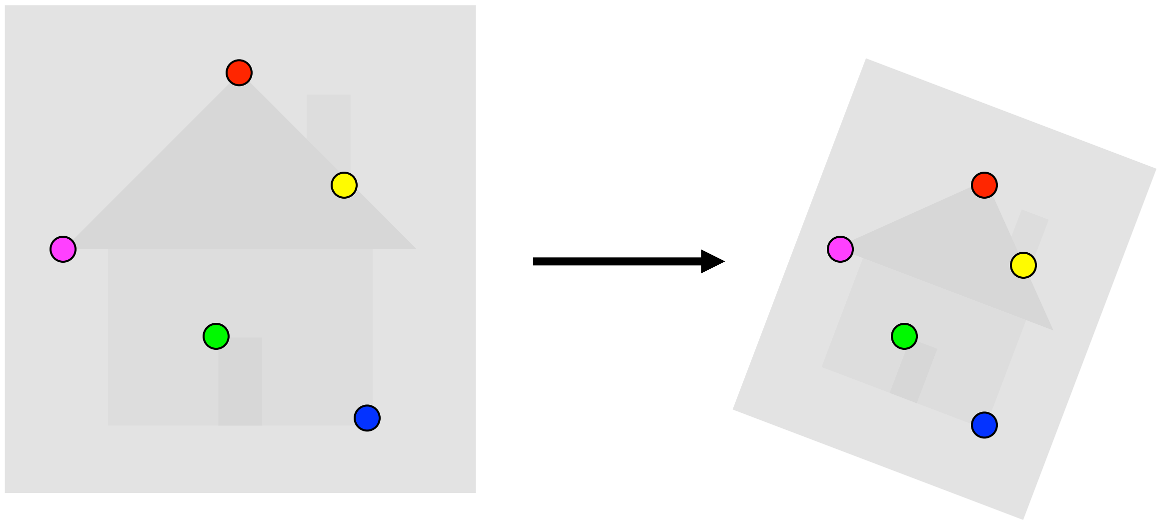
Source: K. Grauman

Overview

- Fitting techniques
  - Least Squares
  - Total Least Squares
- RANSAC
- Hough Voting

- Alignment as a fitting problem

Image alignment

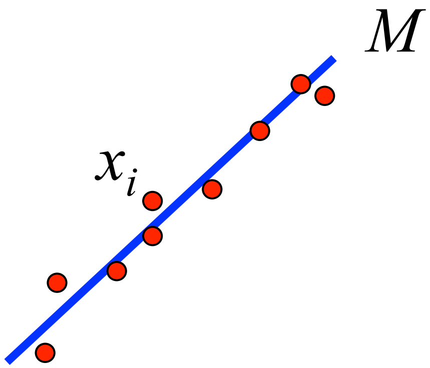


- Two broad approaches:
  - Direct (pixel-based) alignment
    - – Search for alignment where most pixels agree
  - Feature-based alignment
    - – Search for alignment where *extracted features* agree
    - – Can be verified using pixel-based alignment

Source: S. Lazebnik

Alignment as fitting

- Previously: fitting a model to features in one image



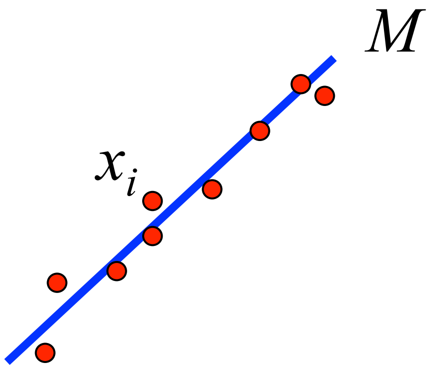
Find model *M* that minimizes

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

Source: S. Lazebnik

Alignment as fitting

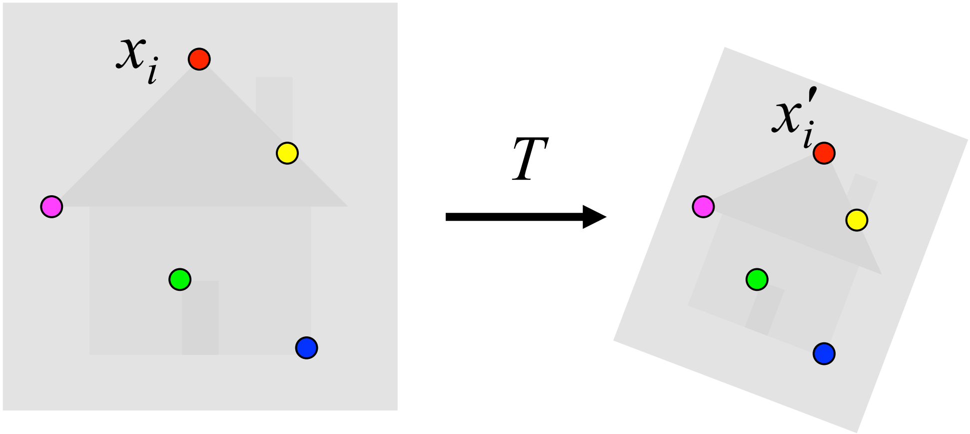
- Previously: fitting a model to features in one image



Find model M that minimizes

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

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



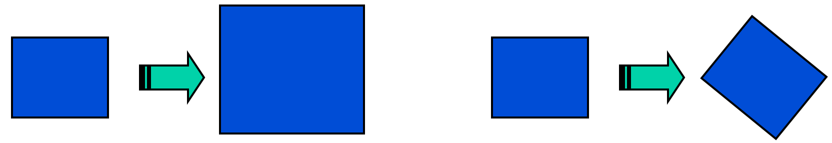
Find transformation T that minimizes

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

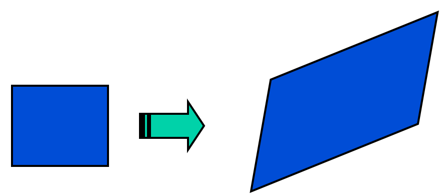
Source: S. Lazebnik

2D transformation models

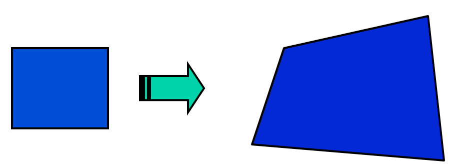
- Similarity (translation, scale, rotation)



- Affine



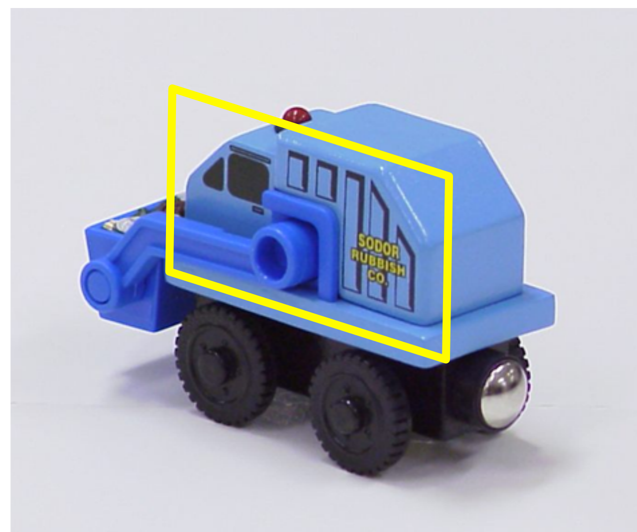
- Projective (homography)



Source: S. Lazebnik

Let’s start with affine transformations

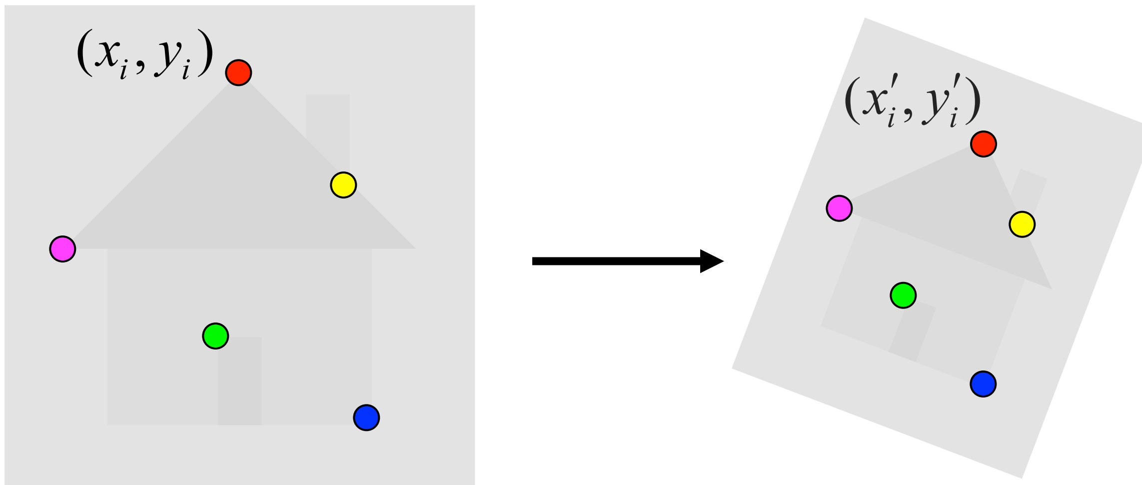
- Simple fitting procedure (linear least squares)
- Approximates viewpoint changes for roughly planar objects and roughly orthographic cameras
- Can be used to initialize fitting for more complex models



Source: S. Lazebnik

Fitting an affine transformation

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



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

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

Source: S. Lazebnik

Fitting an affine transformation

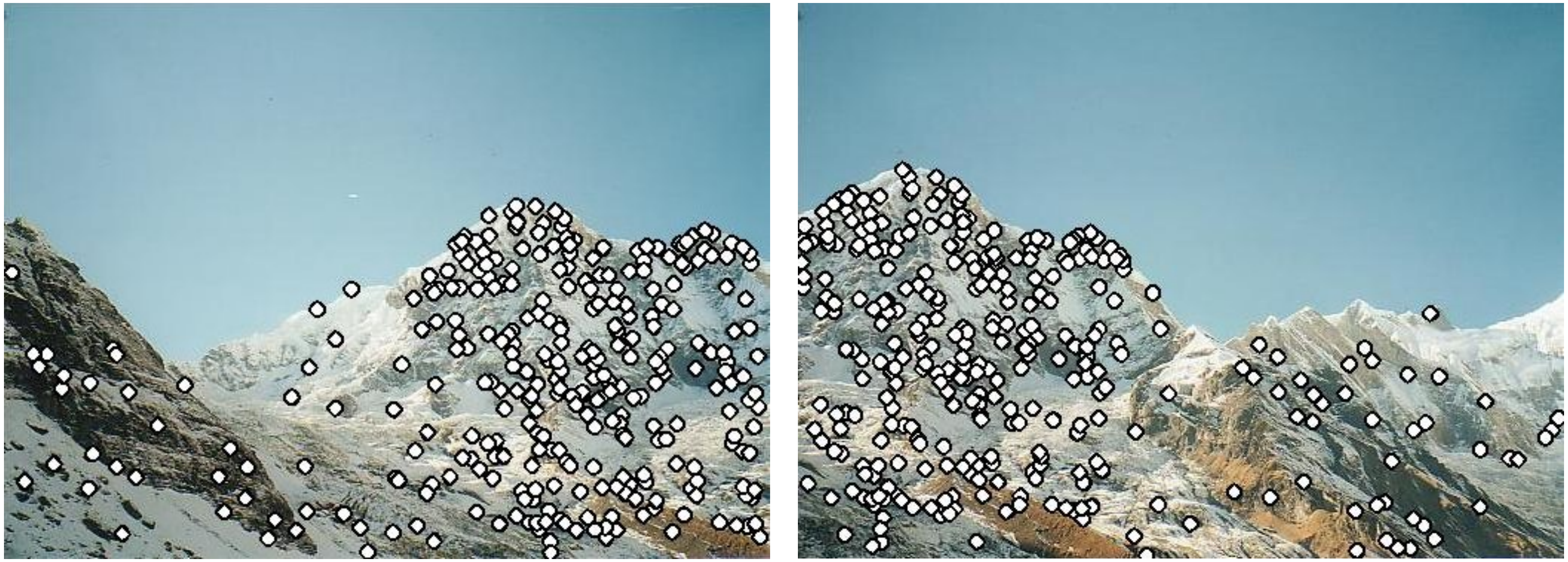
$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

