SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

RIAS Robotics & Intelligent Autonomous Systems

Introduction to Mobile Robotics

Jeff McGough

Department of Computer Science
South Dakota School of Mines and Technology
Rapid City, SD 57701, USA

October 1, 2012

Feature Detectors

Image Features

- Edges
- Corners
- Regions of interest (aka blobs)
- Ridges

Many Feature Detectors,

- Harris Detector (1988)
- Sobel
- SIFT (2004)
- Laplacian of Gaussians

Image Processing

A brief detour on some common image processing techniques ...

- Smoothing
- Convolutions
- Gradient
- Edge detection
- Canny

Convolution

Recall from basic math - the convolution

$$f * g \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

One can smooth a function, say $f(t)$ by convolving it with a smoothing function $g(t)$. This can be done in space as well.

Note that

$$\frac{d}{dt}(f * g) = \frac{d}{dt}\int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(\tau)\frac{d}{dt}g(t-\tau)d\tau$$

$$\frac{d}{dt}(f * g) = \frac{df}{dt} * g = f * \frac{dg}{dt}$$

Images

We will work with discrete functions and discrete data....

An image is thought of as a function $I = f(x, y)$ where f returns intensity at the point (x, y). For gray scale it returns a single variable, for color images there are three values per point (RGB).

Pixels are a basic 2D discretization idea. $x = i\Delta x$, $y = j\Delta y$, then

$$I = f(x, y) \rightarrow f(i\Delta x, j\Delta y) \rightarrow \hat{f}_{i,j}$$

$$= \begin{bmatrix} 37 & 40 & 40 & \dots \\ 41 & 43 & 45 & \dots \\ 46 & 50 & 60 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} = I$$

Smoothing

Gaussian Smoothing or Blurring

- Removes high frequency noise
- Discrete Convolution with Gaussian distributions:

$$G \approx ke^{-(x-\mu_1)^2/\sigma_1^2-(y-\mu_2)^2/\sigma_2^2}$$

- Operate on intensity image: $\hat{I} = G \otimes I$ where

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Stencil is applied at each point of the image

Smoothing

Application to images

$$I = \begin{bmatrix} 15 & 13 & 10 & 9 \\ 10 & 8 & 7 & 5 \\ 9 & 4 & 2 & 1 \\ 5 & 2 & 1 & 0 \end{bmatrix} \qquad G = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Flip Mask
- Multiply mask values by image/pixel values
- Sum up results
- Write result into target

Smoothing

Application to images

$$I = \begin{bmatrix} 15 & 13 & 10 & 9 \\ 10 & 8 & 7 & 5 \\ 9 & 4 & 2 & 1 \\ 5 & 2 & 1 & 0 \end{bmatrix} \qquad G' = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

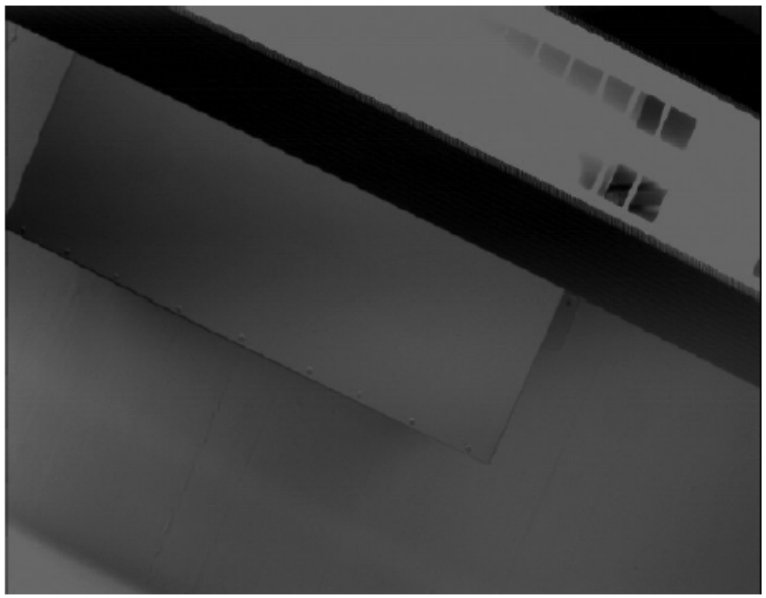
$$[I] * G' = \frac{1}{16} \begin{bmatrix} 1*15 & 2*13 & 1*10 \\ 2*10 & 4*8 & 2*7 \\ 1*9 & 2*4 & 1*2 \end{bmatrix}$$

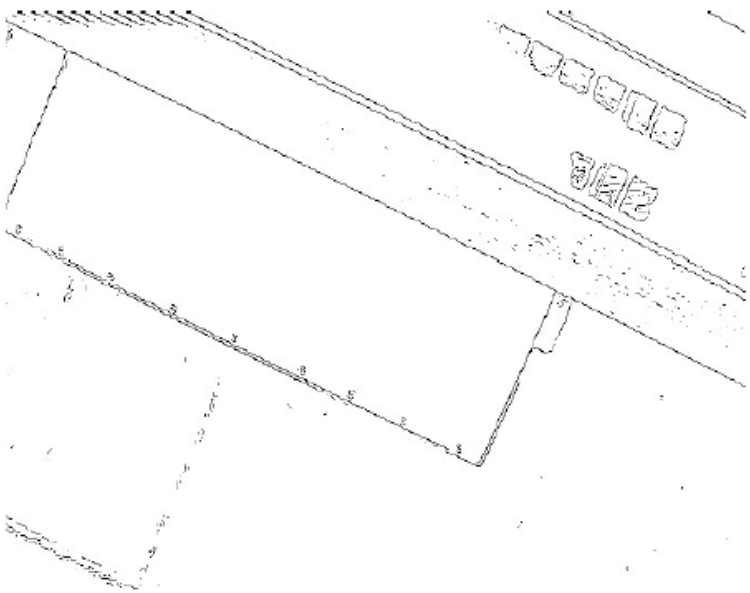
$$I_{2,2} = \frac{1}{16}(1*15+2*13+1*10+2*10+4*8+2*7+1*9+2*4+1*2)$$

Edge detection

Edges

- Locations where there is a large change in brightness
- Ultimate goal of edge detection an idealized line drawing.
- Edge contours in the image correspond to important scene contours.





Edge detection

Edges

- Locations where there is a large change in brightness or intensity
- Change measured using derivatives (by first derivative)
- Differentiate the image once or twice: $\hat{I} = G \otimes I$

$$G = \frac{1}{16}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

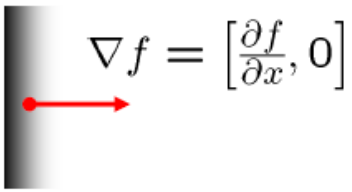
- Look for places where the magnitude of the derivative is large
- Noise is a problem, smoothing is required.

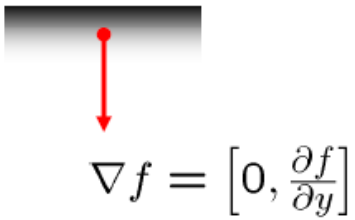
Image Gradient

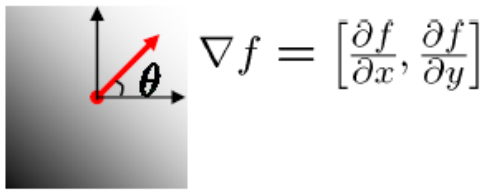
- Gradient of f:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- Gradient points in the direction of greatest change (of intensity).







- The gradient direction is

$$\theta = \arctan\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- The *edge strength* is given by the magnitude

$$\|f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Image Gradient

How does one take a derivative of a digital (discrete) image?

1. Reconstruct continuous image and take a gradient.
2. Finite differences

Recall:

$$\frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

so in our case ...

$$\frac{\partial f}{\partial x} \approx f(x+1, y) - f(x, y)$$

Gradient operators

- **Roberts**

$$|G| \cong \sqrt{r_1^2 + r_2^2} \; ; \quad r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \; ; \quad r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

- **Prewitt**

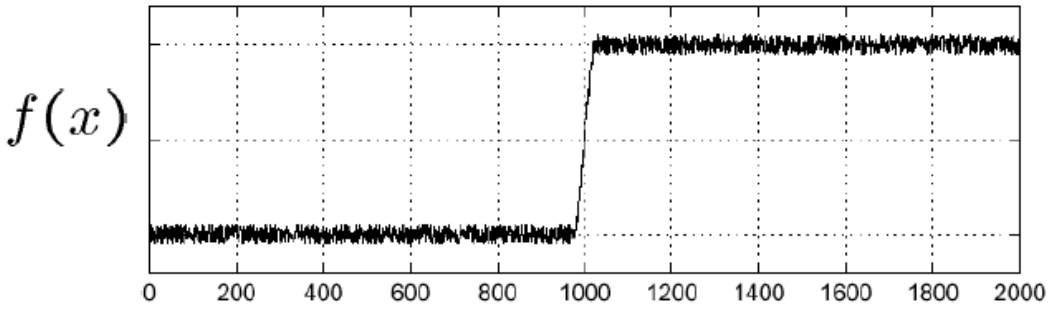
$$|G| \cong \sqrt{p_1^2 + p_2^2} \; ; \quad \theta \cong \operatorname{atan}\left(\frac{p_1}{p_2}\right) \; ; \quad p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \; ; \quad p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

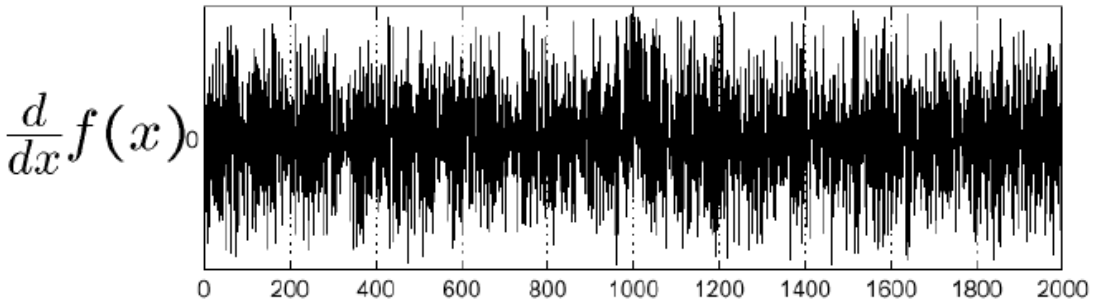
- **Sobel**

$$|G| \cong \sqrt{s_1^2 + s_2^2} \; ; \quad \theta \cong \operatorname{atan}\left(\frac{s_1}{s_2}\right) \; ; \quad s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \; ; \quad s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal





- Where is the edge?

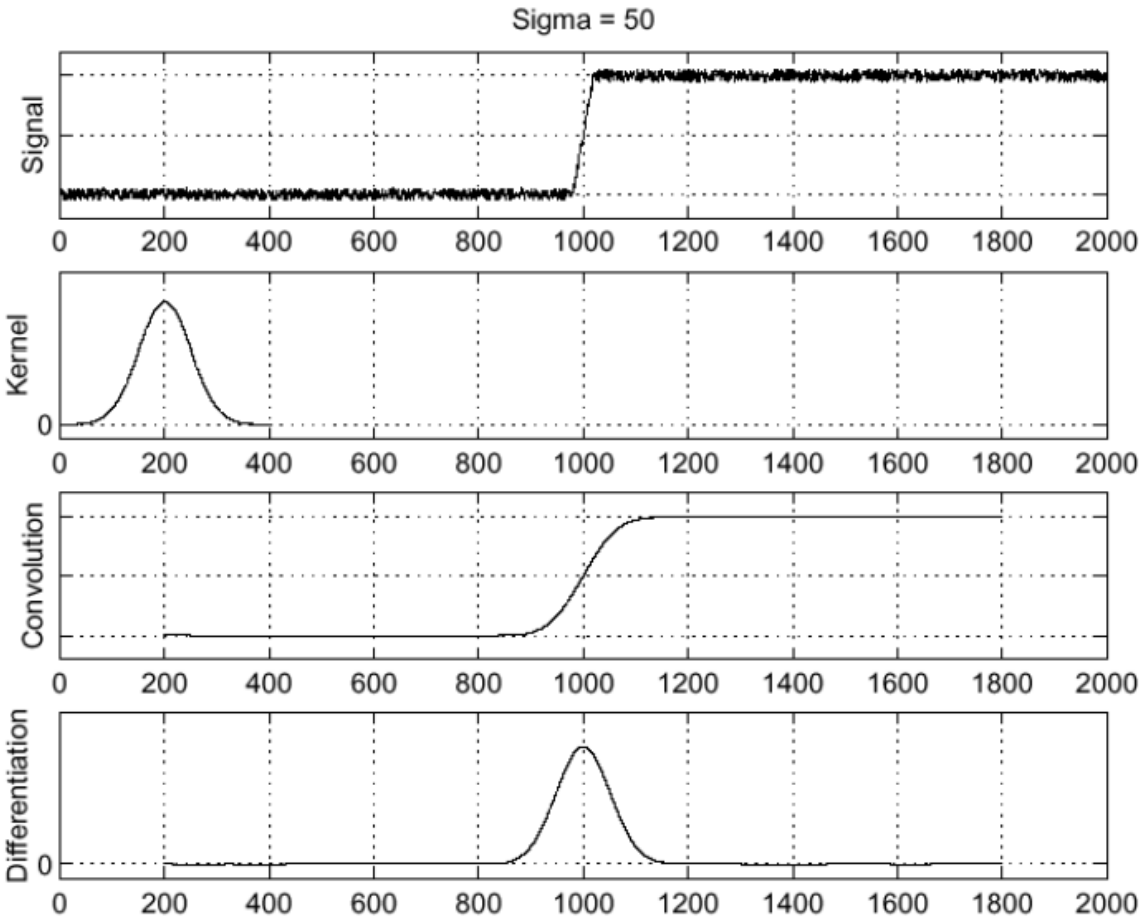
Noise

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Convolution and Differentiation

If you have to apply both smoothing and differentiation,

then you can apply the derivative to the smoothing operator since it appears in the convolution.

$$\frac{d}{dx}(f * h) = \frac{d}{dx}\int_0^L f(\xi)h(x-\xi)d\xi = \int_0^L f(\xi)\frac{d}{dx}h(x-\xi)d\xi$$

Noise

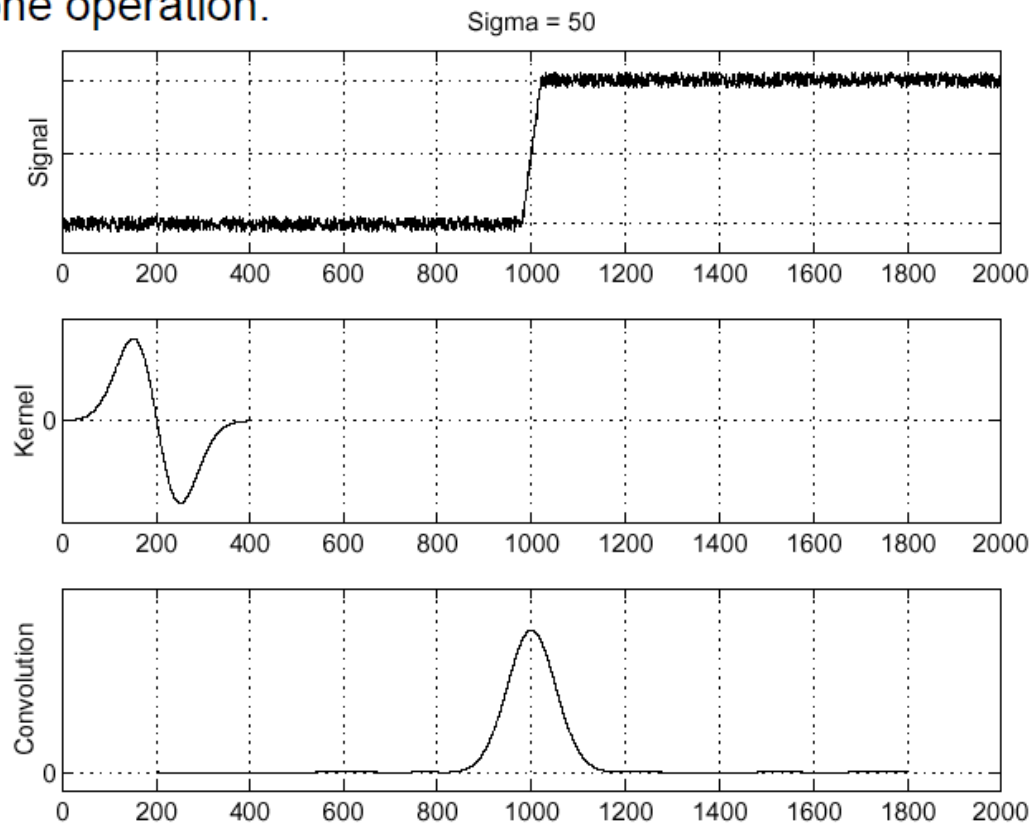
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