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

Source: S. Lazebnik

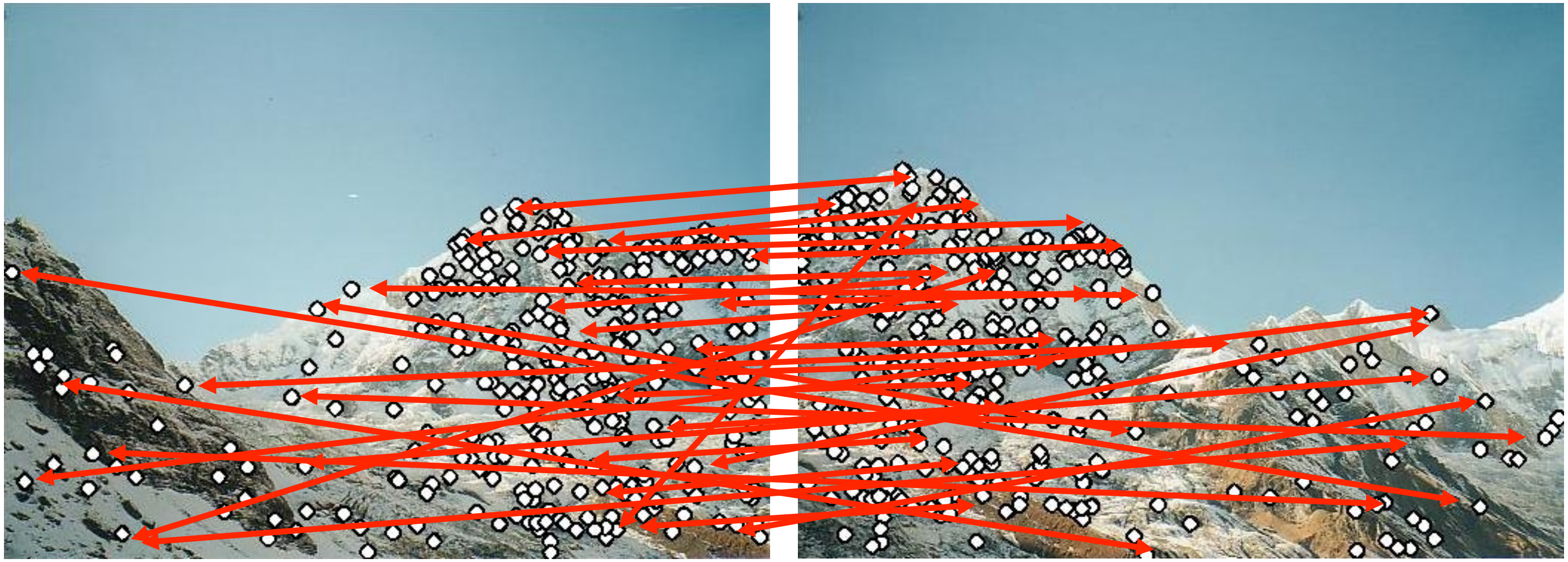
Feature-based alignment outline

Feature-based alignment outline



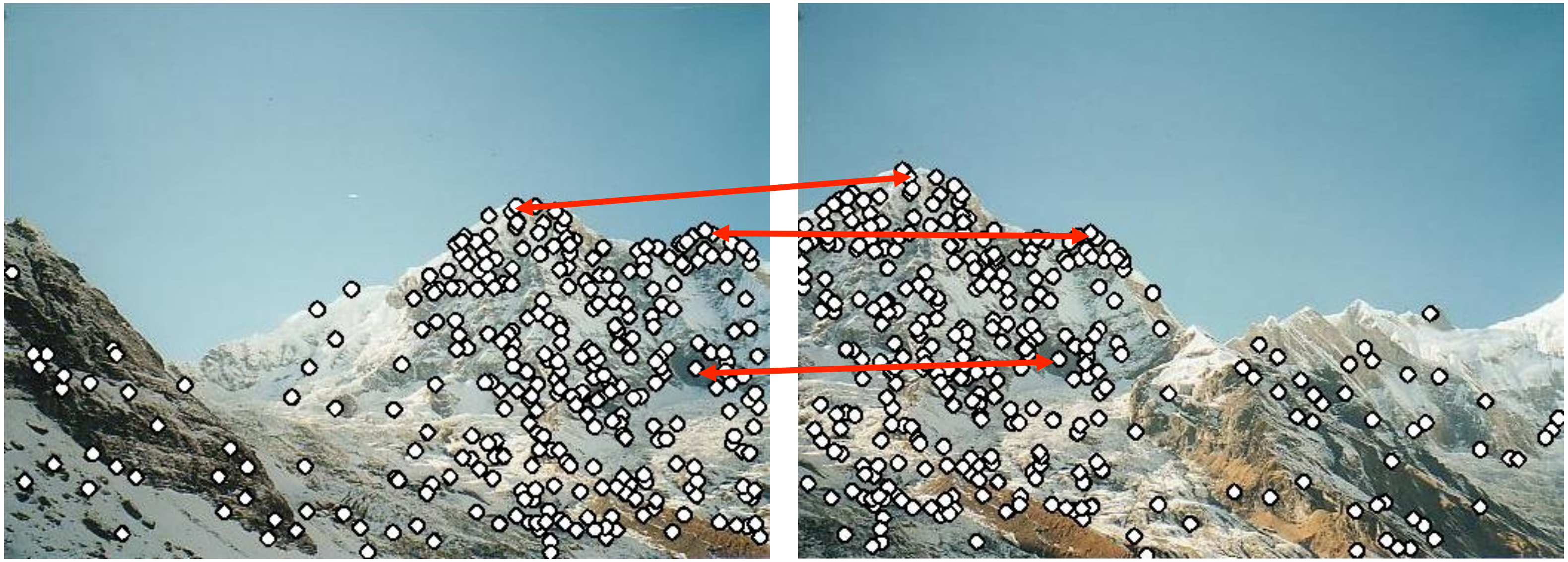
- Extract features

Feature-based alignment outline



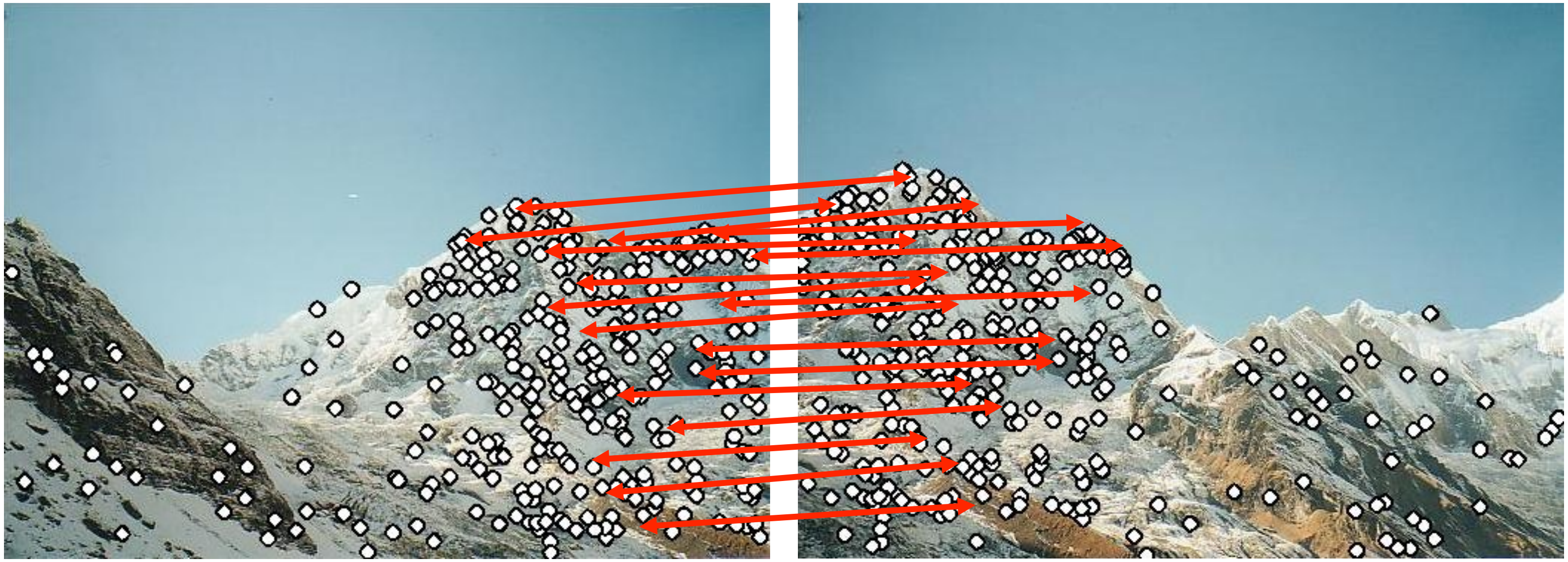
- Extract features
- Compute *putative matches*

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*)

Dealing with outliers

- The set of putative matches contains a very high percentage of outliers
- Geometric fitting strategies:
  - RANSAC
  - Hough transform

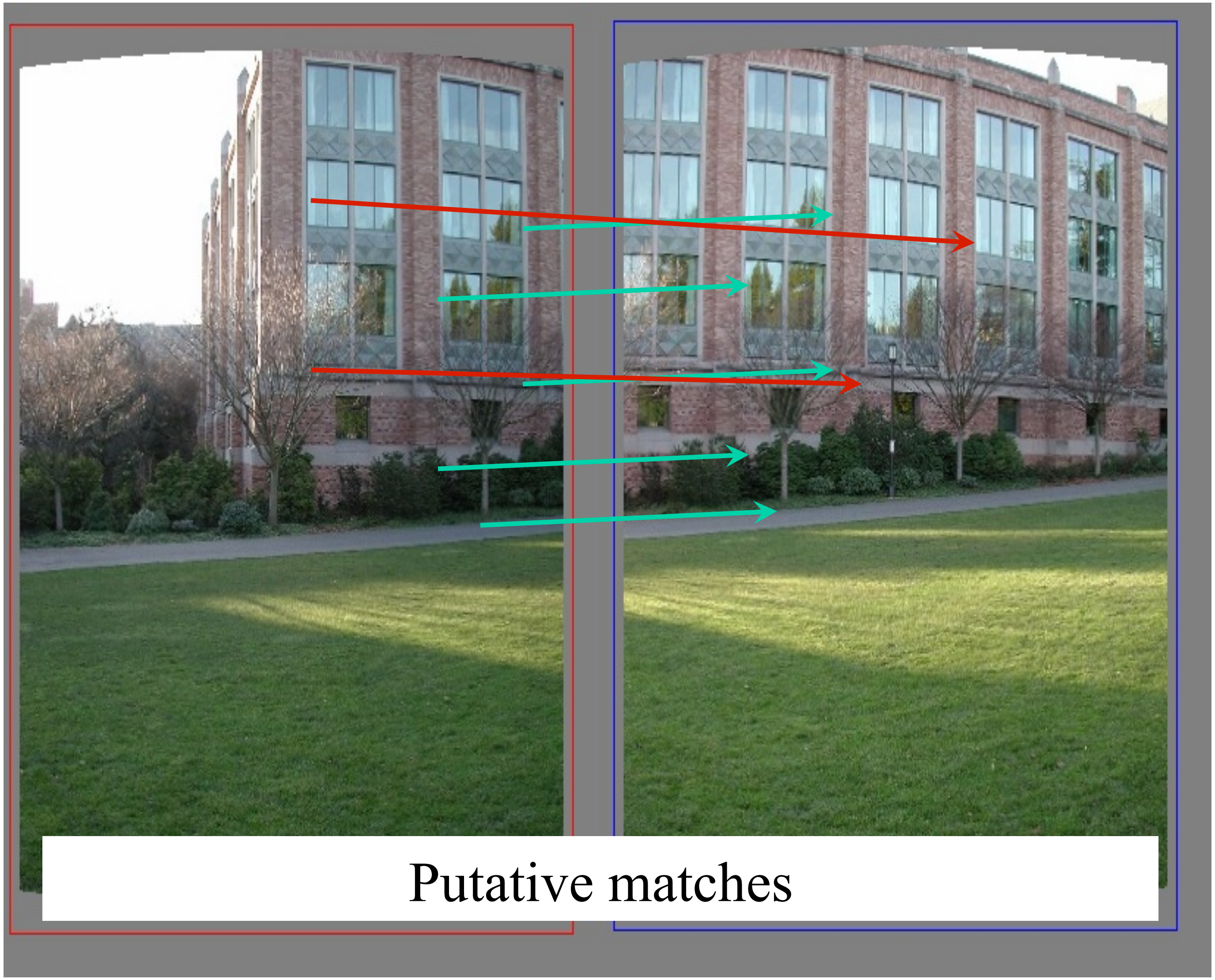
RANSAC

RANSAC loop:

1. Randomly select a *seed 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

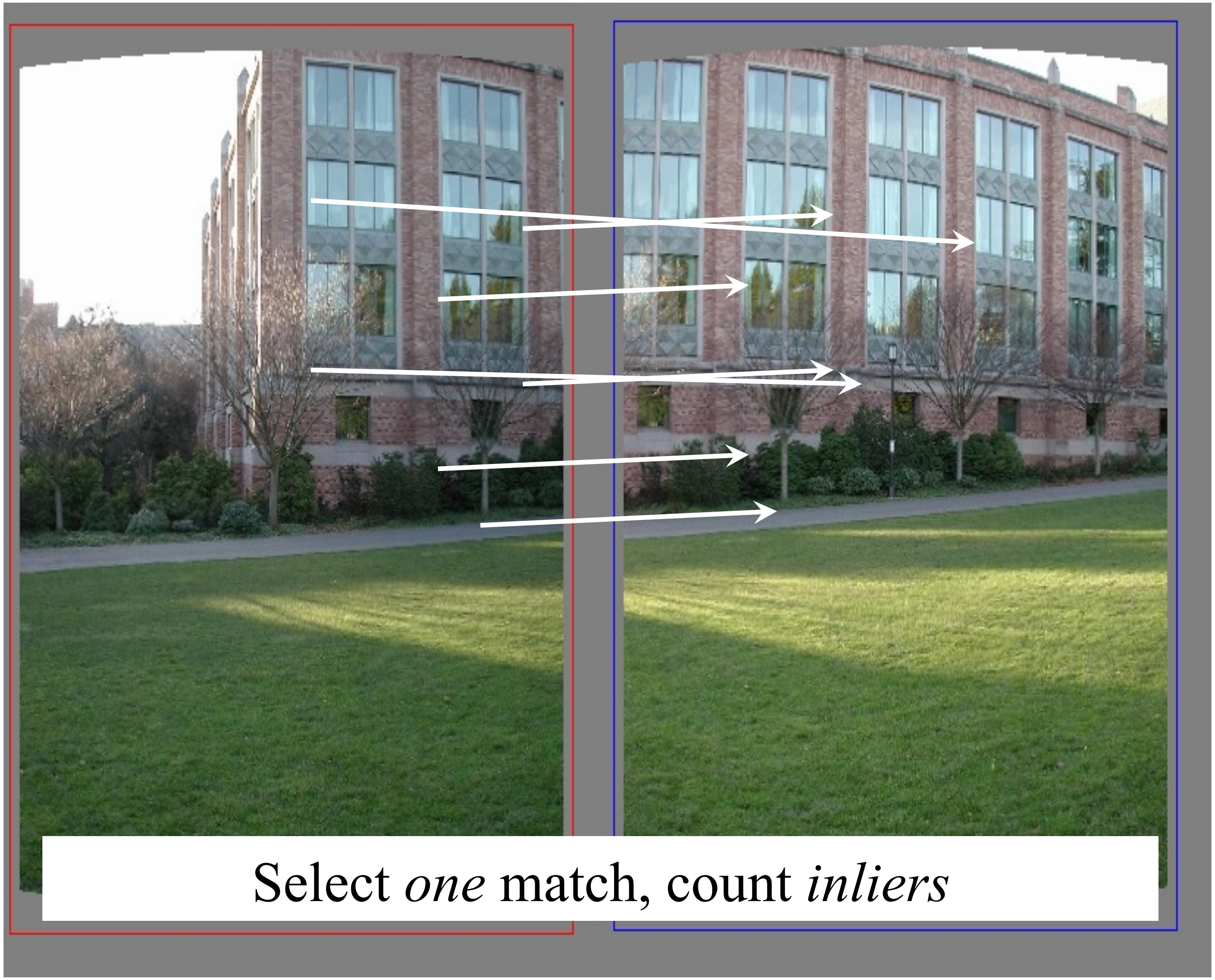
Keep the transformation with the largest number of inliers

RANSAC example: Translation



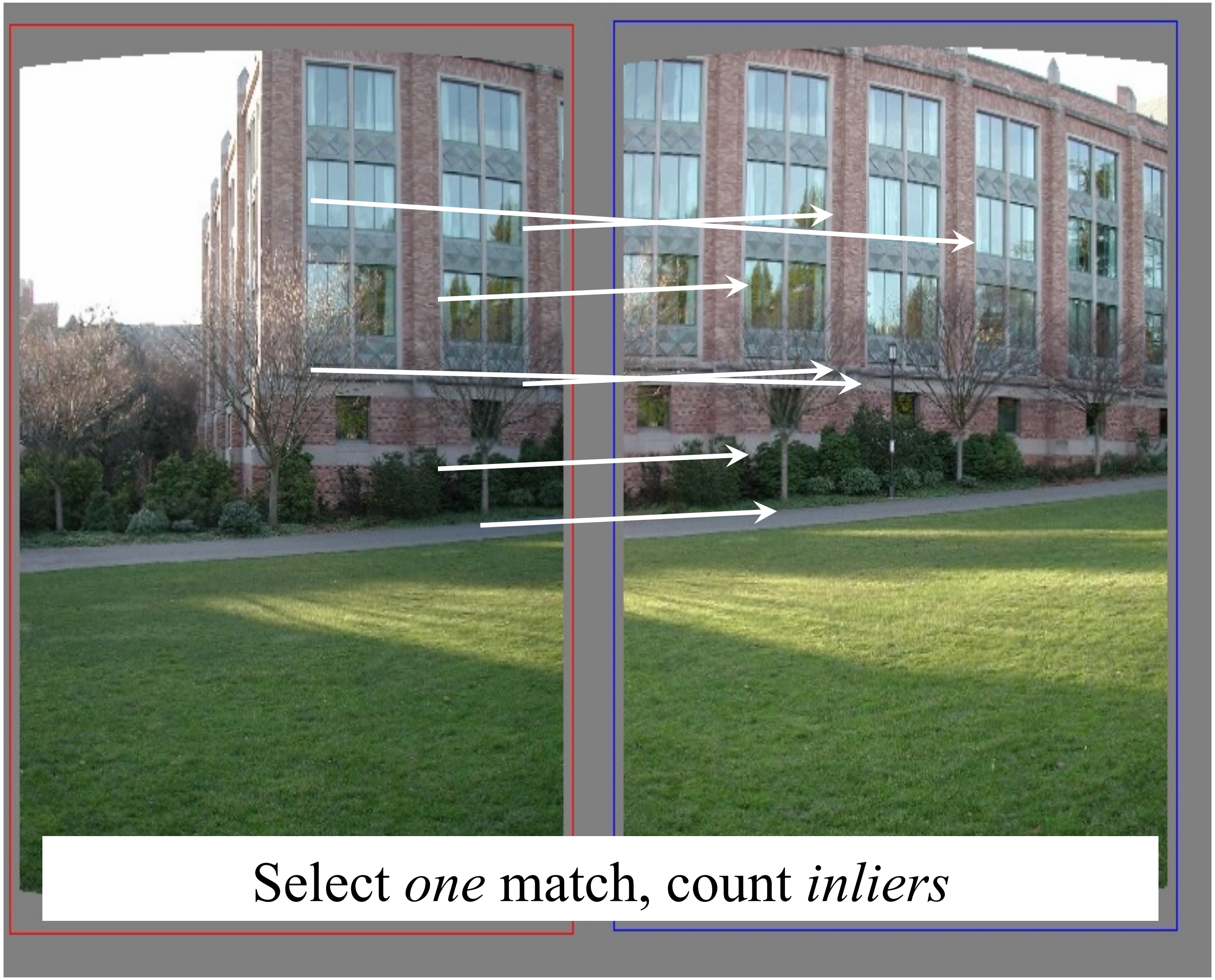
Source: A. Efros

RANSAC example: Translation



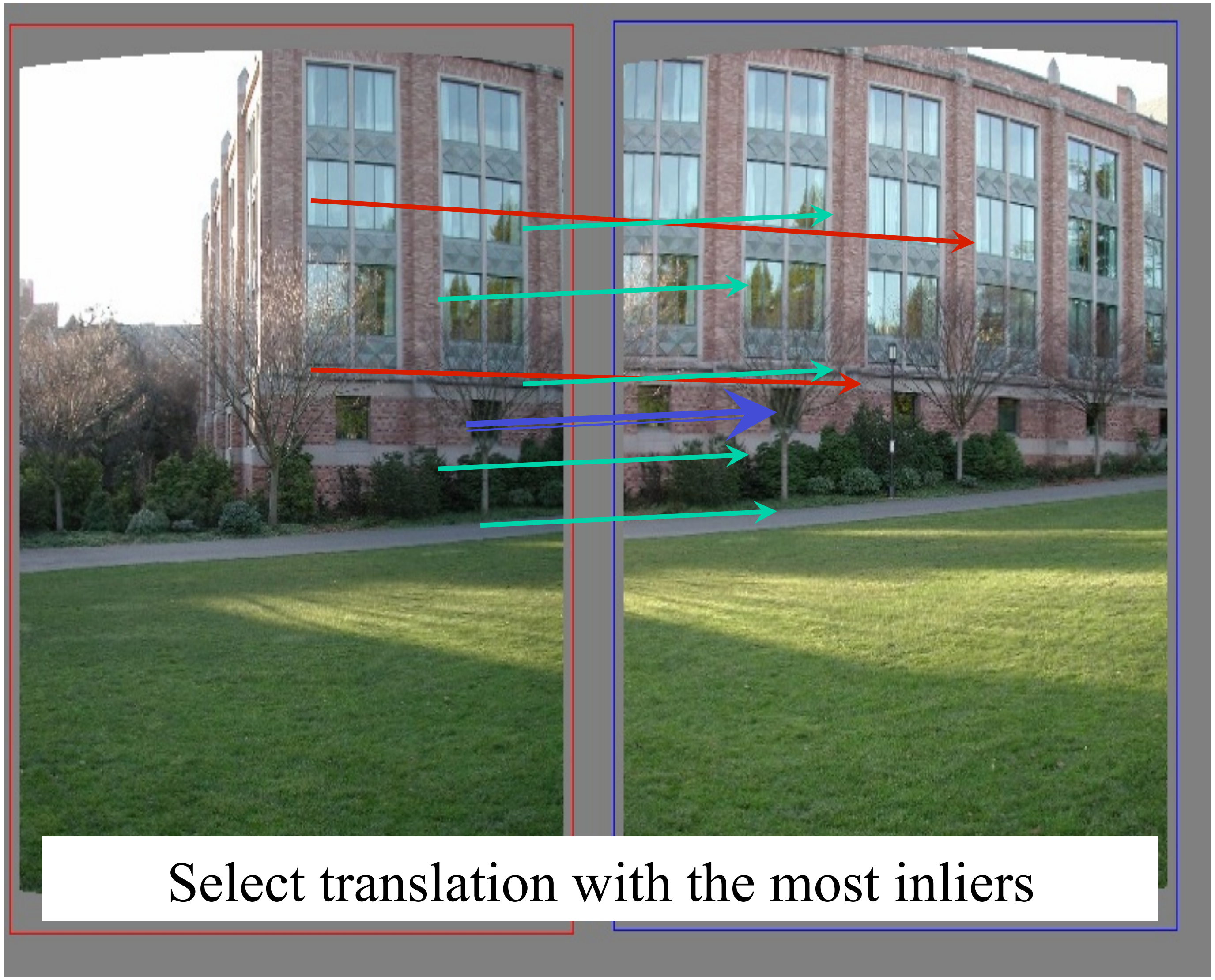
Source: A. Efros

RANSAC example: Translation



Source: A. Efros

RANSAC example: Translation



Source: A. Efros

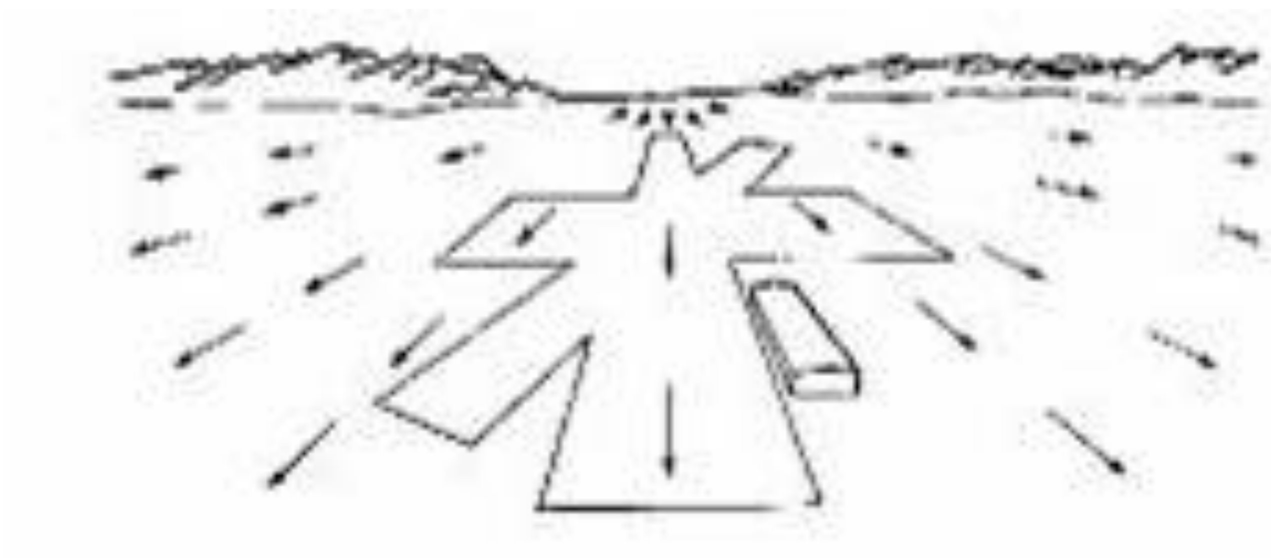
Motion estimation techniques

- Feature-based methods
  - Extract visual features (corners, textured areas) and track them over multiple frames
  - Sparse motion fields, but more robust tracking
  - Suitable when image motion is large (10s of pixels)

- Direct methods
  - Directly recover image motion at each pixel from spatio-temporal image brightness variations
  - Dense motion fields, but sensitive to appearance variations
  - Suitable for video and when image motion is small