f

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



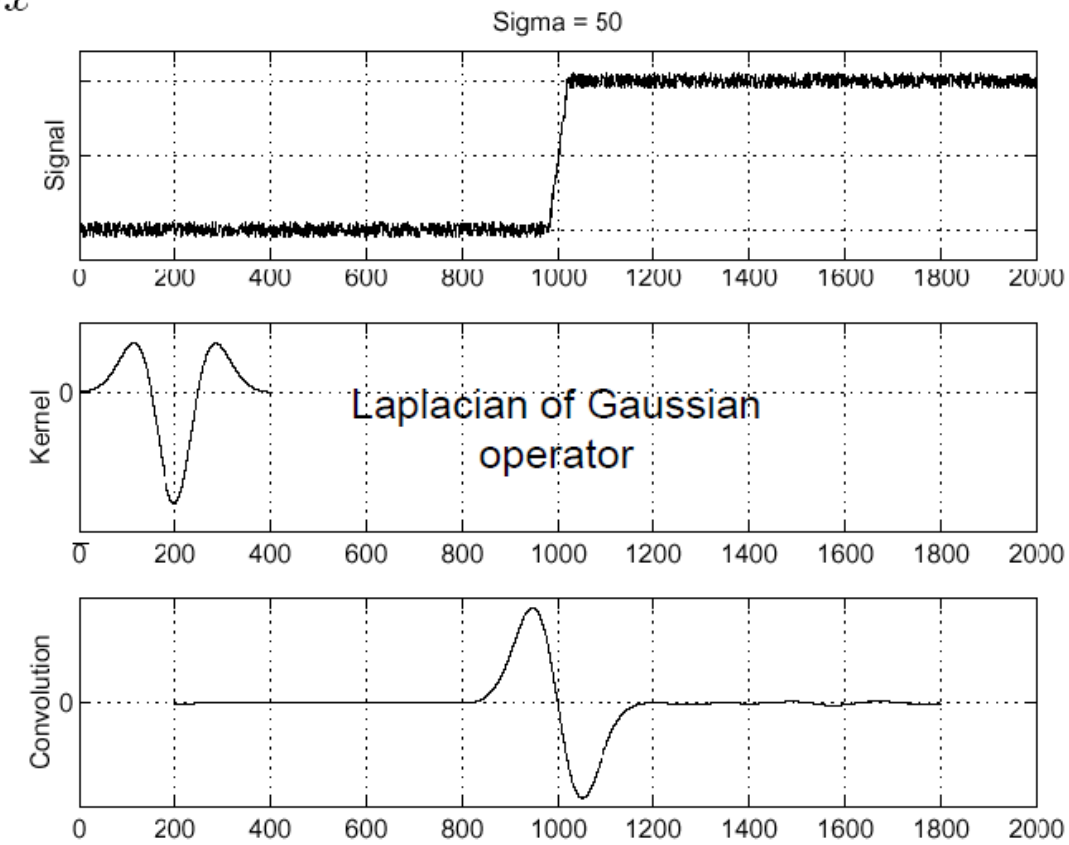
Canny Edge Detection

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$

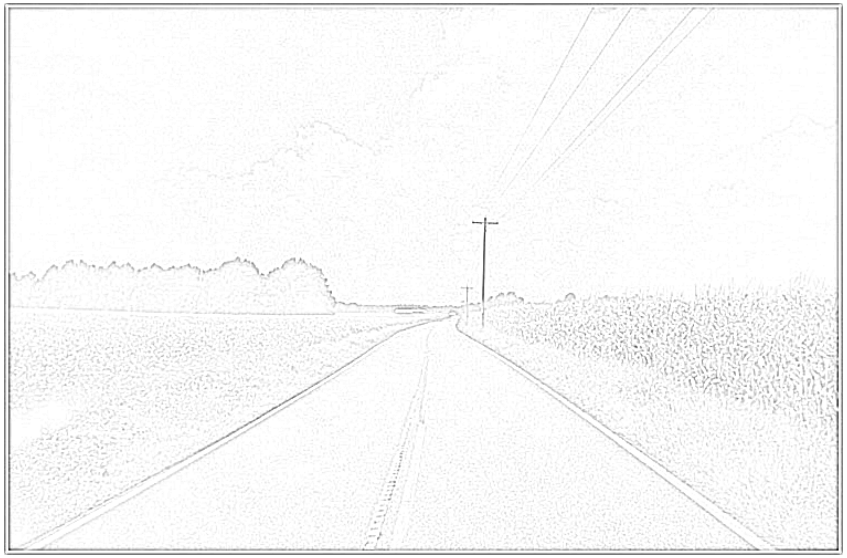


- Where is the edge?
- Zero-crossings of bottom graph

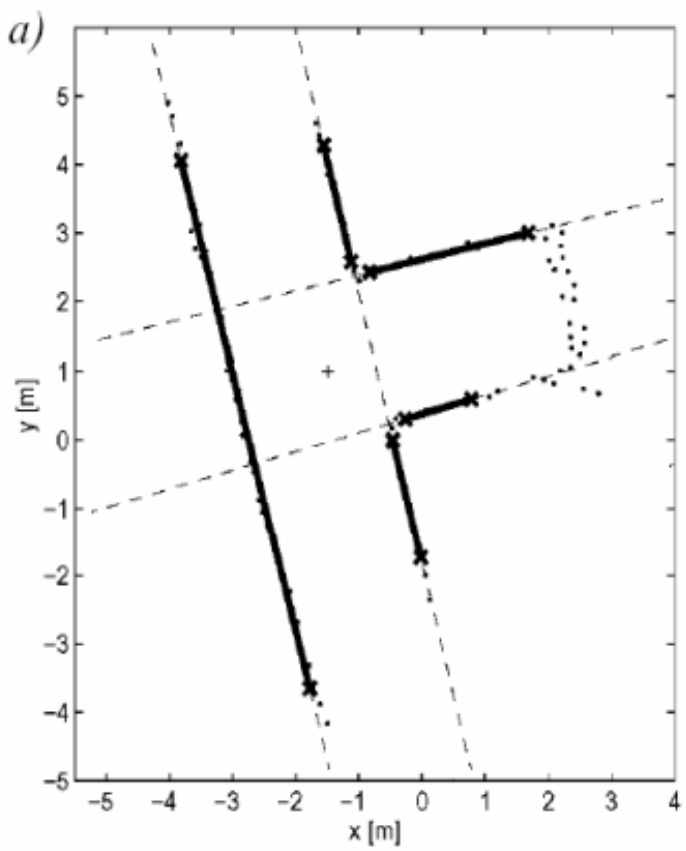
Edge Detection

Line extraction: Use Hough, RANSAC or Split-Merge to determine lines.

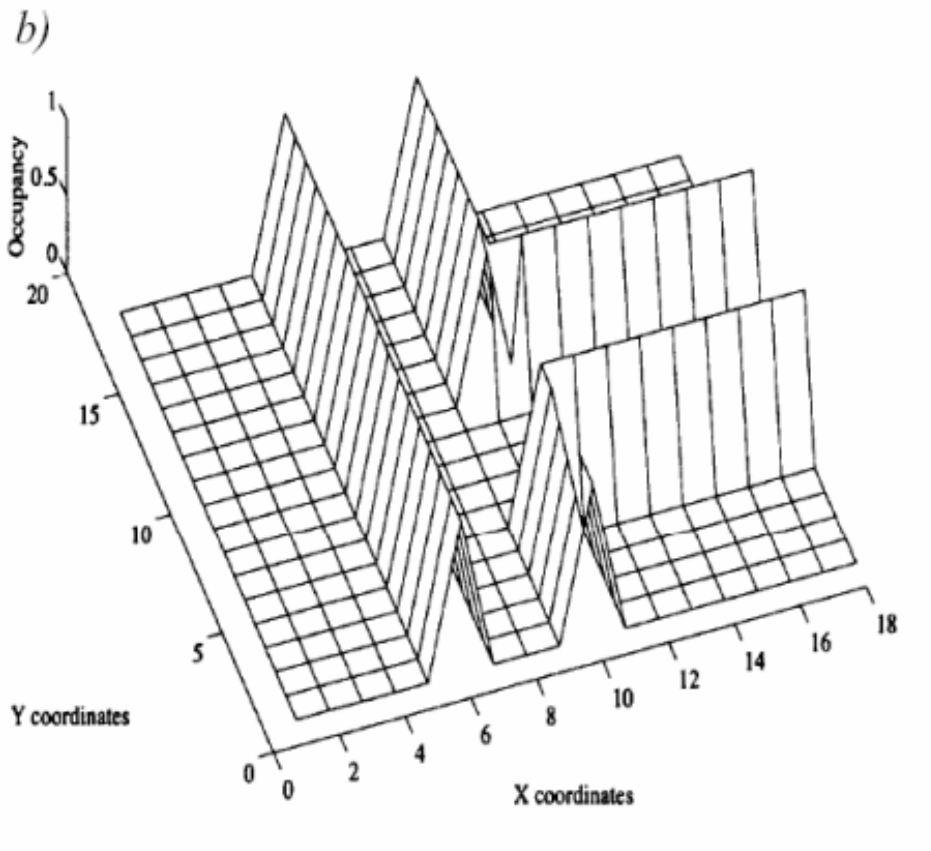
Edge Detection: Application



Environment Models

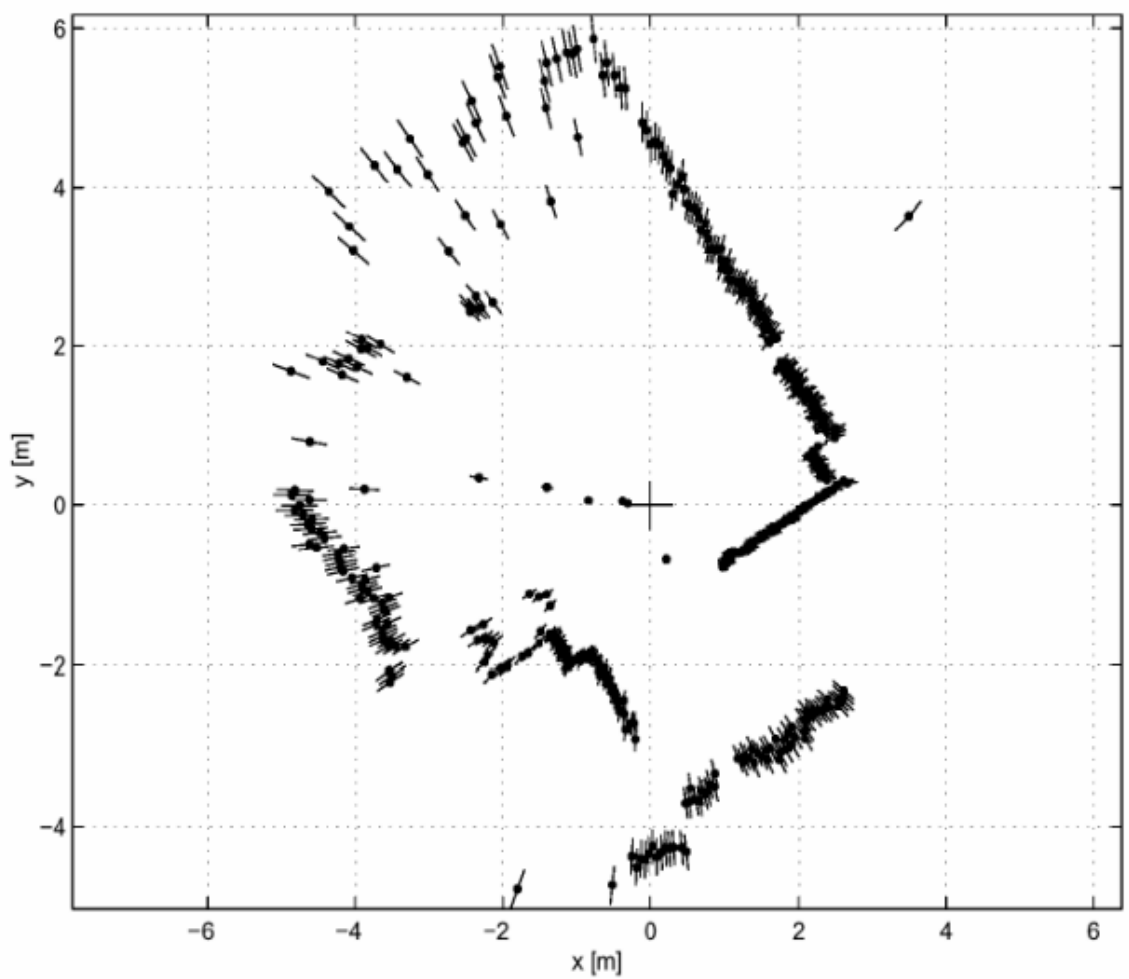


A: Feature base Model



B: Occupancy Grid

Lidar scan of a room

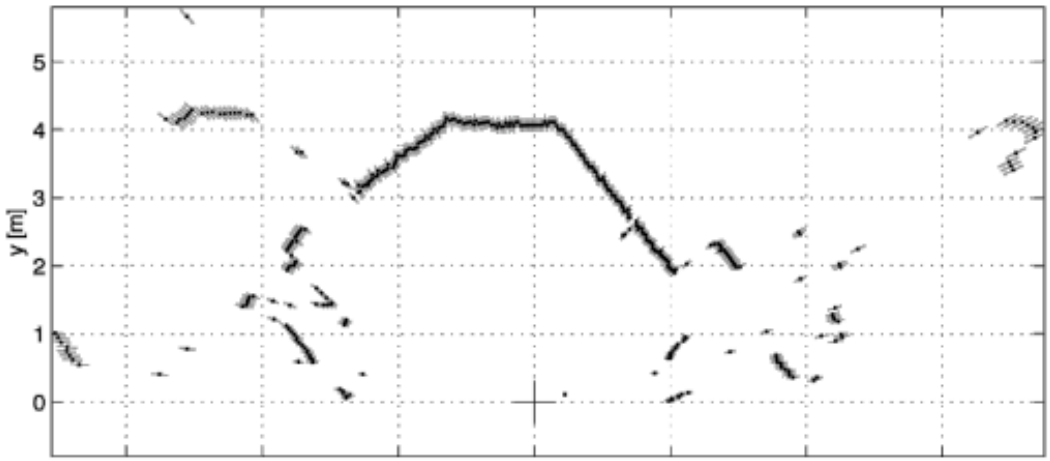


Feature extraction based on range data

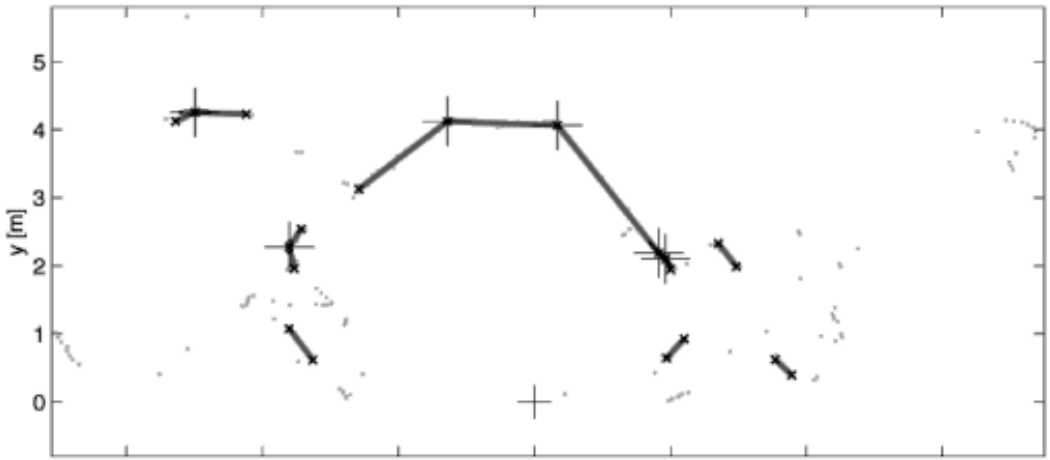
- Geometric primitives like line segments, circles, corners, edges
- Most other geometric primitives the parametric description of the features becomes already to complex and no closed form solutions exist.
- However, lines segments are very often sufficient to model the environment, especially for indoor applications.

Line Extraction

Raw
range data

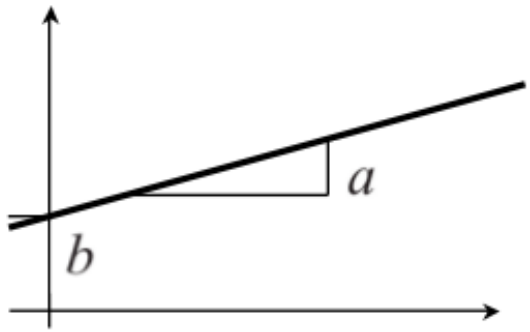


Line
segments



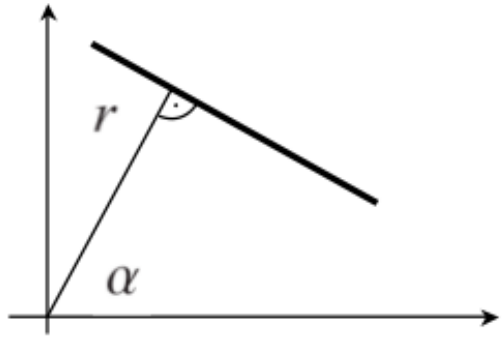
Line Extraction

Choosing the adequate line model



Intercept-Slope

$$y = a\,x + b$$



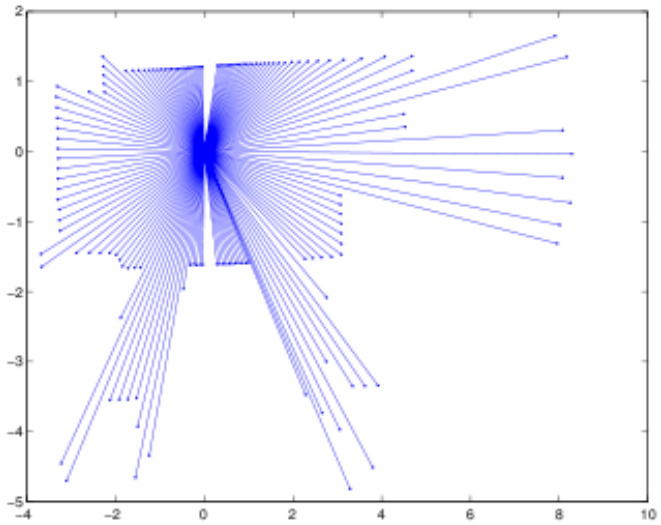
Hessian model

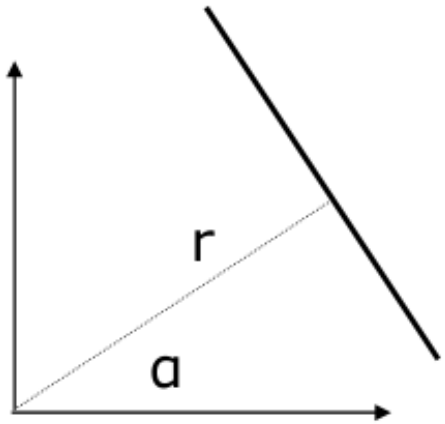
$$x\cos\alpha + y\sin\alpha - r = 0$$

Each model has advantages and drawbacks

Line Extraction

- Scan point in polar form: (ρ_i, θ_i)
- Assumptions:
  - Gaussian noise with $(0, \sigma)$ for ρ
  - Negligible angular uncertainty
- Line model in polar form:
  - $x \cos \alpha + y \sin \alpha = r$
  - $-\pi < \alpha <= \pi$
  - $r >= 0$



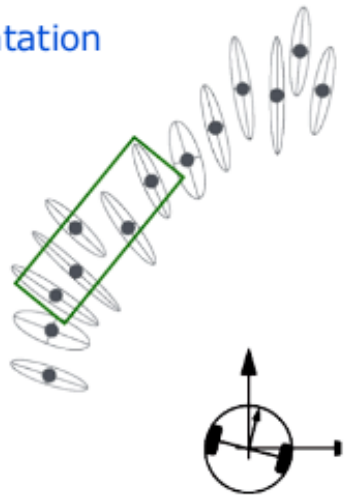


Line Extraction ...

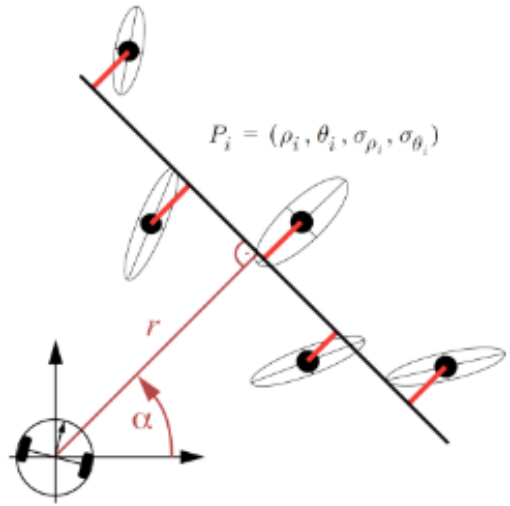
Can be subdivided into two subproblems:

- **Segmentation:** *Which* points contribute?
- **Fitting:** *How* do the points contribute?

Segmentation



Fitting



Issues

Main problems in line extraction

- How many lines?
- Which points belong to which line?
- Given points that belong to a line, how to estimate line parameters?

The methods

- Split and Merge
- Linear Regression
- Clustering
- Incremental
- RANSAC
- Hough Transform
- Expectation maximization

Split and Merge

- The most popular algorithm which arose in computer vision.
- A recursive algorithm of splitting and fitting.
- Related to Iterative-End-Point-Fit.

Split

- Obtain the line passing through the two extreme points
- Find the most distant point to the line
- If distance $>$ threshold, split and repeat with left and right point sets

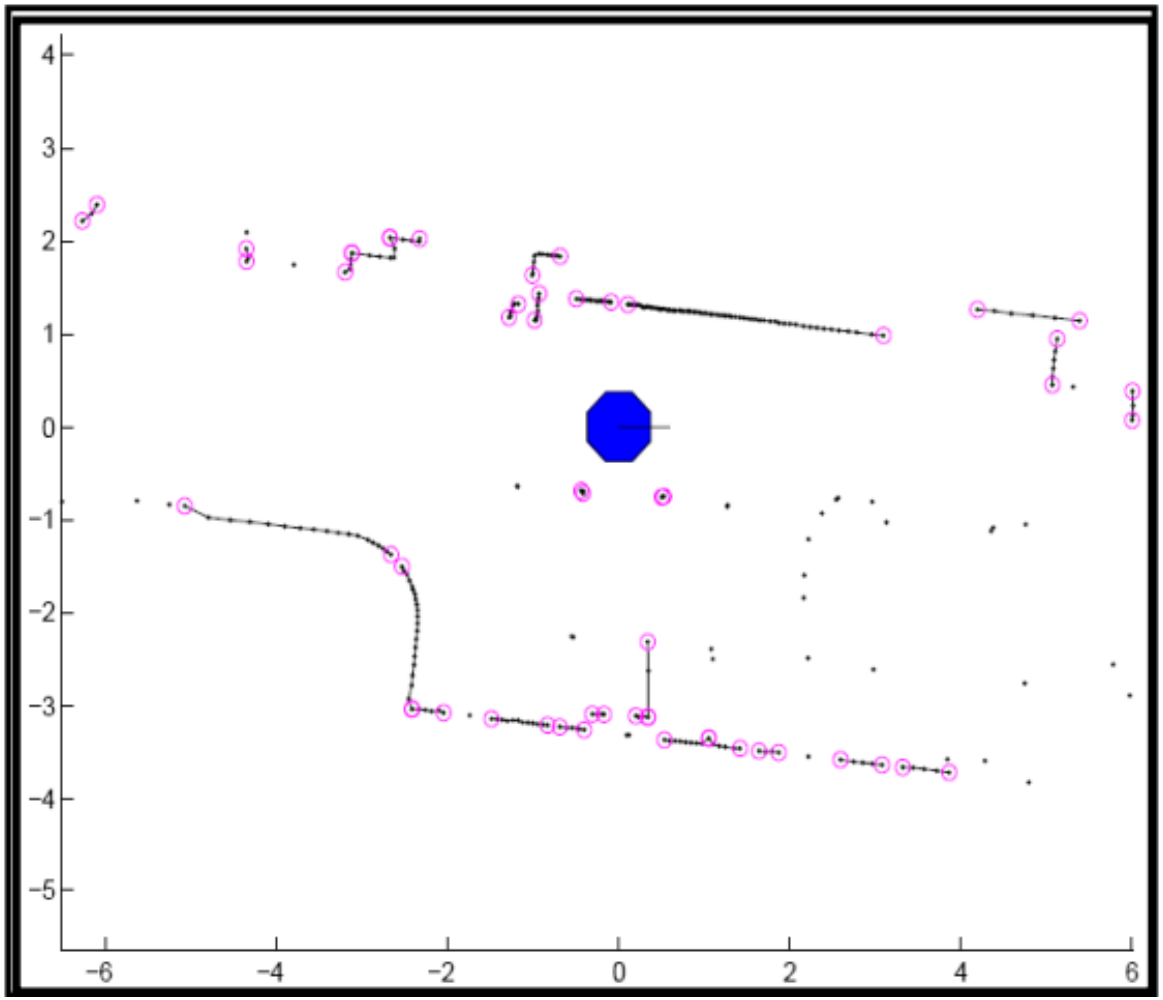
Merge

- If two consecutive segments are close (collinear) enough, obtain the common line and find the most distant point
- If distance $\leq$ threshold, merge both segments

Split and Merge Algorithm

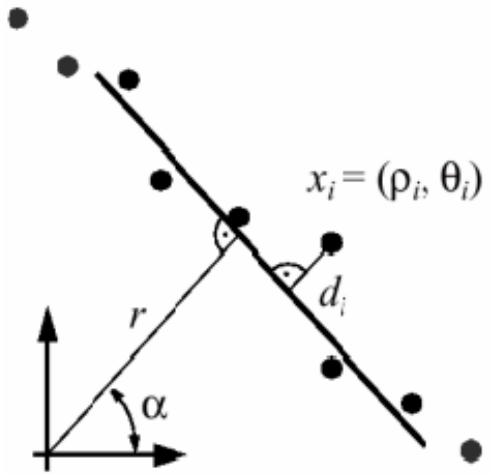
1. Initial: set s_1 consists of N points. Put s_1 in a list $\mathcal{L}$.
2. Fit a line to the next set s_i in $\mathcal{L}$.
3. Detect point P with maximum distance d_P to the line.
4. If d_P is less than a threshold, continue (step 2)
5. Otherwise, split s_i at P into s_{i1} and s_{i2}, replace s_i in $\mathcal{L}$ by s_{i1} and s_{i2} and continue (step 2).
6. When all sets (segments) in $\mathcal{L}$ have been checked, merge collinear segments.

Line Extraction ...



Least squares

$$\rho_i \cos(\theta_i - \alpha) - r = d_i$$



- Least Square

$$S = \sum_i d_i^2 = \sum_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

- Weighted Least Square

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

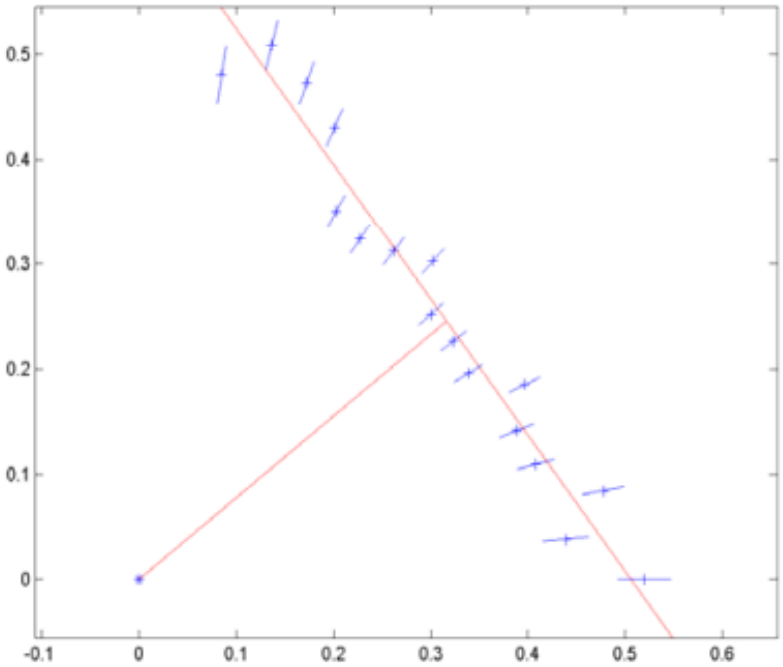
$$w_i = 1/\sigma_i^2$$

$$S = \sum w_i d_i^2 = \sum w_i (\rho_i \cos(\theta_i - \alpha) - r)^2$$

Least squares 2

- 17 measurement
- error (σ) proportional to ρ^2
- weighted least square:

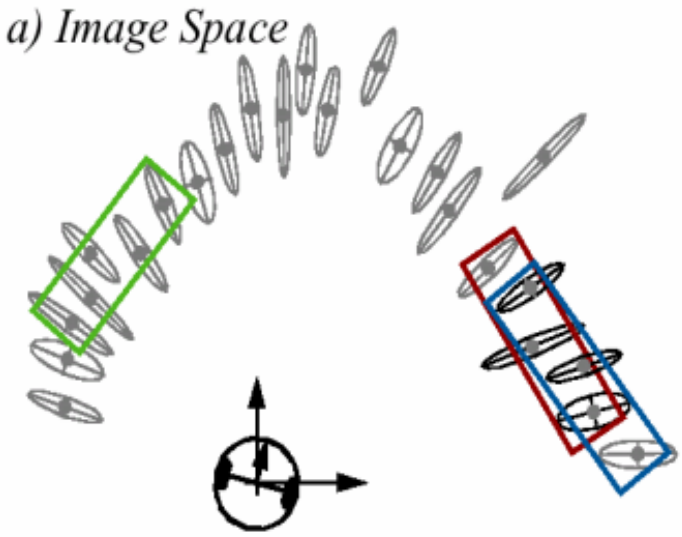
$$w_i = 1/\sigma_i^2$$



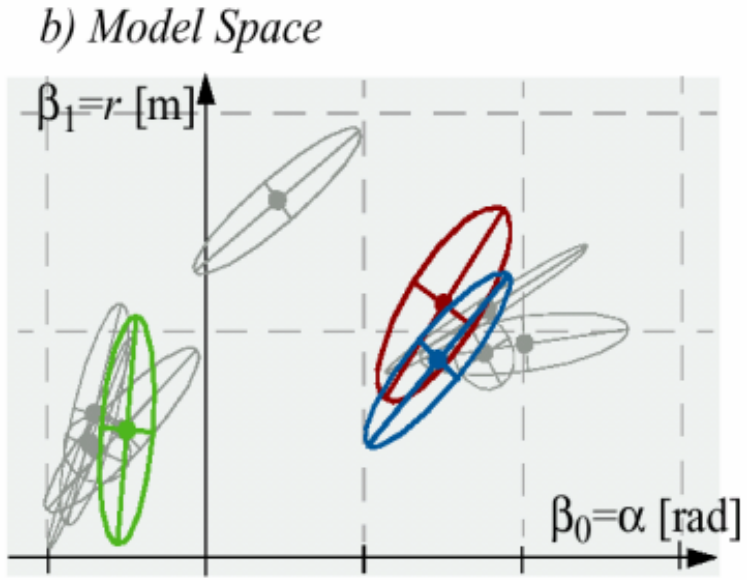
$$\alpha = \frac{1}{2}\operatorname{atan}\left(\frac{\sum w_i\rho_i^2\sin 2\theta_i - \frac{2}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos\theta_i \sin\theta_j}{\sum w_i\rho_i^2\cos 2\theta_i - \frac{1}{\sum w_i}\sum\sum w_i w_j \rho_i \rho_j \cos(\theta_i + \theta_j)}\right)$$

$$r = \frac{\sum w_i \rho_i \cos(\theta_i - \alpha)}{\sum w_i}$$

Clustering



A set of n_f neighboring points of the image space



Evidence accumulation *in the model space*
→ *Clusters of normally distributed vectors*

Incremental

Algorithm

1. Start by the first two points, construct a line
2. Add the next point to the current line model
3. Recompute the line parameters
4. If it satisfies the line condition, continue (step 2)
5. Otherwise, put back the last point, recompute the line parameters, return the line
6. Continue with the next two points, continue (step 2)