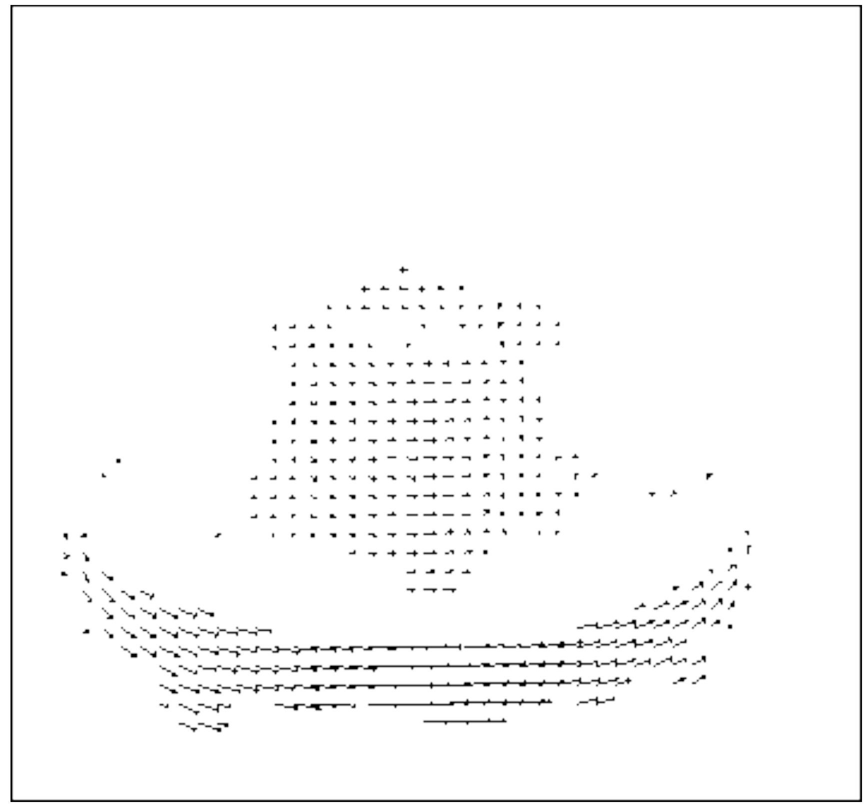
Optical flow

Combination of slides from Rick Szeliski, Steve Seitz, Alyosha Efros and Bill Freeman and Fredo Durand



Motion estimation: Optical flow





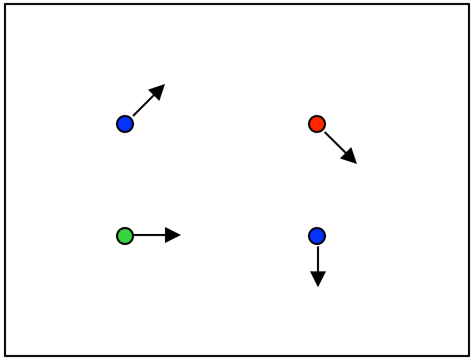
Will start by estimating motion of each pixel separately
Then will consider motion of entire image

Why estimate motion?

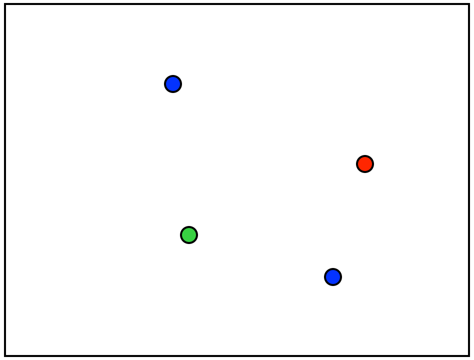
Lots of uses

- Track object behavior
- Correct for camera jitter (stabilization)
- Align images (mosaics)
- 3D shape reconstruction
- Special effects

Problem definition: optical flow



$H(x, y)$



$I(x, y)$

How to estimate pixel motion from image H to image I?

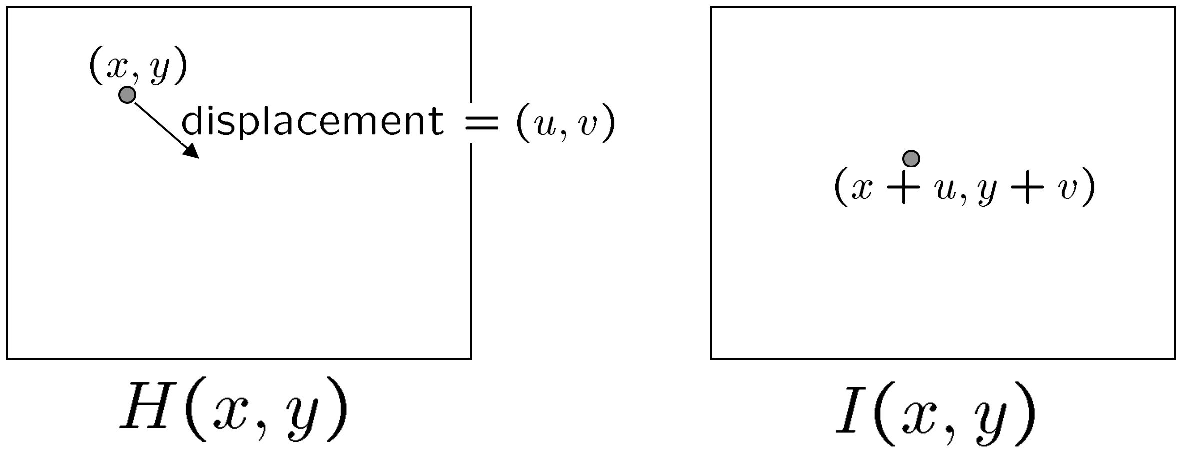
- Solve pixel correspondence problem
  - given a pixel in H, look for nearby pixels of the same color in I

Key assumptions

- color constancy: a point in H looks the same in I
  - For grayscale images, this is brightness constancy
- small motion: points do not move very far

This is called the optical flow problem

Optical flow constraints (grayscale images)



Let's look at these constraints more closely

- brightness constancy: Q: what's the equation?

  H(x,y)=I(x+u, y+v)

- small motion: (u and v are less than 1 pixel)
  - suppose we take the Taylor series expansion of I:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \text{higher order terms}$$

$$\approx I(x, y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v$$

Optical flow equation

Combining these two equations

shorthand: $I_x = \frac{\partial I}{\partial x}$

$$0 = I(x+u, y+v) - H(x,y)$$

$$\approx I(x,y) + I_x u + I_y v - H(x,y)$$

$$\approx (I(x,y) - H(x,y)) + I_x u + I_y v$$

$$\approx I_t + I_x u + I_y v$$

$$\approx I_t + \nabla I \cdot [u \;\; v]$$

In the limit as u and v go to zero, this becomes exact

$$0 = I_t + \nabla I \cdot [\tfrac{\partial x}{\partial t} \;\; \tfrac{\partial y}{\partial t}]$$

Optical flow equation

$$0 = I_t + \nabla I \cdot [u \;\; v]$$

Q: how many unknowns and equations per pixel?

2 unknowns, one equation

Intuitively, what does this constraint mean?

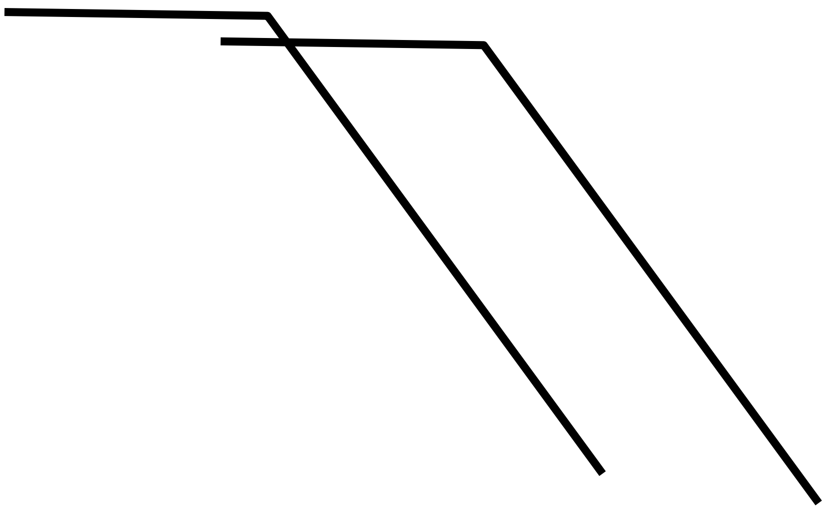
- The component of the flow in the gradient direction is determined
- The component of the flow parallel to an edge is unknown

This explains the Barber Pole illusion

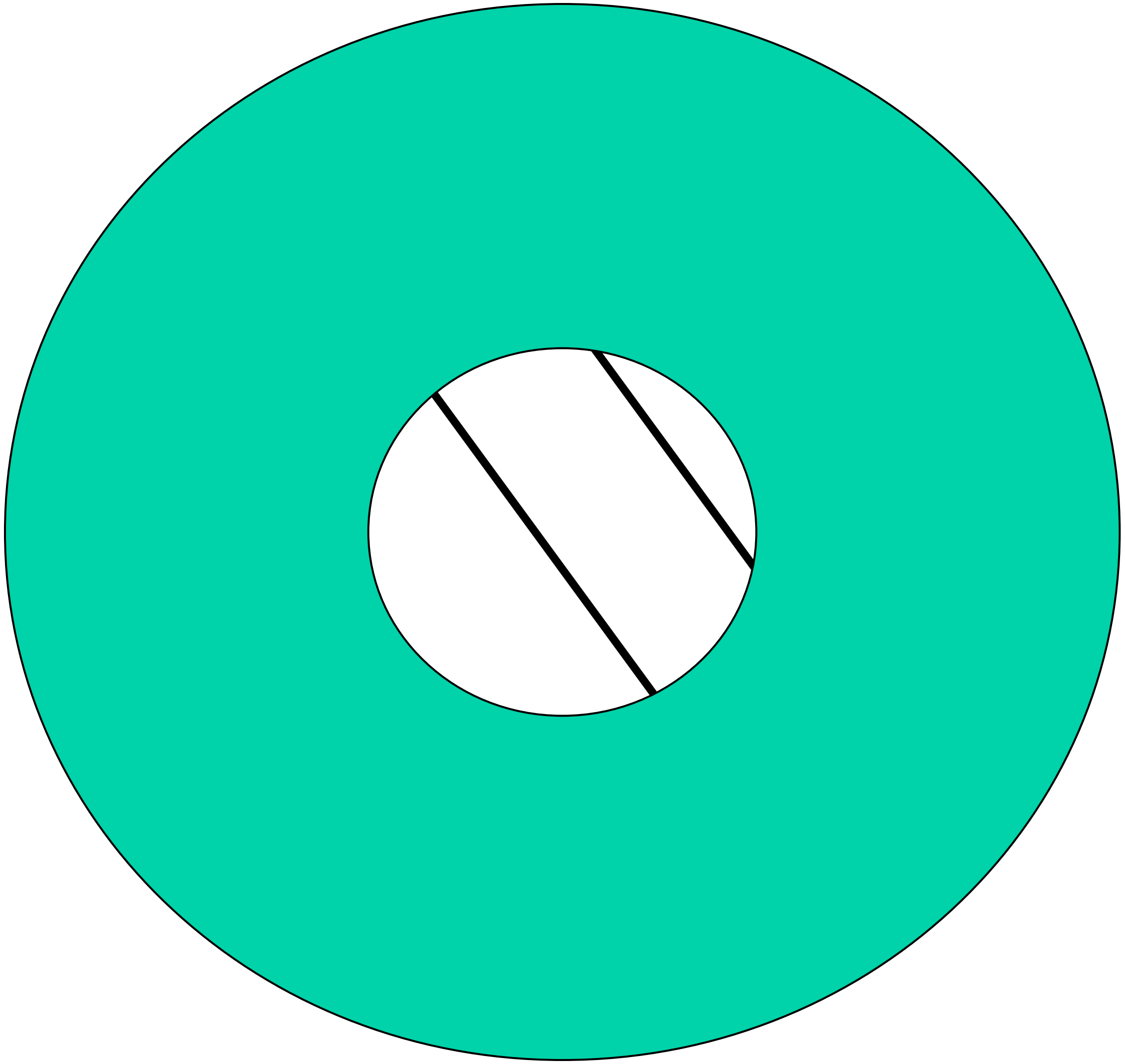
http://www.sandlotscience.com/Ambiguous/Barberpole_Illusion.htm
http://www.liv.ac.uk/~marcob/Trieste/barberpole.html

http://en.wikipedia.org/wiki/Barber's_pol

Aperture problem



Aperture problem



Solving the aperture problem

How to get more equations for a pixel?

- Basic idea: impose additional constraints
  - most common is to assume that the flow field is smooth locally
  - one method: pretend the pixel's neighbors have the same (u,v)
    - » If we use a 5x5 window, that gives us 25 equations per pixel!

$$0 = I_t(\mathbf{p_i}) + \nabla I(\mathbf{p_i}) \cdot [u \ \ v]$$

$$\underset{25\times2}{\underset{A}{\begin{bmatrix} I_x(\mathbf{p_1}) & I_y(\mathbf{p_1}) \\ I_x(\mathbf{p_2}) & I_y(\mathbf{p_2}) \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}}) & I_y(\mathbf{p_{25}}) \end{bmatrix}}} \underset{2\times1}{\underset{d}{\begin{bmatrix} u \\ v \end{bmatrix}}} = - \underset{25\times1}{\underset{b}{\begin{bmatrix} I_t(\mathbf{p_1}) \\ I_t(\mathbf{p_2}) \\ \vdots \\ I_t(\mathbf{p_{25}}) \end{bmatrix}}}$$

RGB version

How to get more equations for a pixel?