RANSAC

Algorithm: RANSAC

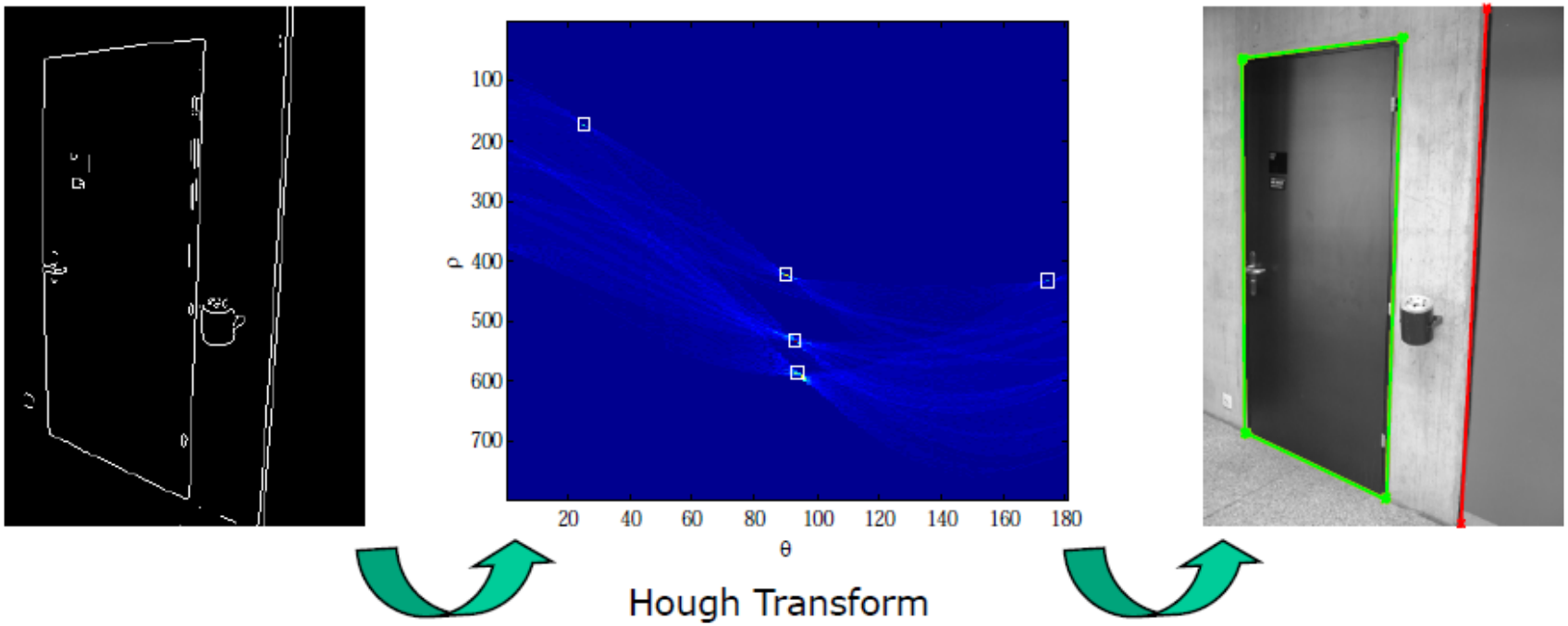
1. Initial: set of N points
2. **repeat**
3. Choose a sample of 2 points at random (uniform sample)
4. Fit a line through the two points
5. Compute the distances of the other points to the line
6. Construct the inlier set
7. If there are enough inliers, recompute the line parameters, store the line, remove the inliers from the set
8. **until** Max iterations reached or two few points left

Hough Transform

- Well known tool in computer vision
- Very successfully applied on intensity images
- It computes an accumulator array of line model space, then selects by voting.
- Difficult to select grid size, does not model noise

[Fisher et al., 2003]

Hough Transform



Hough Transform

- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?

- Option 2:
  - Use a voting scheme: Hough transform

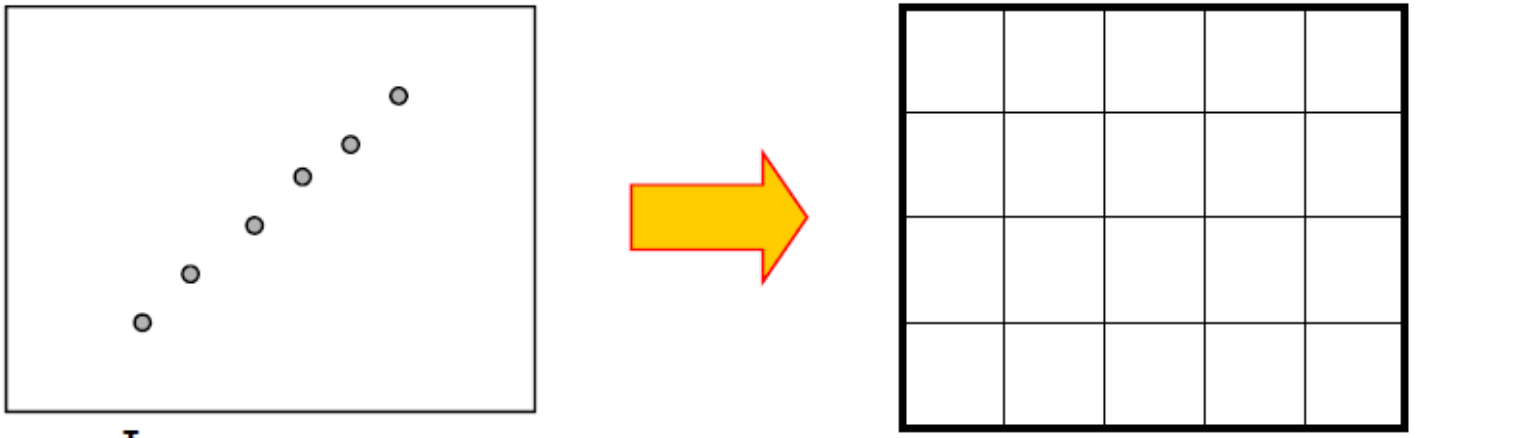
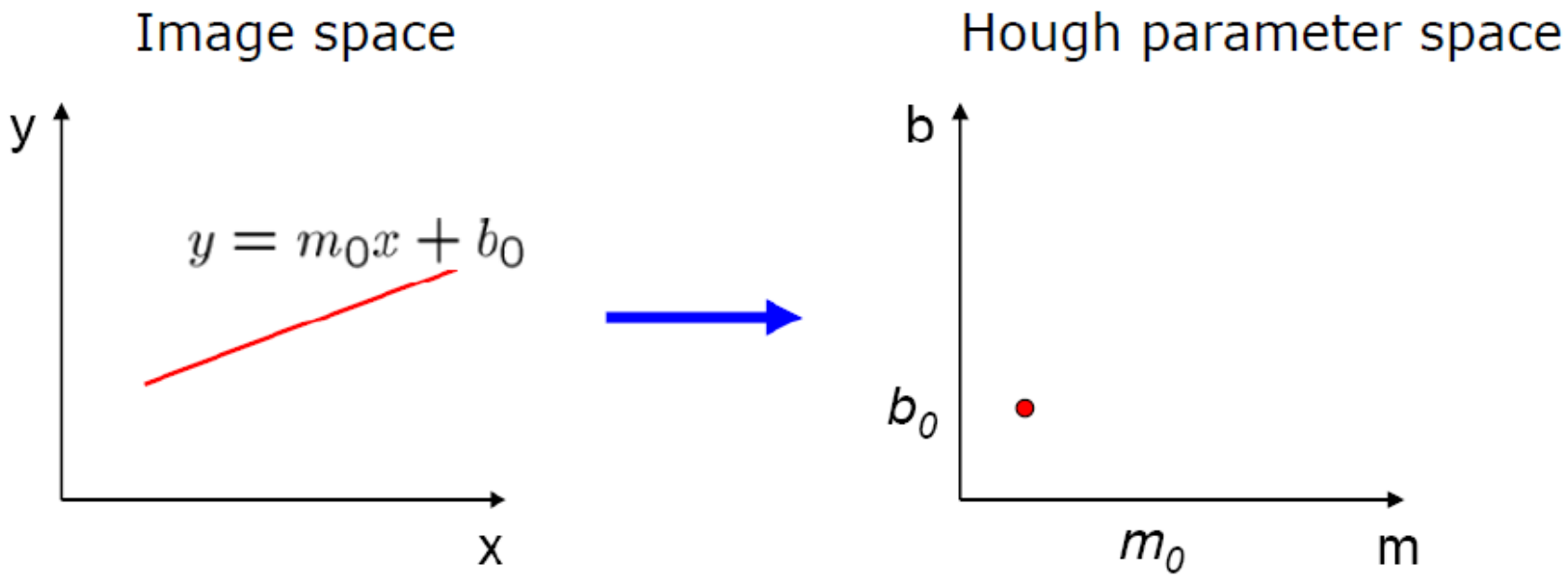


Image space

Hough parameter space

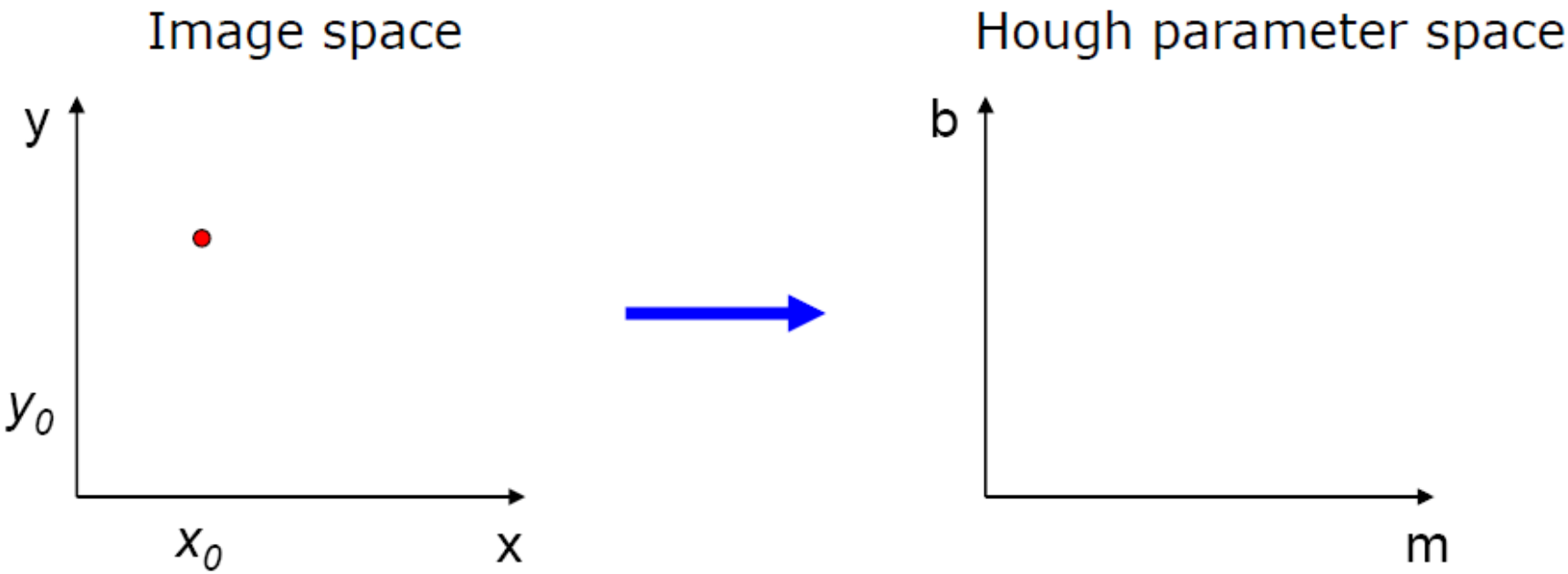
P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Hough Transform



Hough Transform

- 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



Hough Transform

If you have the model $y = mx + b$ and one data point (x_0, y_0), then plugging in that point gives

$$y_0 = mx_0 + b$$

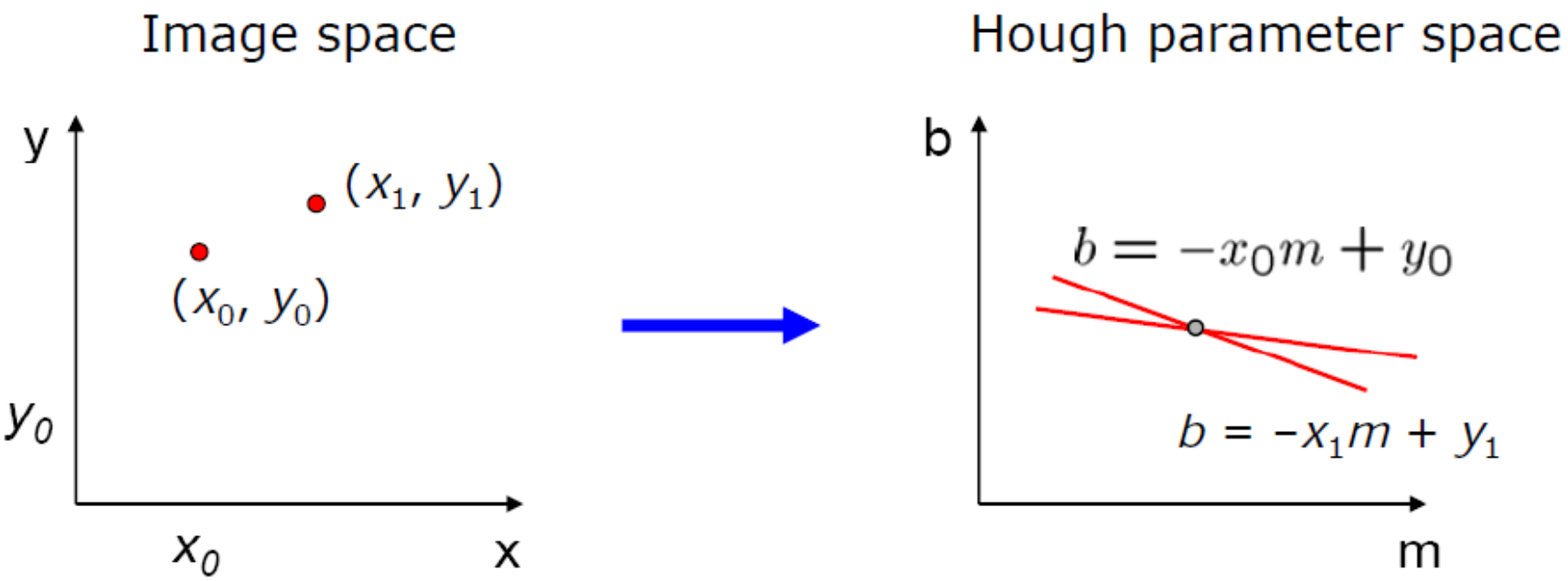
thus

$$b = -x_0 m + y_0.$$

This is an linear equation in b and m. The graph is a line in the m, b plane. Therefor points map to lines and lines map to points in the Hough Transform.

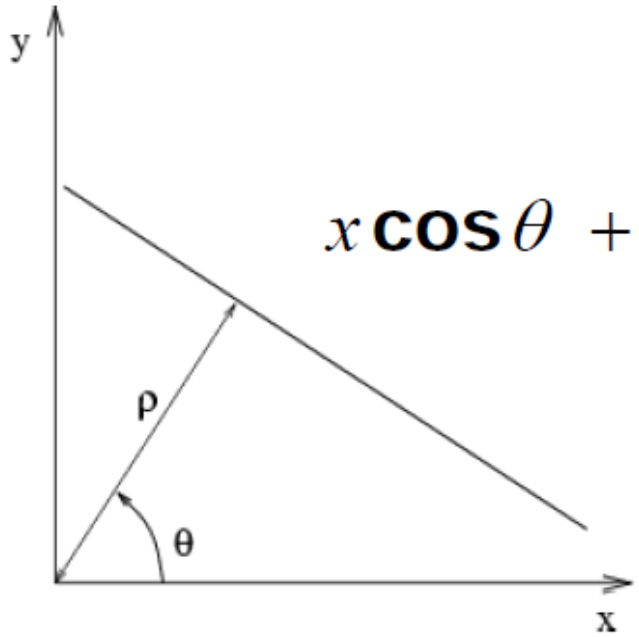
Hough Transform

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



Hough Transform

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

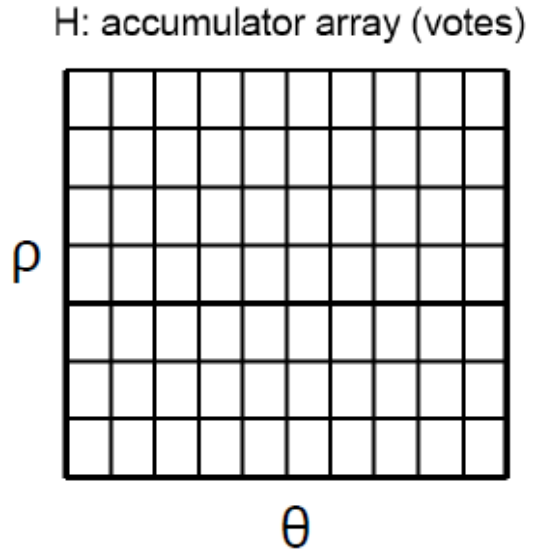


$$x\cos\theta + y\sin\theta = \rho$$

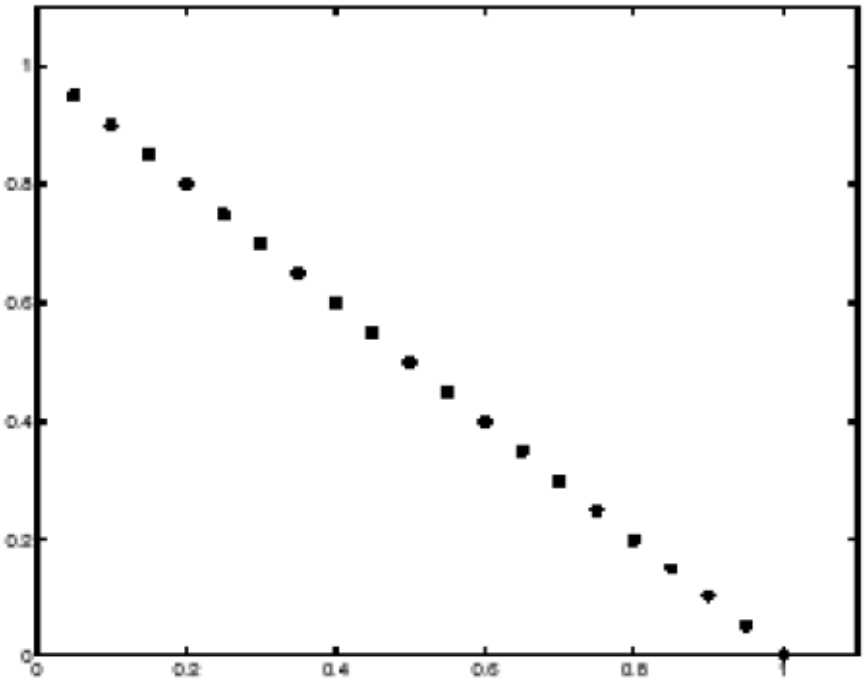
Each point will add a sinusoid in the (θ,ρ) parameter space

Hough Transform

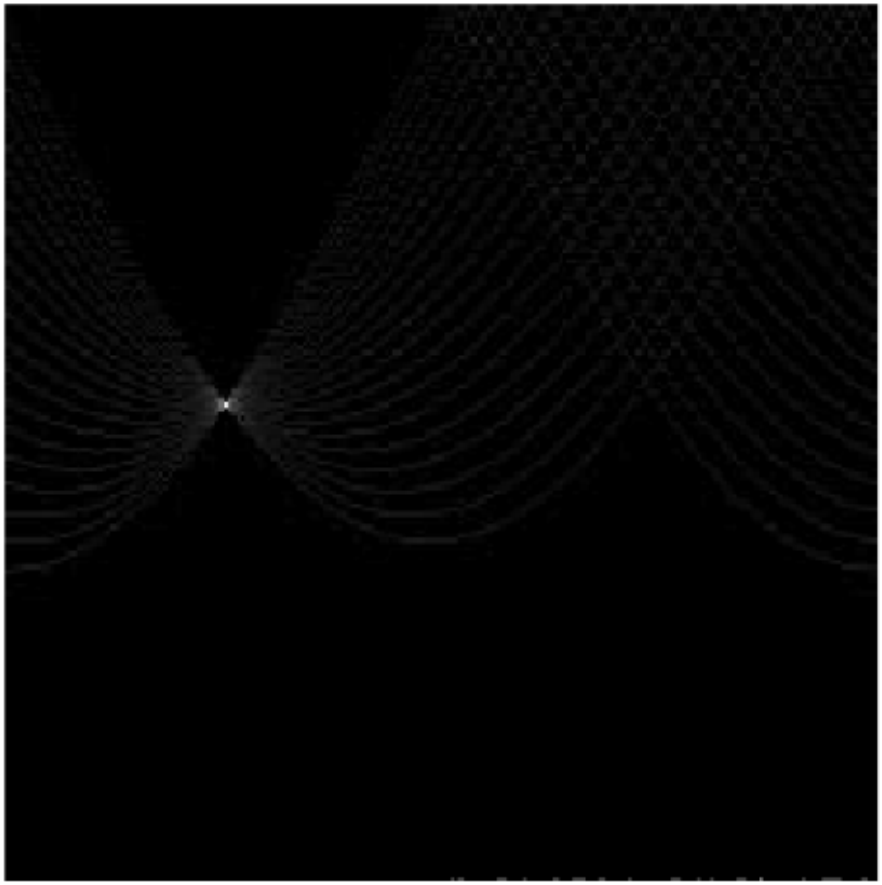
- Initialize accumulator H to all zeros
- For each edge point (x,y) in the image
  For θ = 0 to 180
    ρ = x cos θ + y sin θ
    H(θ, ρ) = H(θ, ρ) + 1
  end
end
- Find the value(s) of (θ, ρ) where H(θ, ρ) is a local maximum
  - The detected line in the image is given by
    ρ = x cos θ + y sin θ



Hough Transform



features

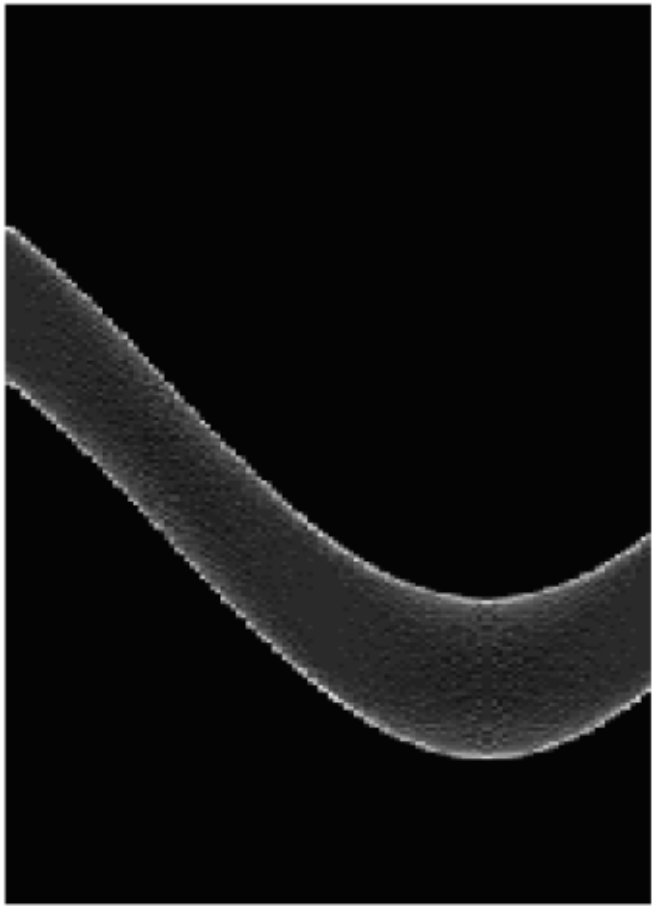


votes

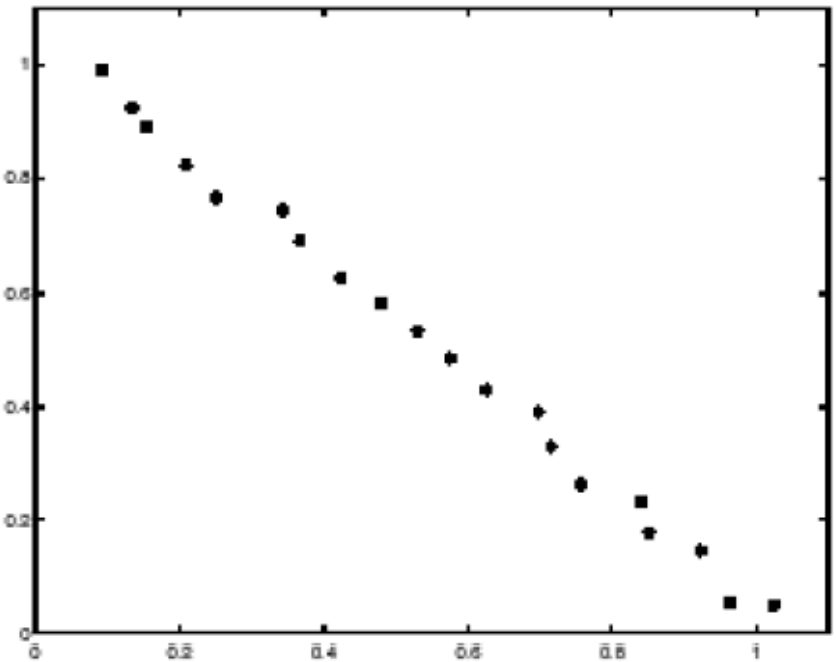
Hough Transform

Square

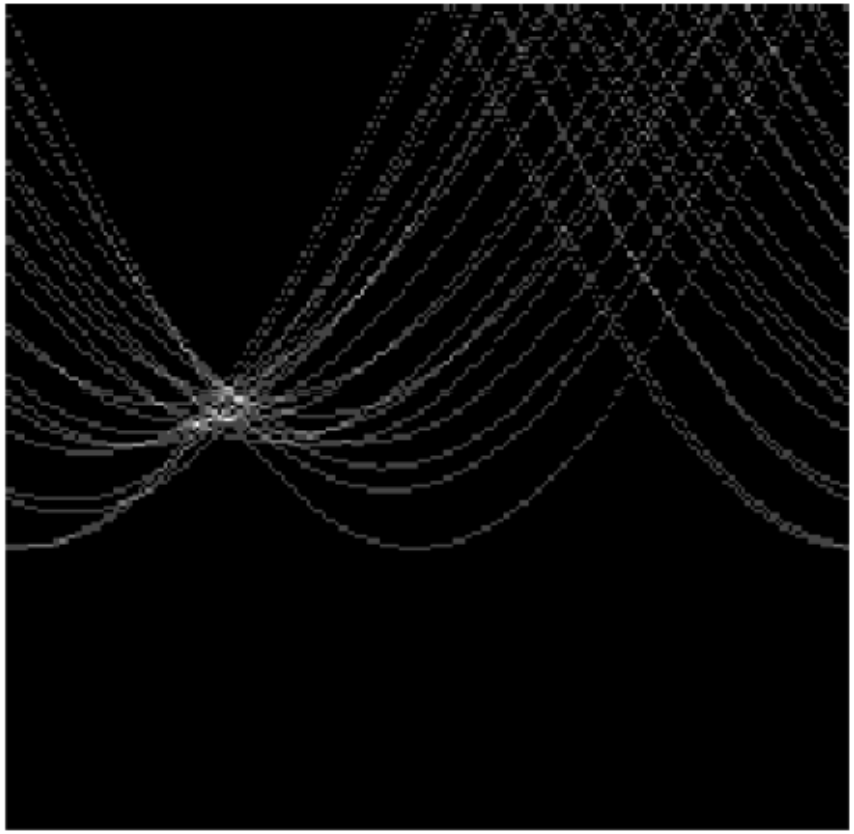
Circle



Hough Transform



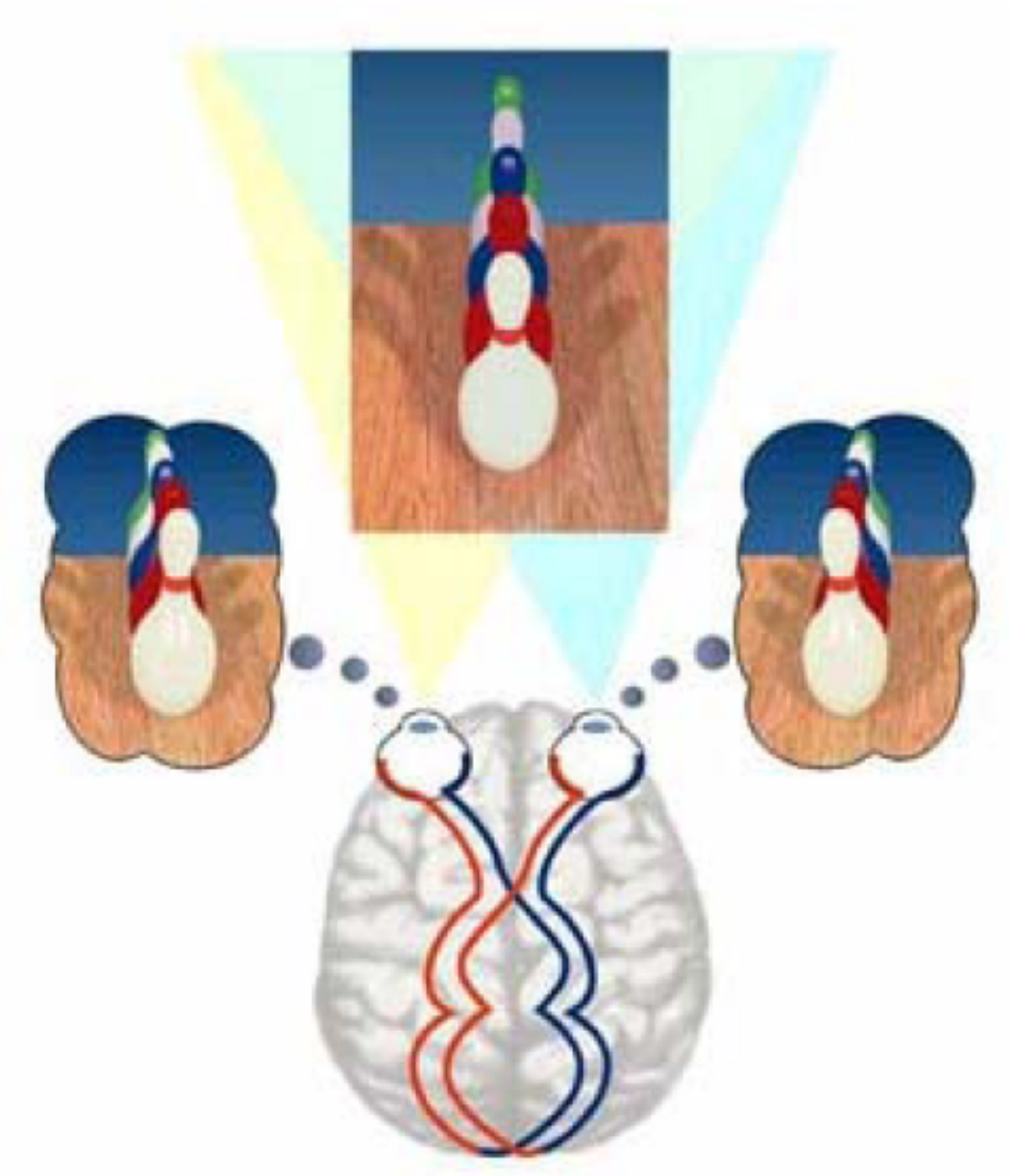
features



votes

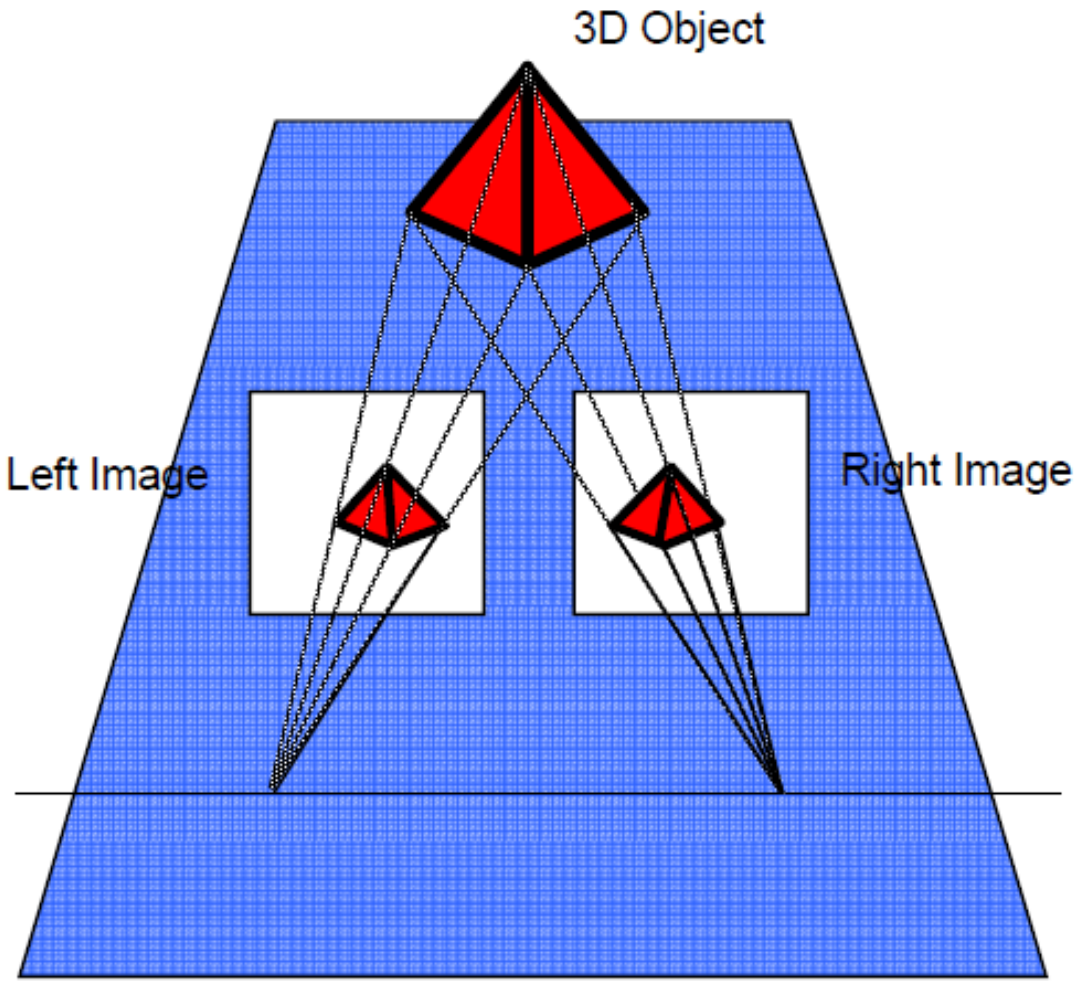
- Peak gets fuzzy and hard to locate

Human Vision



Stereo Vision

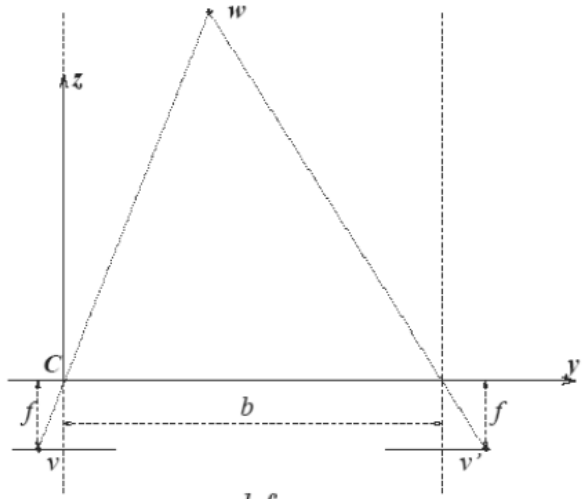
One may reconstruct 3D objects using a pair of cameras