- Basic idea: impose additional constraints
  - most common is to assume that the flow field is smooth locally
  - one method: pretend the pixel's neighbors have the same (u,v)
    - If we use a 5x5 window, that gives us 25*3 equations per pixel!

$$0 = I_t(\mathbf{p_i})[0,1,2] + \nabla I(\mathbf{p_i})[0,1,2] \cdot [u \ \ v]$$

$$\underset{\substack{A \\ 75\text{x}2}}{\begin{bmatrix} I_x(\mathbf{p_1})[0] & I_y(\mathbf{p_1})[0] \\ I_x(\mathbf{p_1})[1] & I_y(\mathbf{p_1})[1] \\ I_x(\mathbf{p_1})[2] & I_y(\mathbf{p_1})[2] \\ \vdots & \vdots \\ I_x(\mathbf{p_{25}})[0] & I_y(\mathbf{p_{25}})[0] \\ I_x(\mathbf{p_{25}})[1] & I_y(\mathbf{p_{25}})[1] \\ I_x(\mathbf{p_{25}})[2] & I_y(\mathbf{p_{25}})[2] \end{bmatrix}} \underset{\substack{d \\ 2\text{x}1}}{\begin{bmatrix} u \\ v \end{bmatrix}} = - \underset{\substack{b \\ 75\text{x}1}}{\begin{bmatrix} I_t(\mathbf{p_1})[0] \\ I_t(\mathbf{p_1})[1] \\ I_t(\mathbf{p_1})[2] \\ \vdots \\ I_t(\mathbf{p_{25}})[0] \\ I_t(\mathbf{p_{25}})[1] \\ I_t(\mathbf{p_{25}})[2] \end{bmatrix}}$$

Note that RGB is not enough to disambiguate because R, G & B are correlated

Just provides better gradient

Lukas-Kanade flow

Prob: we have more equations than unknowns

$$\underset{25\times2}{A}\ \underset{2\times1}{d} = \underset{25\times1}{b} \longrightarrow \text{minimize } \|Ad - b\|^2$$

Solution: solve least squares problem

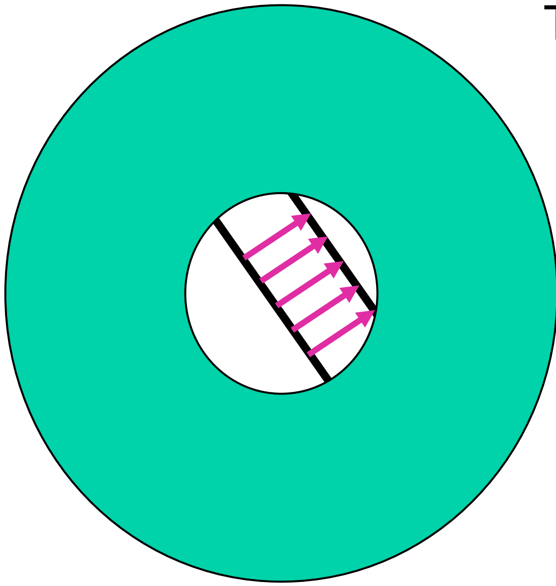
- minimum least squares solution given by solution (in d) of:

$$\underset{2\times2}{(A^T A)}\ \underset{2\times1}{d} = \underset{2\times1}{A^T b}$$

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

- The summations are over all pixels in the K x K window
- This technique was first proposed by Lukas & Kanade (1981)

Aperture Problem and Normal Flow

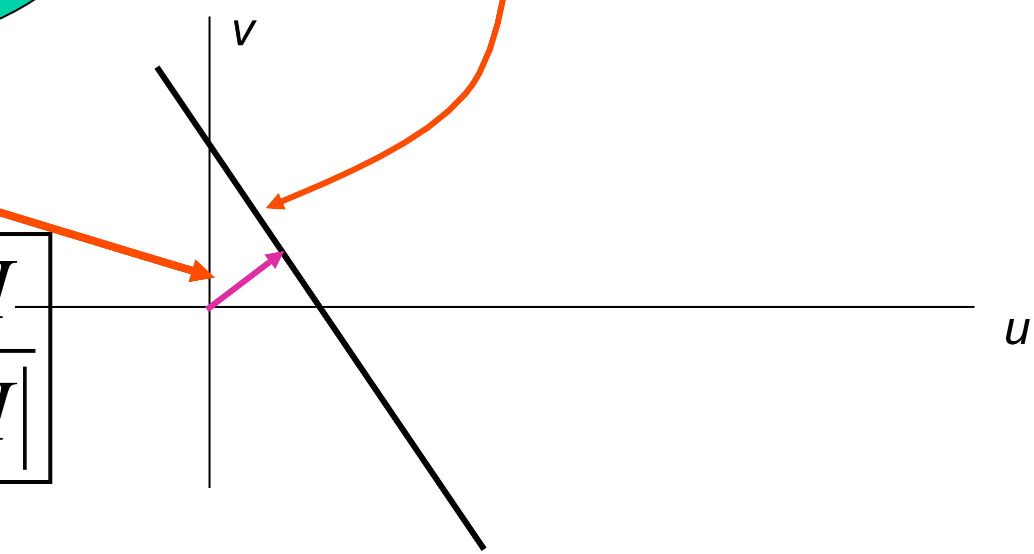


The gradient constraint:

$$I_x u + I_y v + I_t = 0$$

$$\nabla I \bullet \vec{U} = 0$$

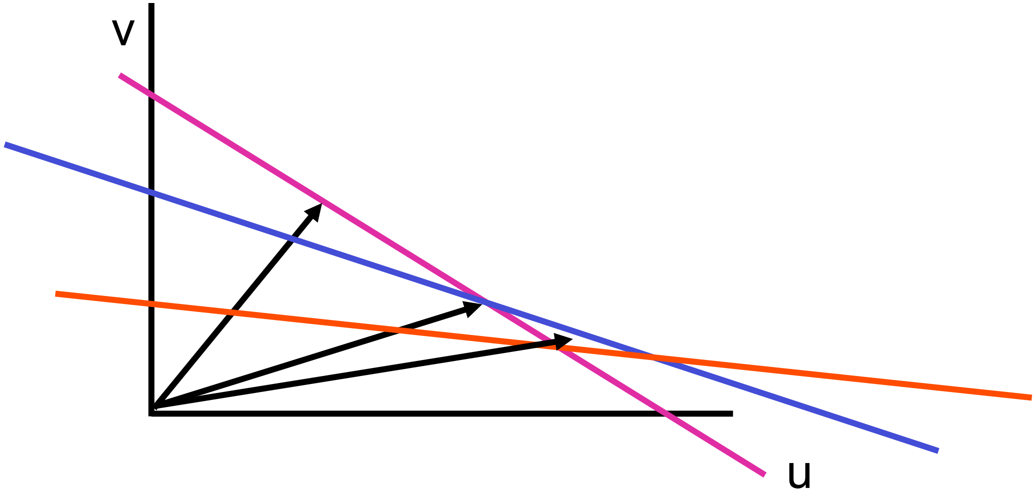
Defines a line in the *(u,v)* space



Normal Flow:

$$u_\perp = -\frac{I_t}{|\nabla I|}\frac{\nabla I}{|\nabla I|}$$

Combining Local Constraints



$$\nabla I^1 \bullet U = -I_t^1$$

$$\nabla I^2 \bullet U = -I_t^2$$

$$\nabla I^3 \bullet U = -I_t^3$$

etc.

Conditions for solvability

- Optimal (u, v) satisfies Lucas-Kanade equation

$$\underbrace{\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}}_{A^T A} \begin{bmatrix} u \\ v \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}}_{A^T b}$$

When is This Solvable?

- A^TA should be invertible
- A^TA should not be too small due to noise
  – eigenvalues λ_1 and λ_2 of A^TA should not be too small
- A^TA should be well-conditioned
  – λ_1 / λ_2 should not be too large (λ_1 = larger eigenvalue)

A^TA is solvable when there is no aperture problem

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y] = \sum \nabla I (\nabla I)^T$$

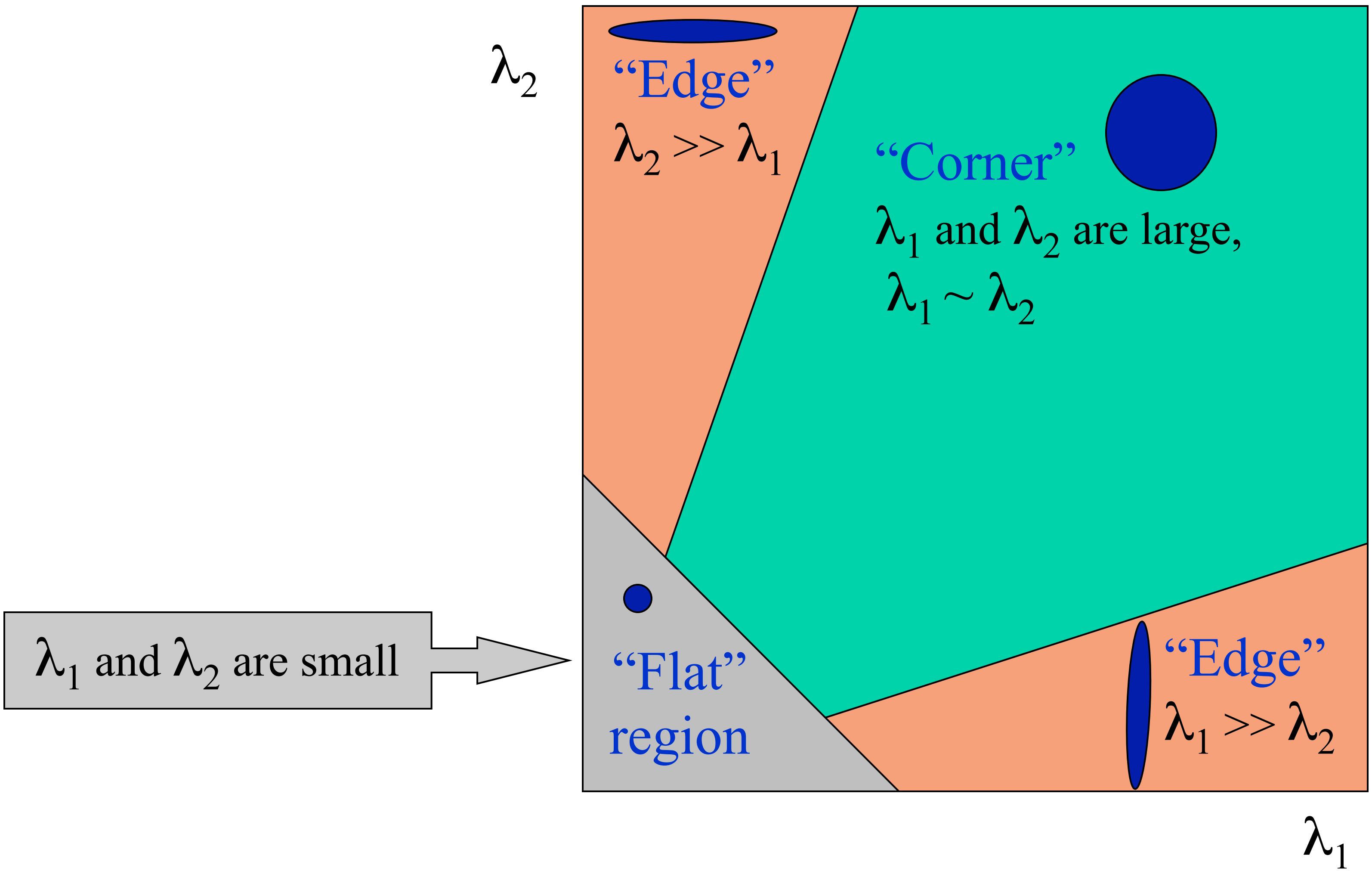
Eigenvectors of $A^T A$

$$A^T A = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T$$

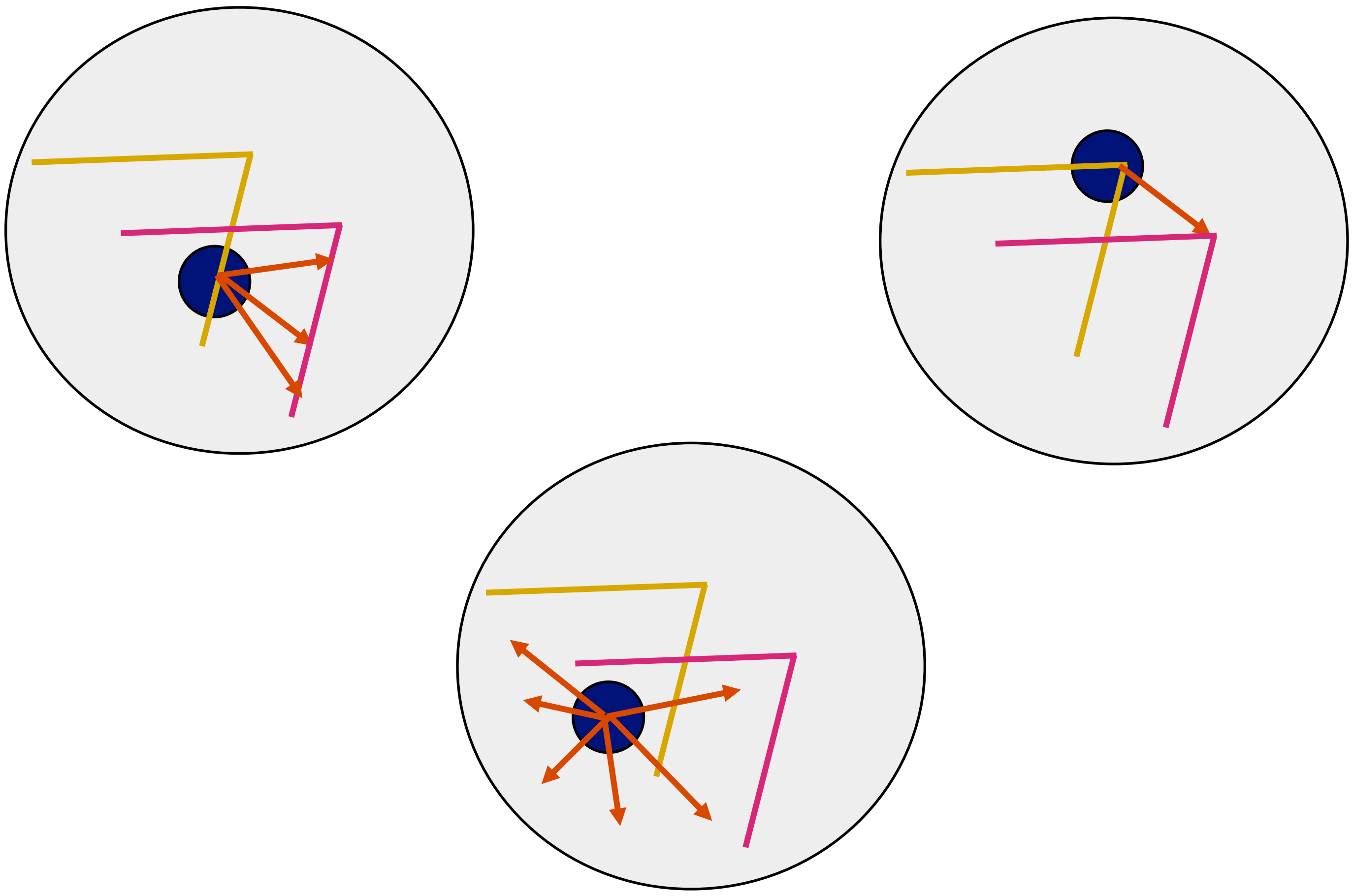
- Recall the Harris corner detector: $M = A^T A$ is the *second moment matrix*
- The eigenvectors and eigenvalues of M relate to edge direction and magnitude
  - The eigenvector associated with the larger eigenvalue points in the direction of fastest intensity change
  - The other eigenvector is orthogonal to it

Interpreting the eigenvalues

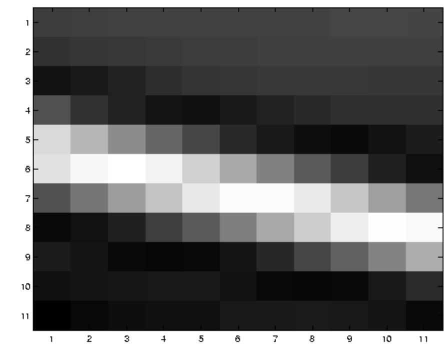
Classification of image points using eigenvalues of the second moment matrix:

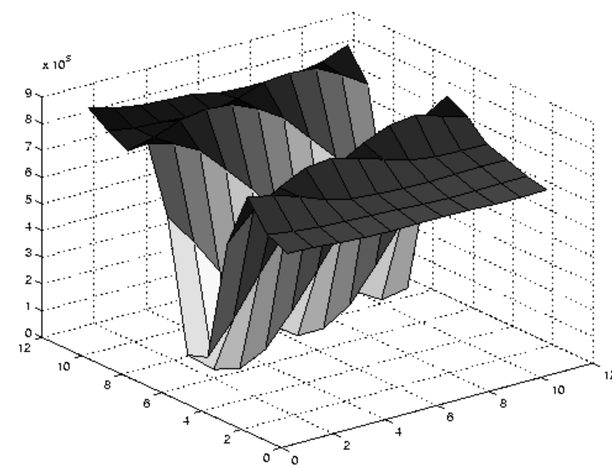


Local Patch Analysis



Edge



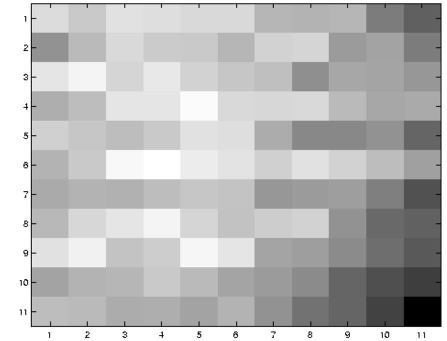


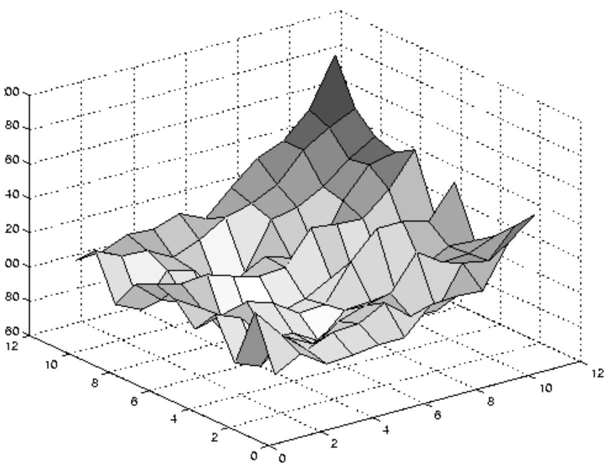
$$\sum \nabla I (\nabla I)^T$$

– large gradients, all the same

– large λ_1, small λ_2

Low texture region



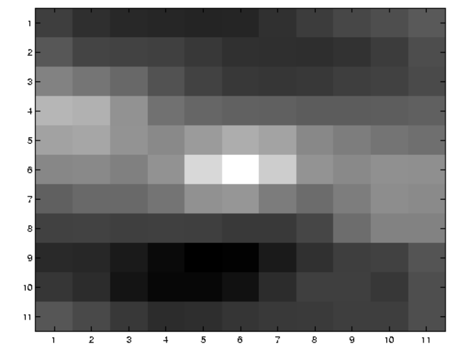


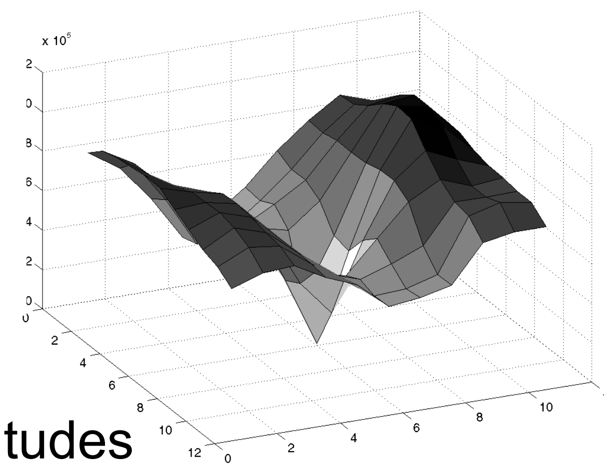
$$\sum \nabla I (\nabla I)^T$$

– gradients have small magnitude

– small λ_1, small λ_2

High textured region





$$\sum \nabla I (\nabla I)^T$$

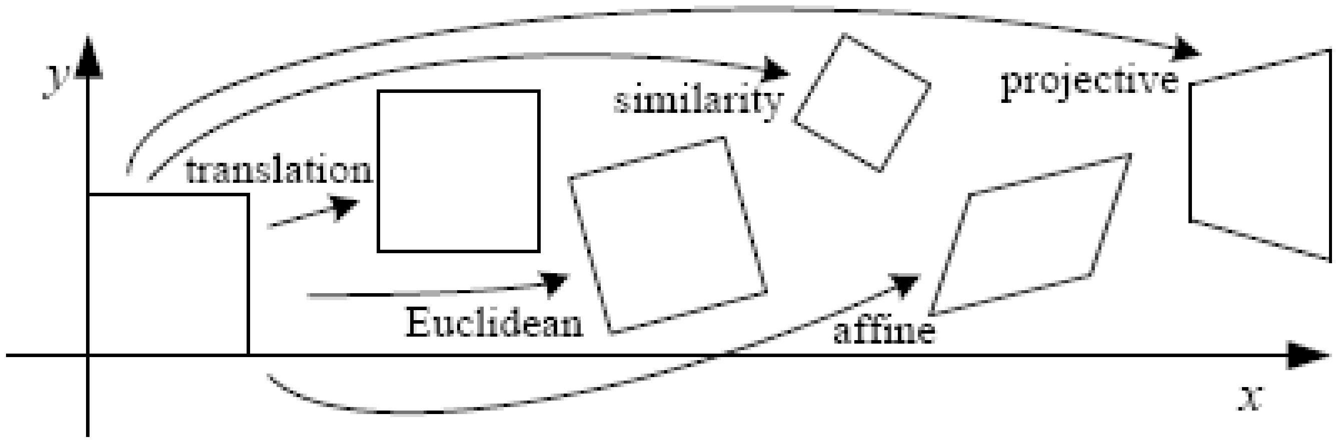
– gradients are different, large magnitudes
– large λ_1, large λ_2

Observation

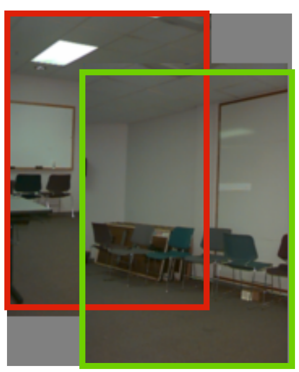
This is a two image problem BUT

- Can measure sensitivity by just looking at one of the images!
- This tells us which pixels are easy to track, which are hard
  - very useful later on when we do feature tracking...

Motion models

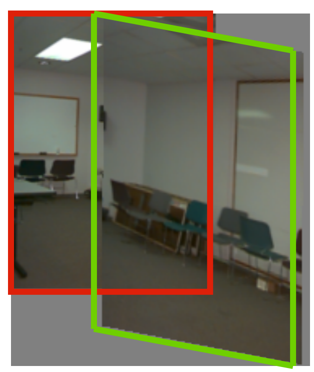


Translation



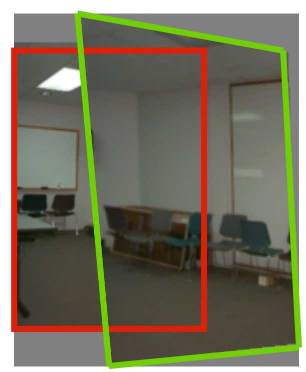
2 unknowns

Affine



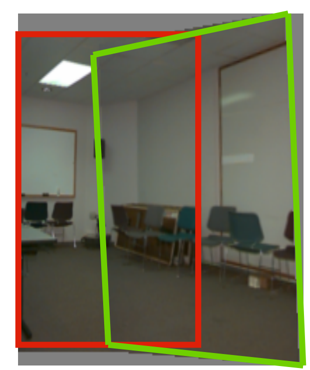
6 unknowns

Perspective



8 unknowns

3D rotation

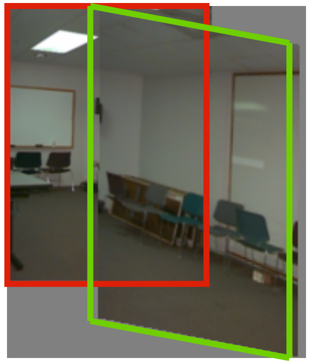


3 unknowns

Affine motion

$$u(x, y) = a_1 + a_2 x + a_3 y$$

$$v(x, y) = a_4 + a_5 x + a_6 y$$



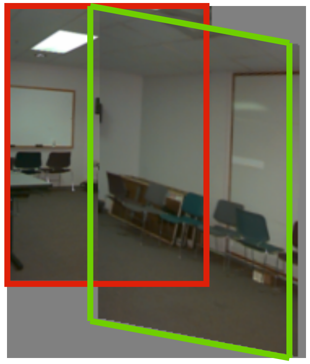
- Substituting into the brightness constancy equation:

$$I_x \cdot u + I_y \cdot v + I_t \approx 0$$

Affine motion

$$u(x,y) = a_1 + a_2 x + a_3 y$$

$$v(x,y) = a_4 + a_5 x + a_6 y$$



- Substituting into the brightness constancy equation:

$$I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \approx 0$$

- Each pixel provides 1 linear constraint in 6 unknowns

- Least squares minimization:

$$Err(\vec{a}) = \sum \left[I_x(a_1 + a_2 x + a_3 y) + I_y(a_4 + a_5 x + a_6 y) + I_t \right]^2$$

Errors in Lukas-Kanade

What are the potential causes of errors in this procedure?