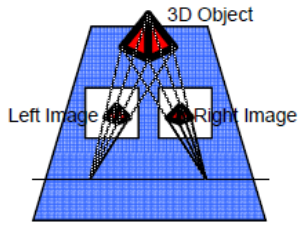


Stereo Vision

The simplified case is an ideal case. It assumes that both cameras are identical and are aligned on a horizontal axis

$$\begin{cases} \frac{-f}{z} = \frac{-v}{y} \\ \frac{-f}{z} = \frac{v'}{b-y} \end{cases}$$





$$z = \frac{bf}{v - v'}$$ Distance

- b = baseline, distance between the optical centers of the two cameras
- f = focal length
- v-v' = disparity

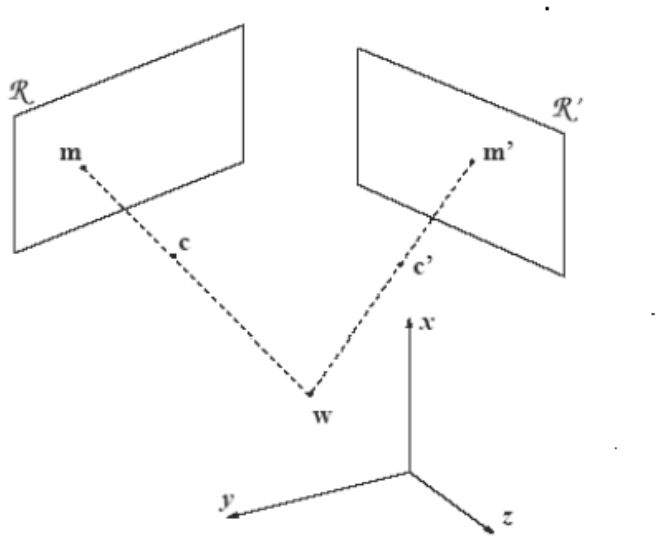
Stereo Vision

$$z = \frac{bf}{v - v'}$$

- Distance is inversely proportional to disparity $(v - v')$ closer objects can be measured more accurately
- Disparity is proportional to b. For a given disparity error, the accuracy of the depth estimate increases with increasing baseline b. However, as b is increased, some objects may appear in one camera, but not in the other.
- Increasing image resolution improves accuracy

Improved Stereo Vision

- Two identical cameras do not exist in nature!
- Aligning both cameras on a horizontal axis is very hard, also with the most expensive stereo cameras!



- In order to be able to use a stereo camera, we need first to estimate the relative pose between the cameras, that is, Rotation and Translation
- However as the two cameras are not identical, we need to estimate: focal length, image center, radial distortion

Calibration

Four conjugate points are required to determine r_{ij}

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} + \begin{bmatrix} r_{01} \\ r_{02} \\ r_{03} \end{bmatrix}$$

Solve with LU or QR factorization techniques. One needs to rewrite as a vector problem, but it is still a linear system.

$$x_{r,1} = x_{l,1} r_{11} + y_{l,1} r_{12} + z_{l,1} r_{13} + r_{01}$$

....

Calibration

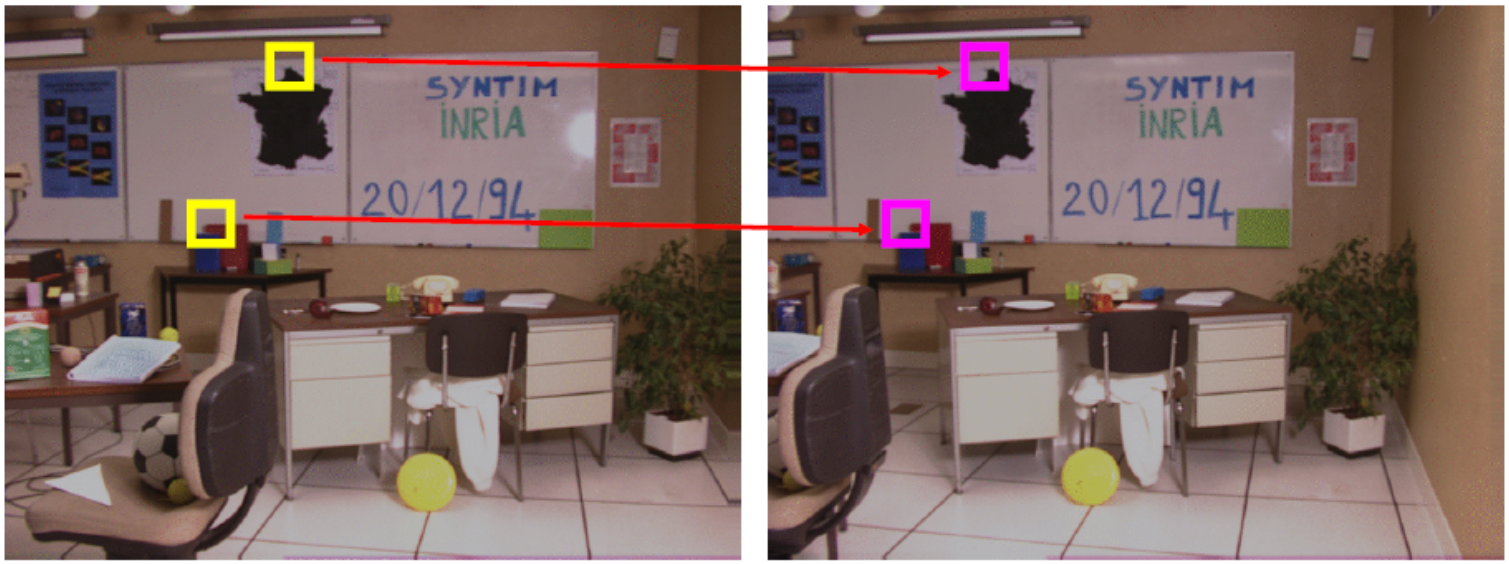
First row:

$$x_{r,1} = \begin{bmatrix} x_{l,1} & y_{l,1} & z_{l,1} & 1 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \\ r_{01} \\ r_{21} \\ r_{22} \\ r_{23} \\ r_{02} \\ r_{31} \\ r_{32} \\ r_{33} \\ r_{03} \end{bmatrix}$$

Remaining rows follow ...

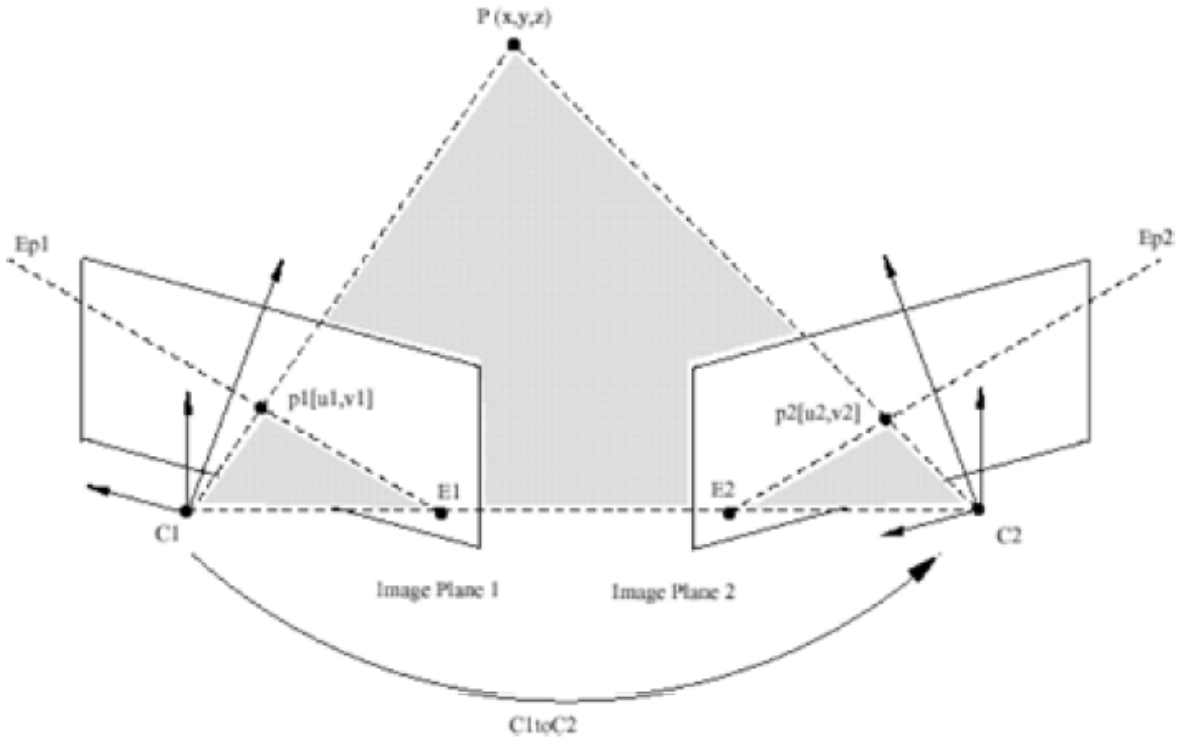
Stereo Vision: Correspondence Problem

- Matching between points in the two images which are projection of the same 3D real point
- Correspondence search could be done by comparing the observed points with all other points in the other image. Typical similarity measures are the Correlation and image Difference.
- This image search can be computationally very expensive! Is there a way to make the correspondence search 1 dimensional?



Correspondence Problem: Epipolar Constraint

- The correspondent of a point in an image must lie on a line in the other image, called Epipolar Line

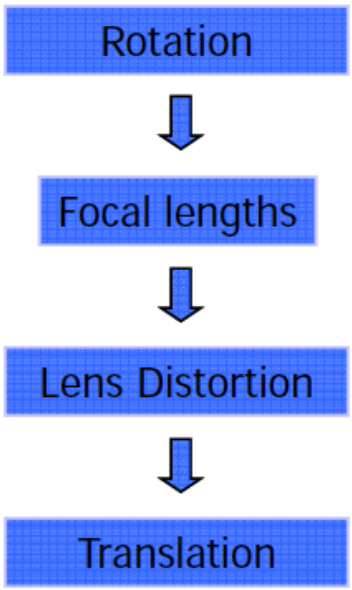


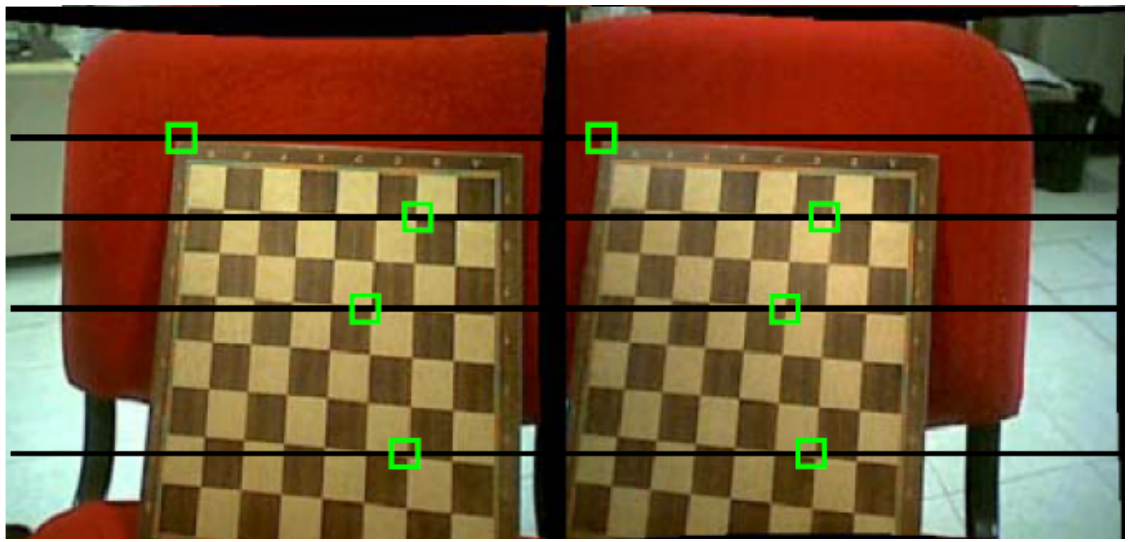
Correspondence Problem: Epipolar Constraint

- Thanks to the epipolar constraint, conjugate points can be searched along epipolar lines: this reduces the computational cost to 1 dimension!

Epipolar Rectification

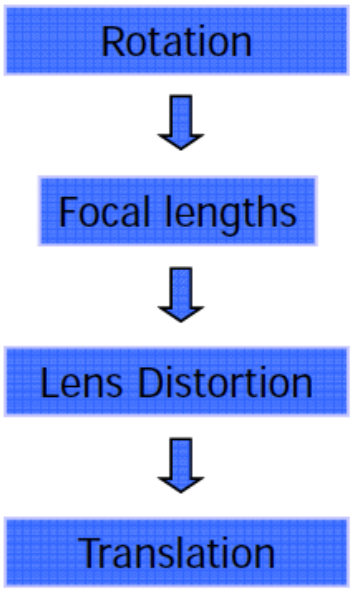
- Determines a transformation of each image plane so that pairs of conjugate epipolar lines become collinear and parallel to one of the image axes (usually the horizontal one)

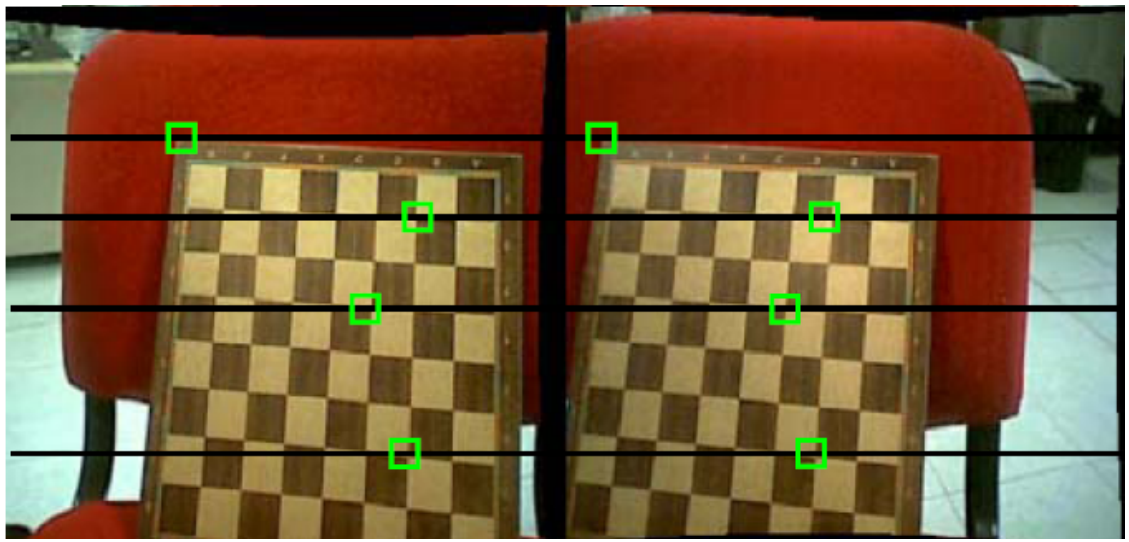




Epipolar Rectification

- Determines a transformation of each image plane so that pairs of conjugate epipolar lines become collinear and parallel to one of the image axes (usually the horizontal one)





Disparity Map

- Find the conjugate points of all image pixels of the original images
- For each pair of conjugate points compute the disparity $d = v - v'$
- $d(x, y)$ is called the Disparity map.

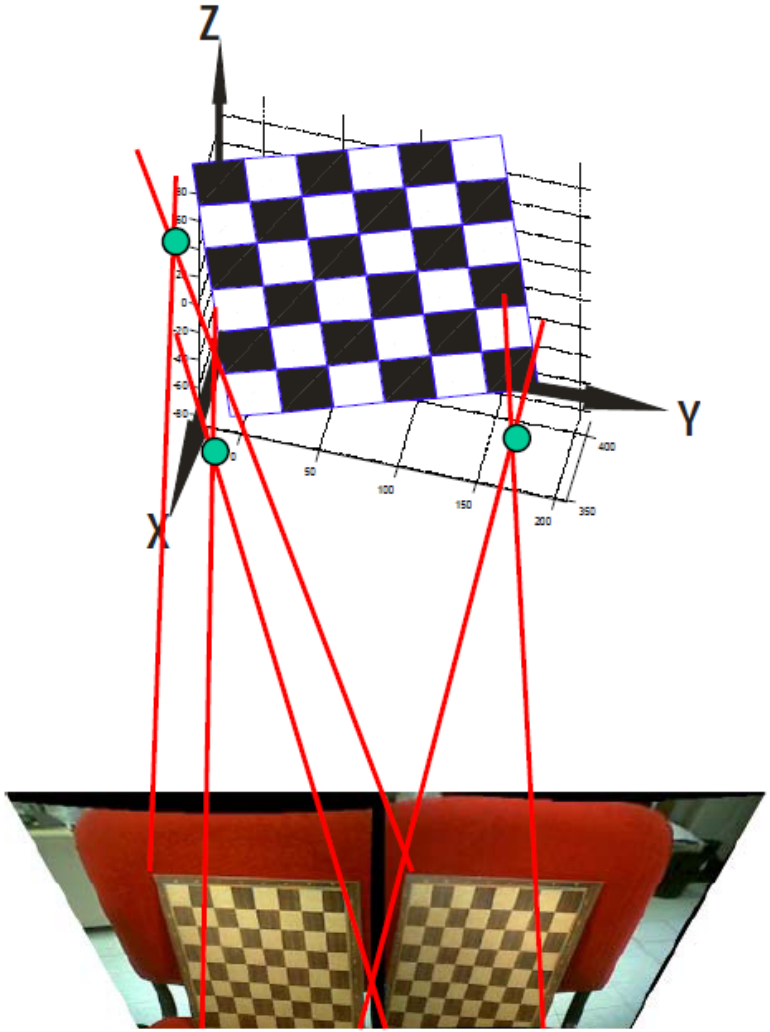
Left image

Right image

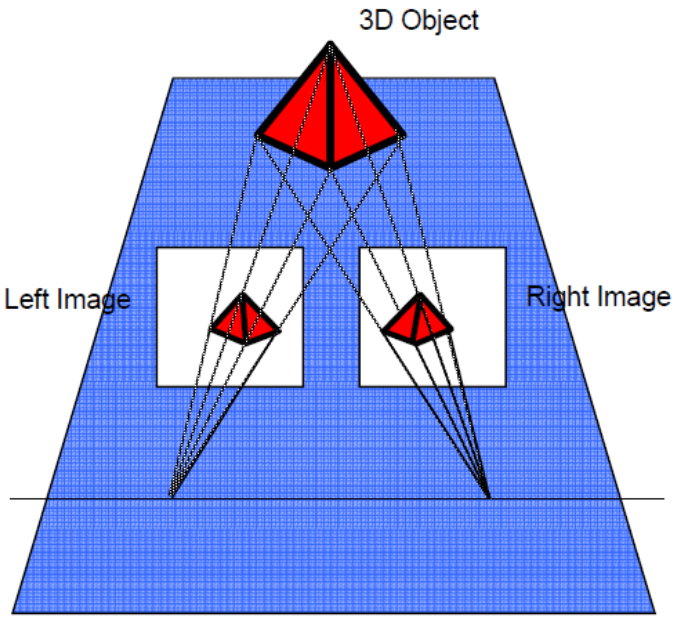

Disparity map

Disparity maps are usually visualized as grey-scale images. Objects that are closer to the camera appear lighter, those who are further appear darker.

Triangulation - 3D reconstruction



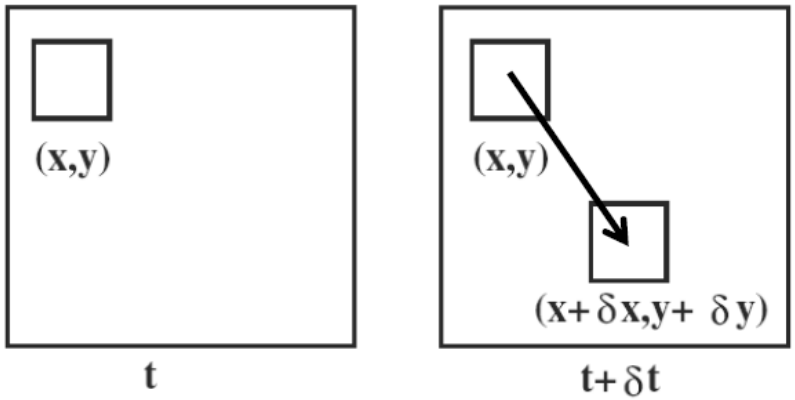
Stereo Vision Summary



1. Stereo camera calibration $\rightarrow$ compute camera relative pose
2. Epipolar rectification $\rightarrow$ align images
3. Search Correspondences
4. Output: compute stereo triangulation or disparity map
5. Consider baseline and image resolution to compute accuracy

Optical Flow

- Optical flow is an approximation of the apparent motion of objects within an image. Algorithms used to calculate optical flow attempt to find correlations between near frames in a video, generating a vector field showing where each pixel or region in the original image moved to in the second image Typically the motion is represented as vectors originating or terminating at pixels in a digital image sequence.
- Estimating the optical flow is useful in pattern recognition, computer vision, and other image processing applications. It is closely related to motion estimation and motion compensation.

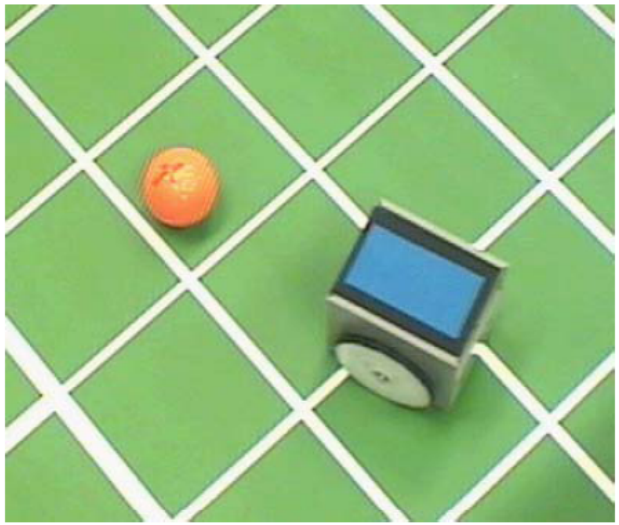


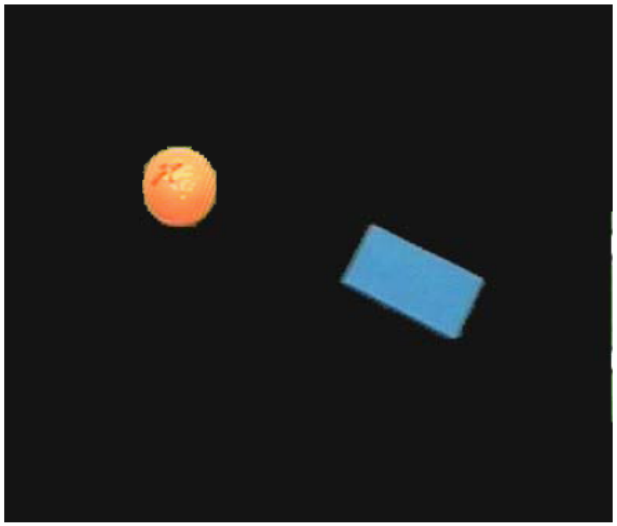
Optical Flow

- Phase Correlation
  - Calculate (2D) Fourier Transform of each image: $F_1 = \mathcal{F}(I_1)$, $F_2 = \mathcal{F}(I_2)$
  - Calculate the cross-power spectrum: $R = \frac{F_1 F_2^*}{\|F_1 F_2^*\|}$
  - Calculate the inverse Fourier Transform: $r = \mathcal{F}^{-1}(R)$
  - Compute max of r: $(\Delta x, \Delta y) = \text{argmax}_{(x,y)} r$
- Minimum Sum of Differences
  - Build a template
  - Place over image and compute differences in values (pointwise)
  - Sum absolute values
  - Minimize over all shifts $\Rightarrow (\Delta x, \Delta y)$

Color Tracking

- Motion estimation of ball and robot for soccer playing using color tracking





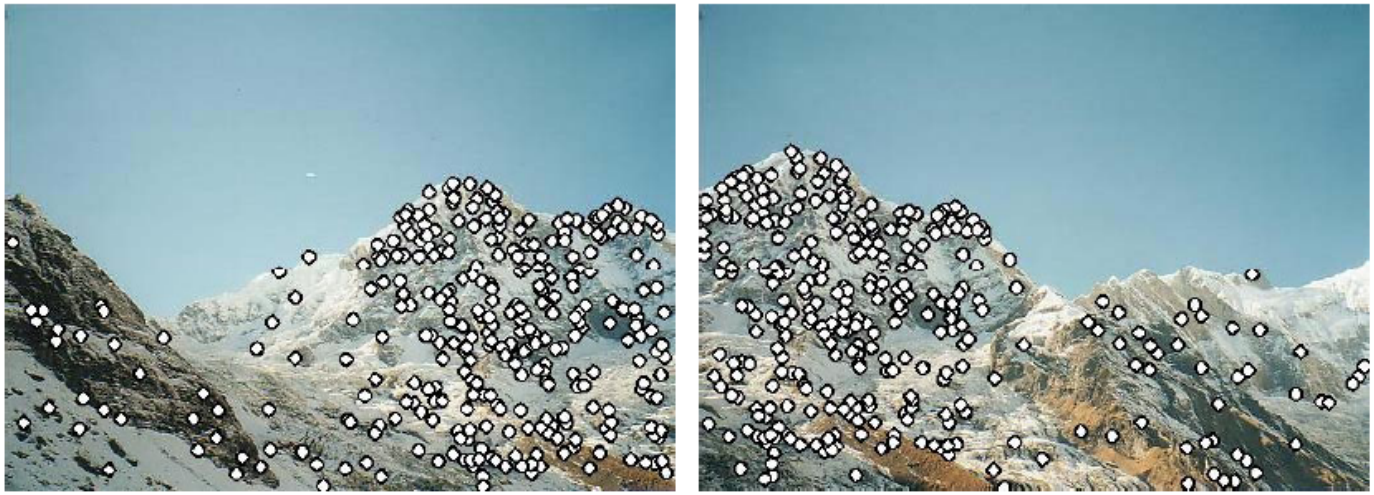
Matching Images

How can we align images (some additional details)...

- Stereo Vision
- Creating panoramas
- Recognition of known landscapes

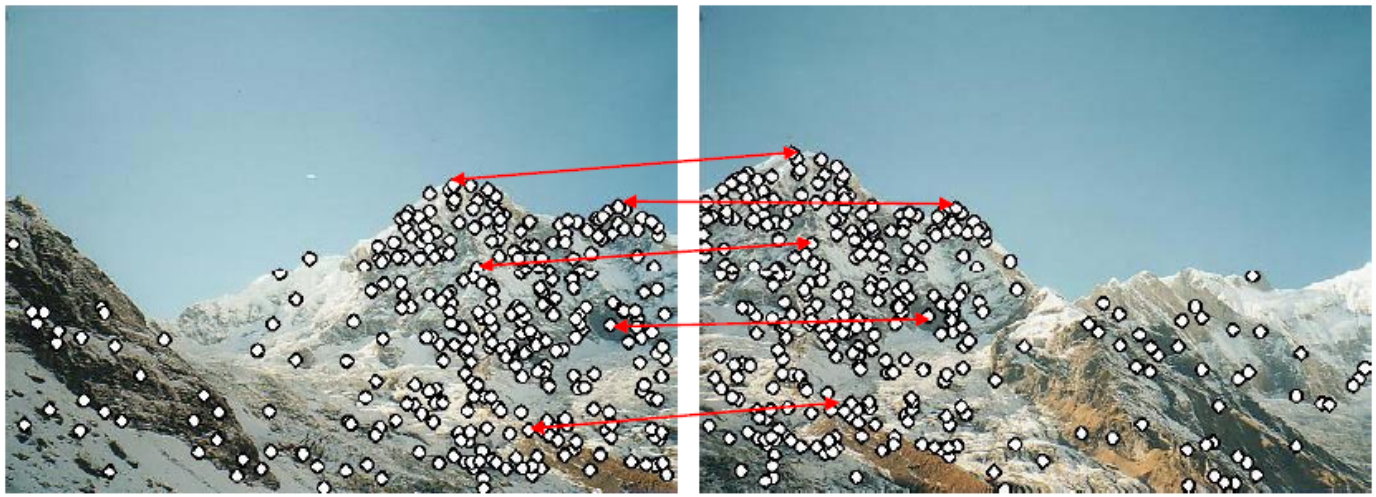
Matching Images

Detect features on both images



Matching Images

Find corresponding points



Matching Images

Use points to align images

Problems

Problem1
Detect the same points in both images

no chance to match!

We need a repeatable detector

Problems

Problem2

For each point, determine the point that corresponds in the other image

We need a reliable and distinctive descriptor

Image Stitching

The simpliest way to establish the correspondence between two images is to translate one image relative to another and match. This is commonly approached via minimizing the sum of squared differences (SSD) over different shifts:

$$E_{SSD}(u, v) \equiv \sum_{i,j} [I_1(i + u, j + v) - I_0(i, j)]^2$$

The shift can fall outside the valid image range, so a window function is introduced

$$E_{SSD}(u, v) \equiv \sum_{i,j} w_1(i, j) w_2(i + u, j + v) [I_1(i + u, j + v) - I_0(i, j)]^2$$

Image Stitching

The goal is to determine $(\hat{u}, \hat{v})$:

$$(\hat{u}, \hat{v}) = \text{argmin}_{(u,v)} \; E_{SSD}(u, v)$$

$$= \text{argmin}_{(u,v)} \; \sum_{i,j} w_1(i,j) w_2(i+u, j+v) \left[I_1(i+u, j+v) - I_0(i,j) \right]^2$$

How does one find the min?

Optimization

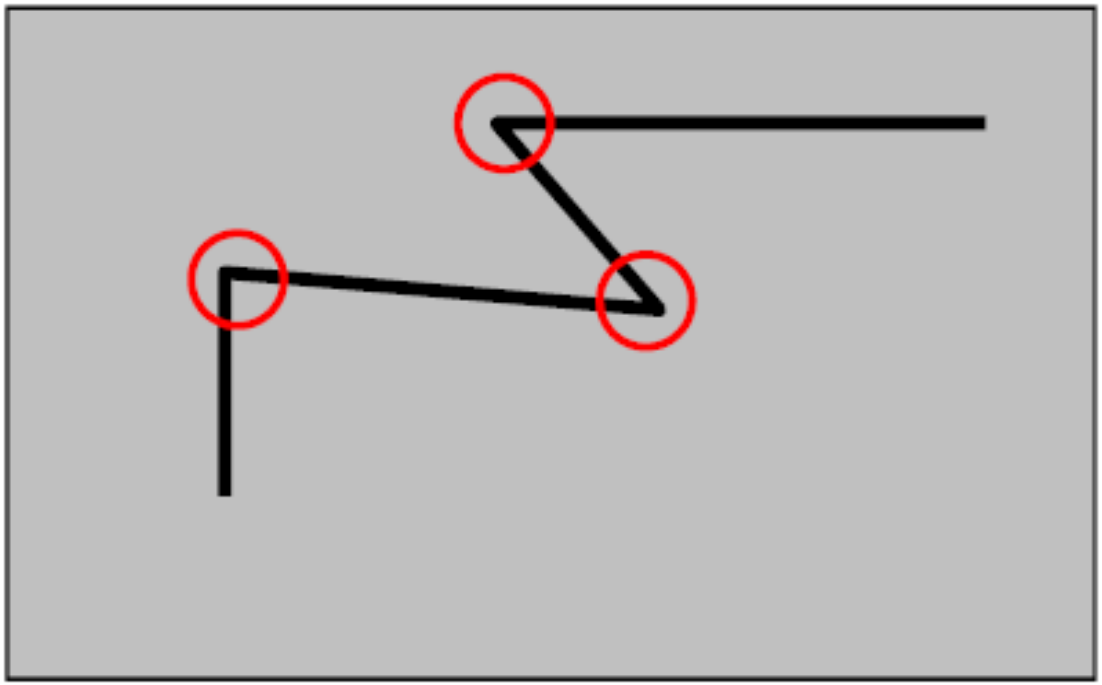
Assume that the image is $n \times n$ pixels.