- Suppose A^TA is easily invertible
- Suppose there is not much noise in the image

When our assumptions are violated

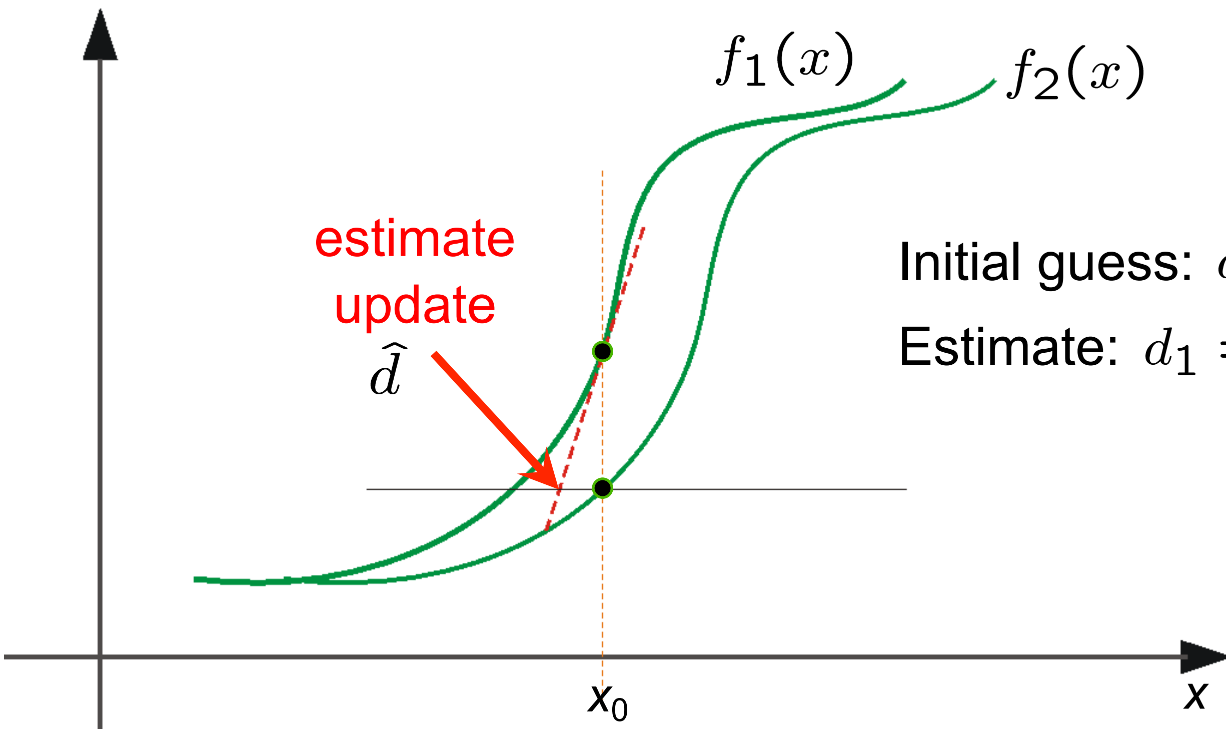
- Brightness constancy is not satisfied
- The motion is not small
- A point does not move like its neighbors
  - window size is too large
  - what is the ideal window size?

Iterative Refinement

Iterative Lukas-Kanade Algorithm

1. Estimate velocity at each pixel by solving Lucas-Kanade equations
2. Warp H towards I using the estimated flow field
   - use image warping techniques
3. Repeat until convergence

Optical Flow: Iterative Estimation

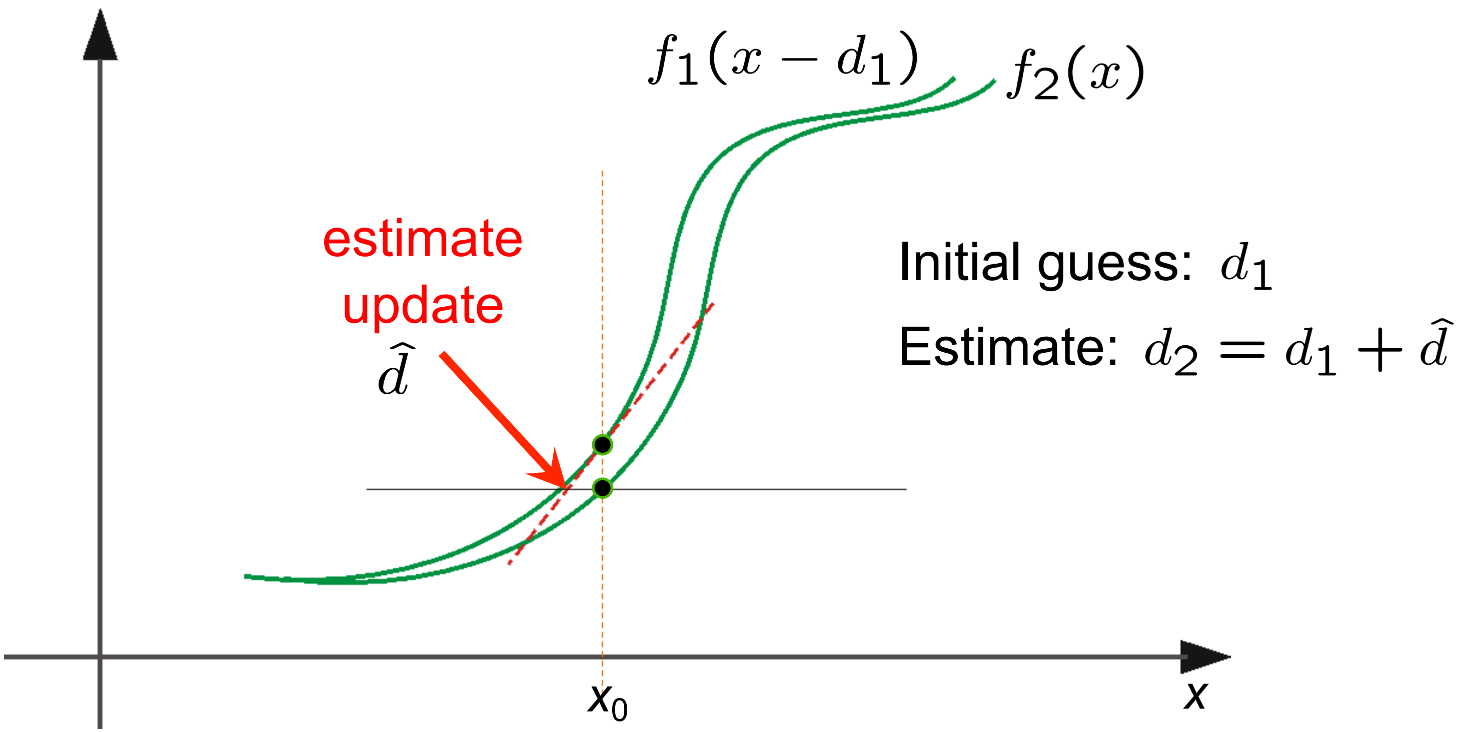


Initial guess: $d_0 = 0$

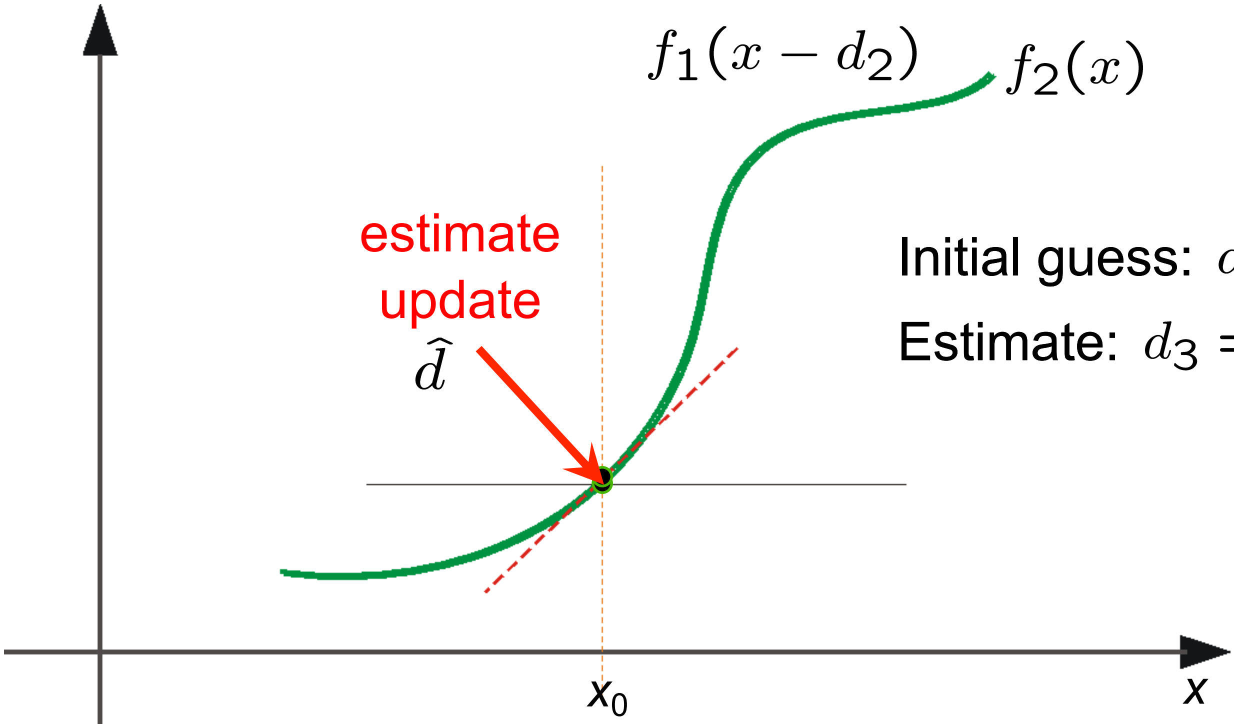
Estimate: $d_1 = d_0 + \hat{d}$

(using *d* for *displacement* here instead of *u*)

Optical Flow: Iterative Estimation



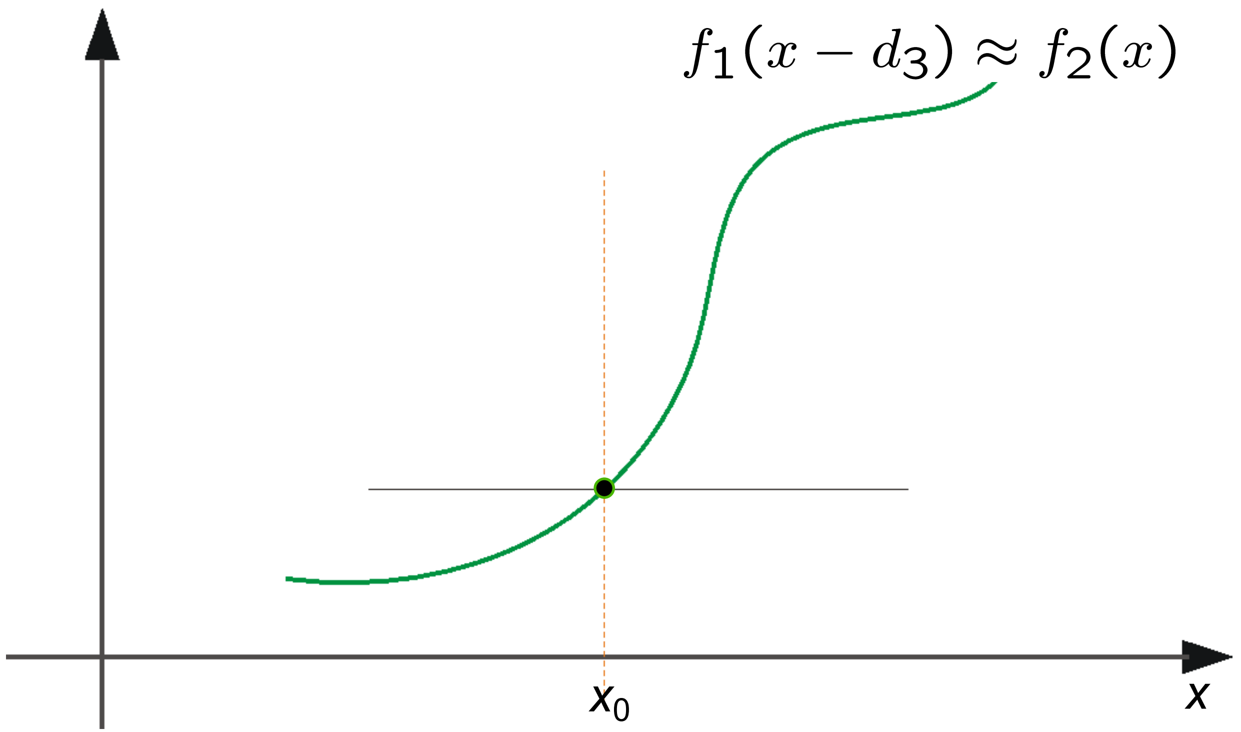
Optical Flow: Iterative Estimation



Initial guess: d_2

Estimate: $d_3 = d_2 + \hat{d}$

Optical Flow: Iterative Estimation



Optical Flow: Iterative Estimation

Some Implementation Issues:

- Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)
- Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration.
- Often useful to low-pass filter the images before motion estimation (for better derivative estimation, and linear approximations to image intensity)

Revisiting the small motion assumption

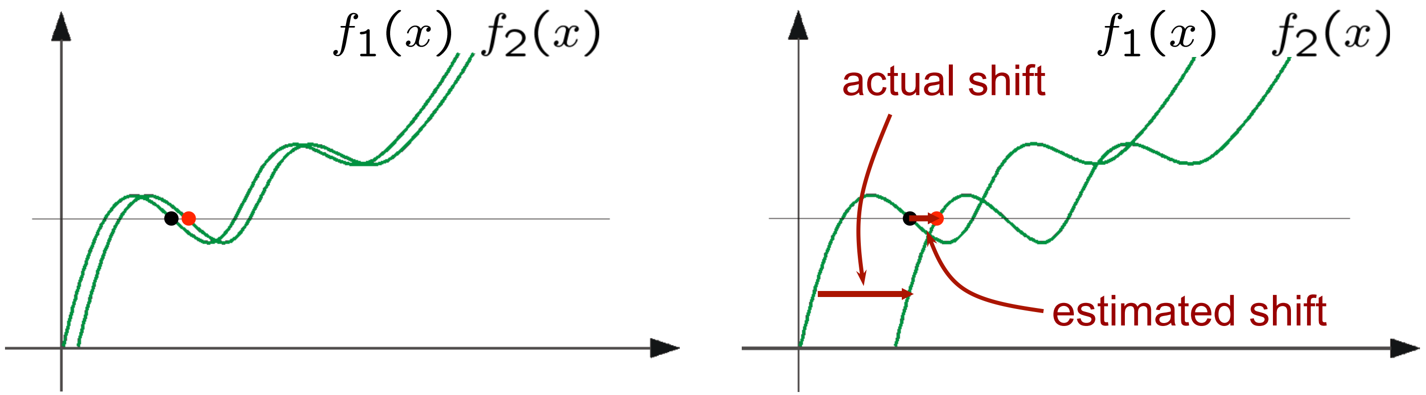
Is this motion small enough?

- Probably not—it's much larger than one pixel (2^{nd} order terms dominate)
- How might we solve this problem?

Optical Flow: Aliasing

Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity.

I.e., how do we know which 'correspondence' is correct?

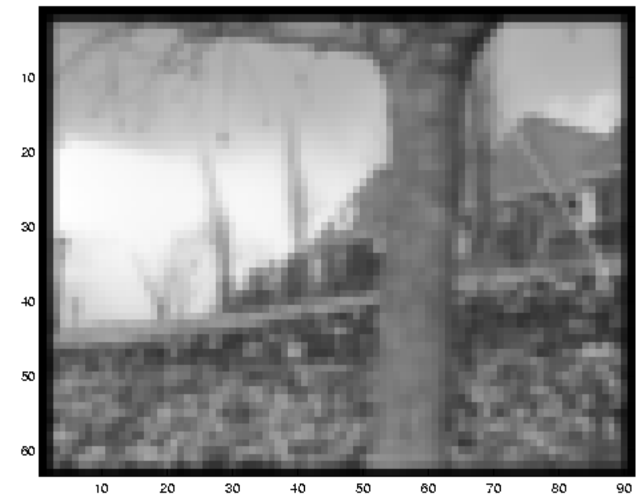


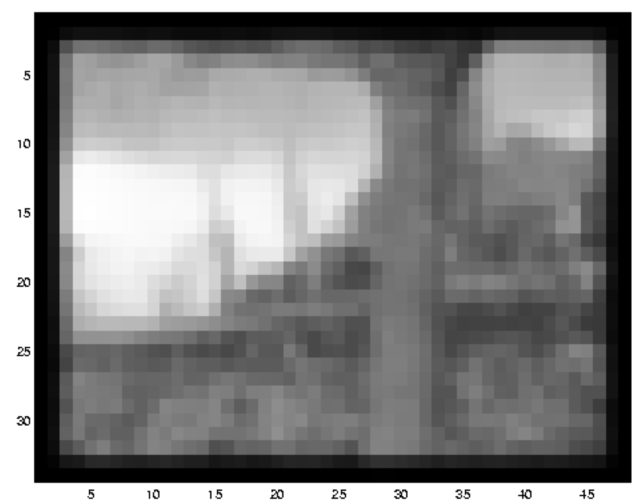
nearest match is correct (no aliasing)

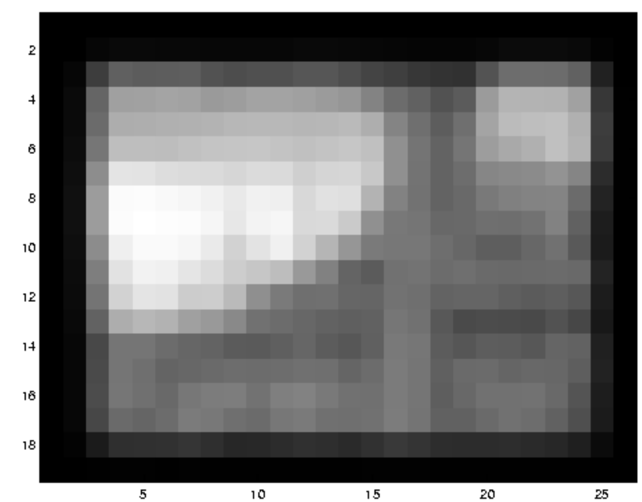
nearest match is incorrect (aliasing)

To overcome aliasing: coarse-to-fine estimation.

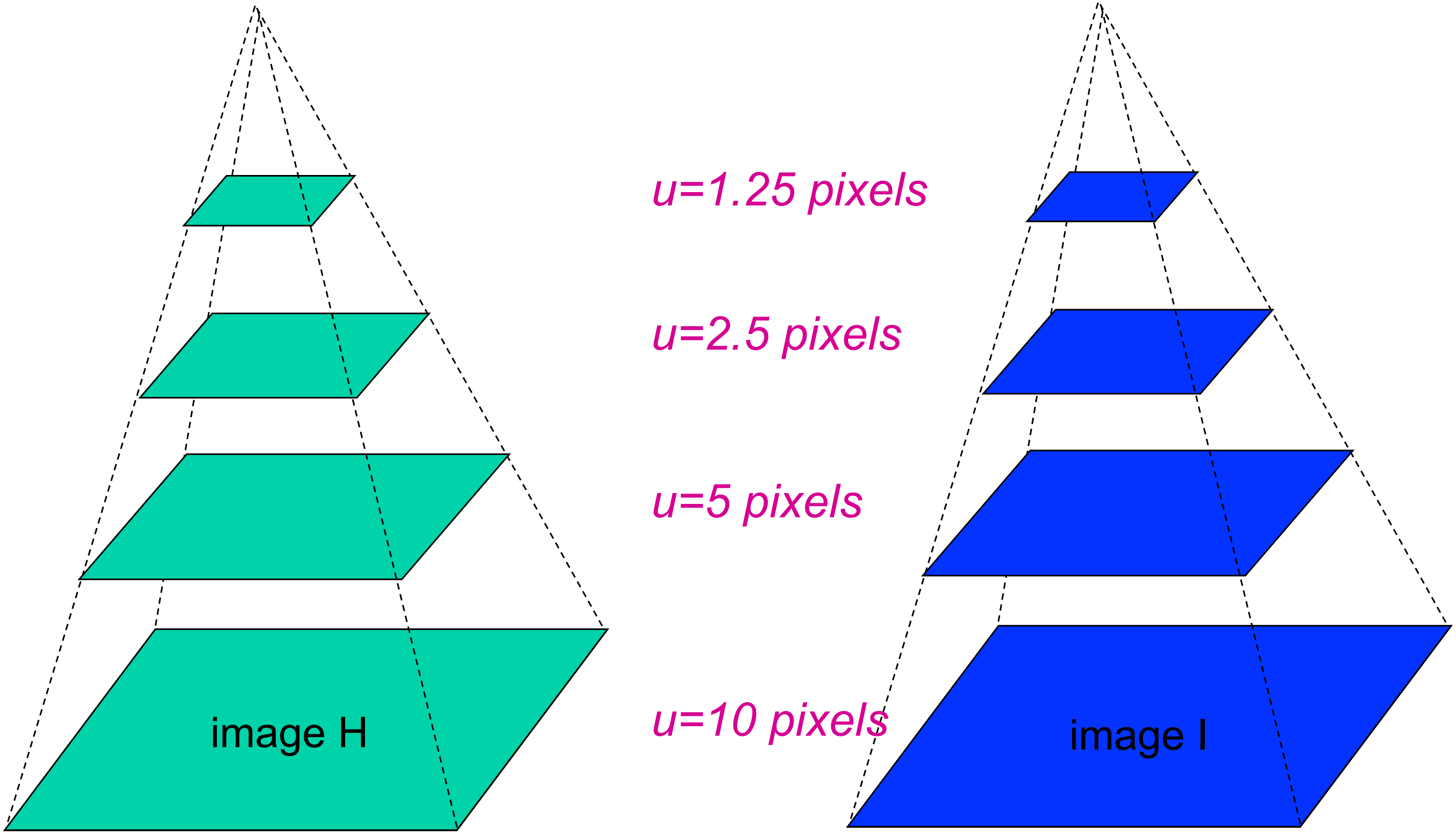
Reduce the resolution!







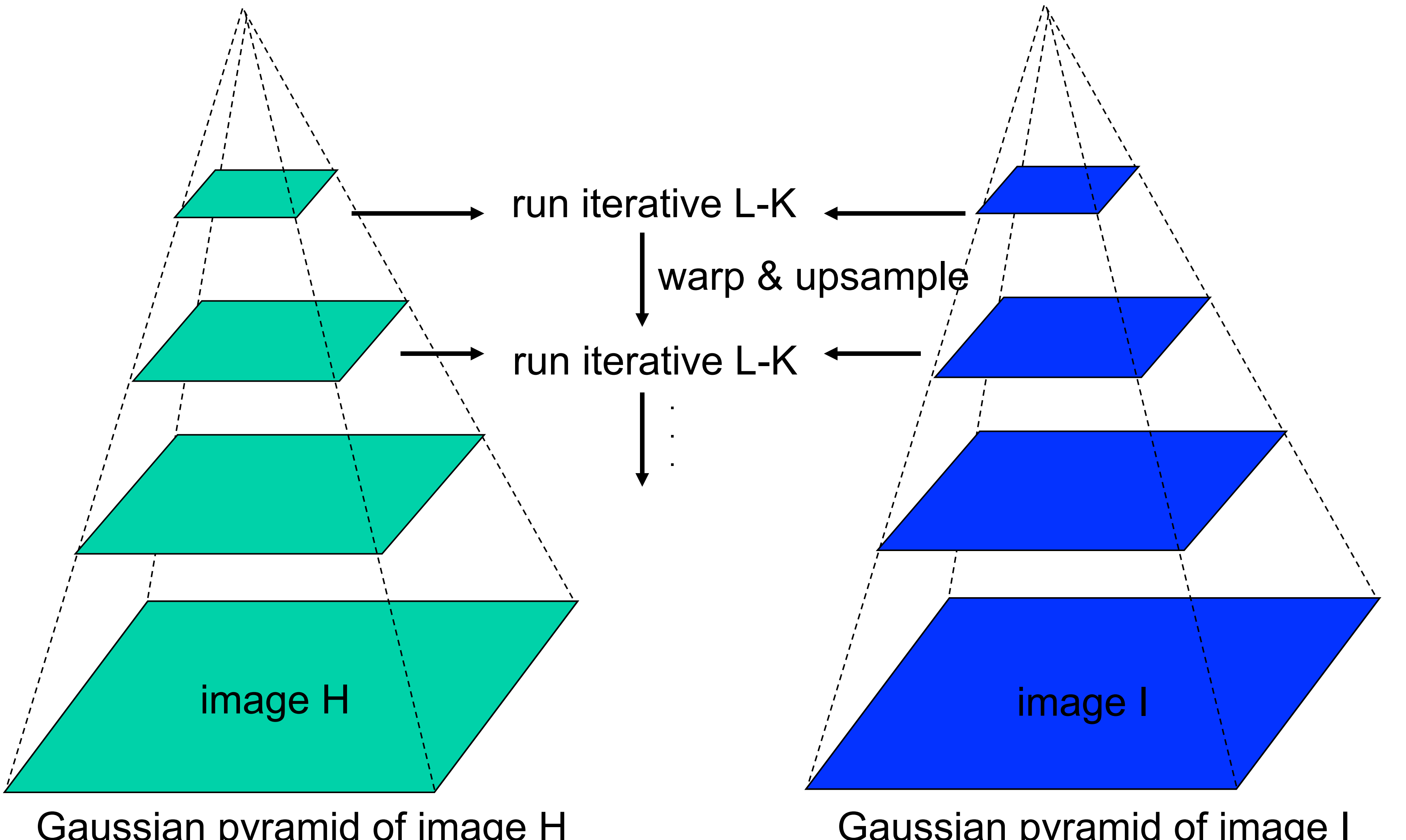
Coarse-to-fine optical flow estimation



Gaussian pyramid of image H

Gaussian pyramid of image I

Coarse-to-fine optical flow estimation



Recap: Classes of Techniques

Feature-based methods (e.g. SIFT+Ransac+regression)

- Extract visual features (corners, textured areas) and track them over multiple frames
- Sparse motion fields, but possibly robust tracking
- Suitable especially when image motion is large (10-s of pixels)

Direct-methods (e.g. optical flow)

- Directly recover image motion from spatio-temporal image brightness variations
- Global motion parameters directly recovered without an intermediate feature motion calculation
- Dense motion fields, but more sensitive to appearance variations
- Suitable for video and when image motion is small (< 10 pixels)