- For each pixel, E_{SSD} is a sum over all pixels within the window.
- For this discussion, assume the average case has the window one half the image.
- This gives us roughly a workload of $0.5n^2$ multiplies (plus additional additions) per shift.
- A discrete search is needed to locate the optimal shift.
- Steepest descent or other types of search algorithms may be employed.

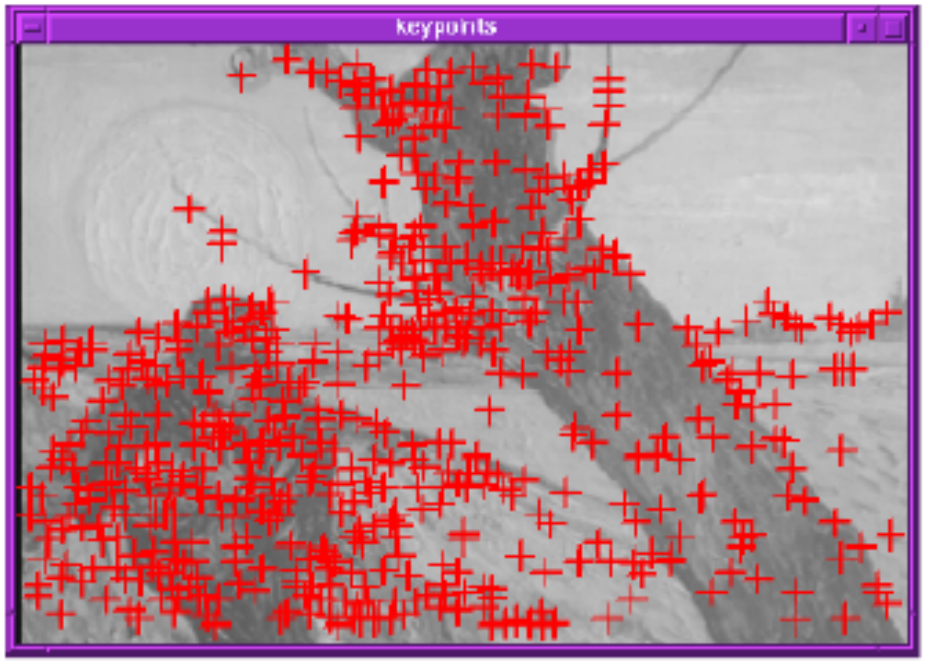
This type of Template Matching is prone to many issues like changes in intensity and slight deviations in rotation. Is there a more robust approach?

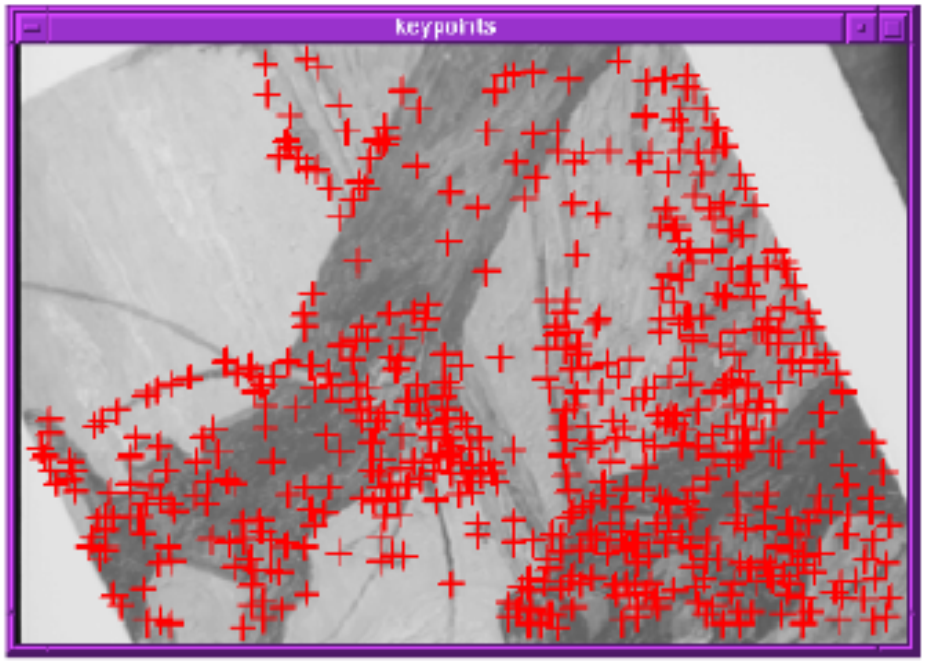
Harris Corner Detector

Corners as features.



Corner Detector

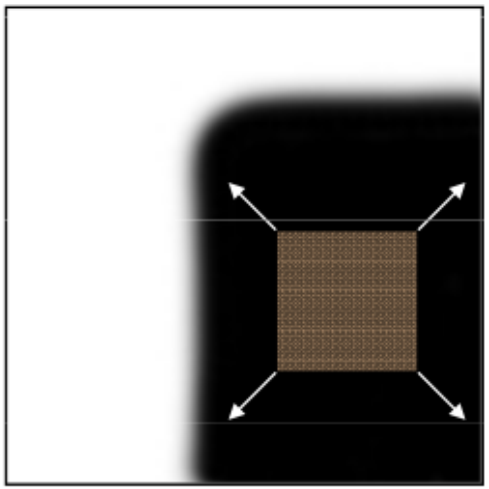




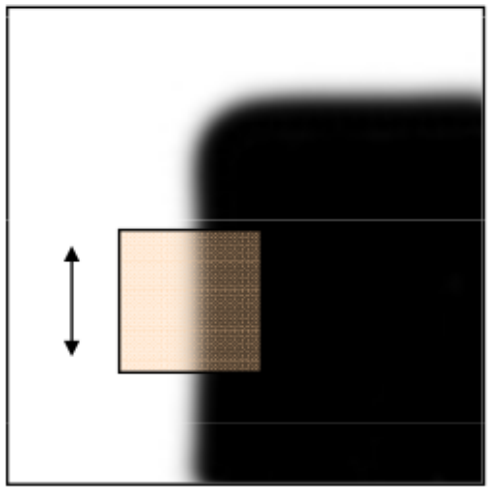
- In the region around a corner, image gradient has high variation in two or more directions.
- Corners are repeatable and distinctive.

Corner Detector

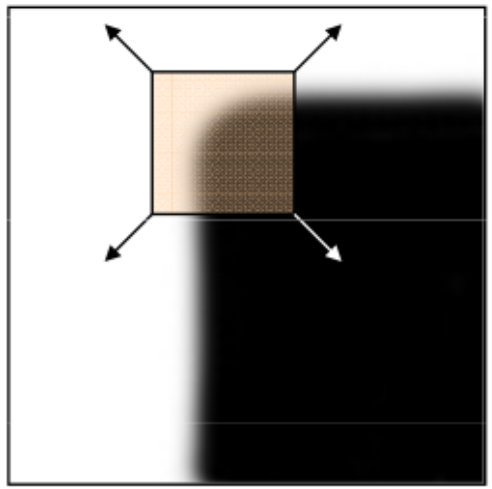
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give a *large change* in intensity



"flat" region: no change in all directions



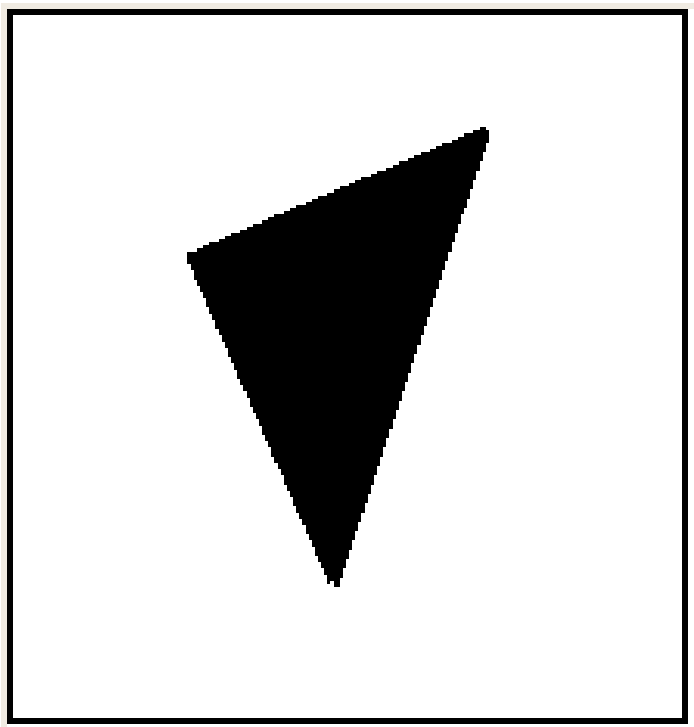
"edge": no change along the edge direction

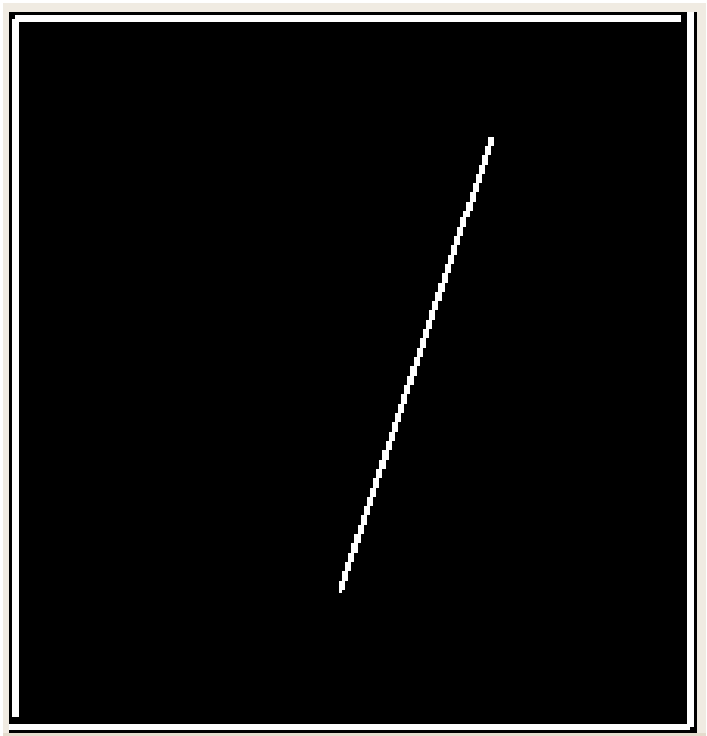


"corner": significant change in all directions

Try Sobel

Start with the figure on the left,
apply the Sobel horizontal and vertical filters:





Harris Detector

How does one determine corners? We want a rotation and translation invariant measure for finding corners.

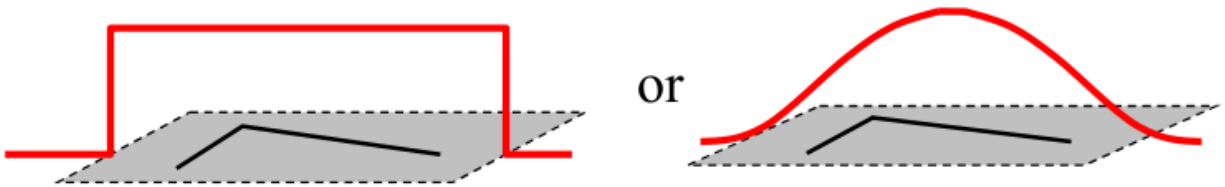
Without loss of generality, we will assume a grayscale 2-dimensional image is used. Let this image be given by I. Consider taking an image patch over the area (x, y) and shifting it by (u, v). The weighted sum of squared differences (SSD) between these two patches, denoted E, is given by:

Sums square differences

$$E(u,v) = \sum_x \sum_y w(x,y) \left[I(x+u, y+v) - I(x,y) \right]^2$$

Window Function Shifted Intensity Intensity

Window function



1 in window, 0 outside

Gaussian

Approximation for E

Expand $I(x+u, y+v)$ in a Taylor Series:

$$I(x+u, y+v) \approx I(x,y) + I_u(x,y)u + I_v(x,y)v$$

(the linearization) and so

$$E(u,v) \approx \sum_x \sum_y [I_u(x,y)u + I_v(x,y)v]^2$$

Note that

$$[I_u(x,y)u + I_v(x,y)v]^2 = [u,v] \begin{bmatrix} I_u^2 & I_u I_v \\ I_u I_v & I_v^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

so

$$\sum_x \sum_y [I_u(x,y)u + I_v(x,y)v]^2 = \sum_x \sum_y [u,v] \begin{bmatrix} I_u^2 & I_u I_v \\ I_u I_v & I_v^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Linearization of E

Structure Tensor

$$M = \begin{bmatrix} \sum_x \sum_y I_u^2 & \sum_x \sum_y I_u I_v \\ \sum_x \sum_y I_u I_v & \sum_x \sum_y I_v^2 \end{bmatrix}$$

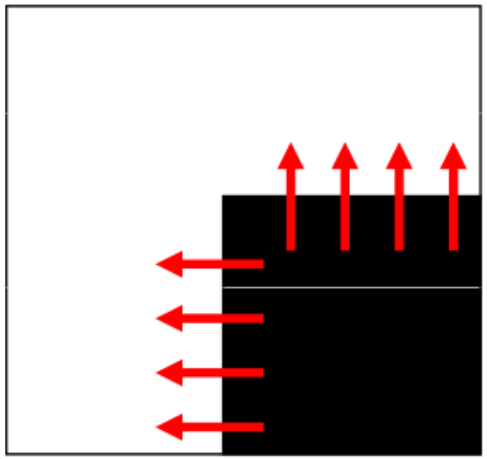
Thus we have that

$$E(u, v) \approx [u, v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

Note that M is a 2x2 matrix of sums of image derivatives. These derivatives are computed via finite differences (with several approaches done).

Interpretation of M

Assume that you have an image aligned corner:



$$M = \begin{bmatrix} \sum_x \sum_y I_u^2 & \sum_x \sum_y I_u I_v \\ \sum_x \sum_y I_u I_v & \sum_x \sum_y I_v^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means dominant gradient directions align with x or y axis.

Interpretation of M

If either λ is close to 0, then this is not a corner, so look for locations where both are large.

Based on the magnitudes of the eigenvalues, the following inferences can be made based on this argument:

1. If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then this pixel (u,v) has no features of interest.
2. If $\lambda_1 \approx 0$ and λ_2 has some large positive value, then an edge is found.
3. If λ_1 and λ_2 have large positive values, then a corner is found.

Interpretation of M

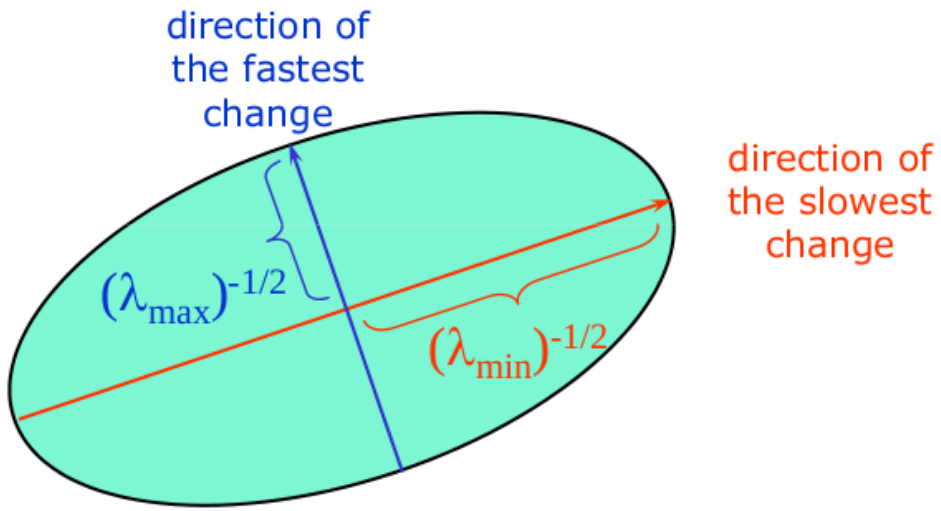
Since M is symmetric, we have by a similarity transform

$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

We can visualize M as an ellipse with axis lengths determined by the eigenvalues and orientation determined by R (the eigenvectors).

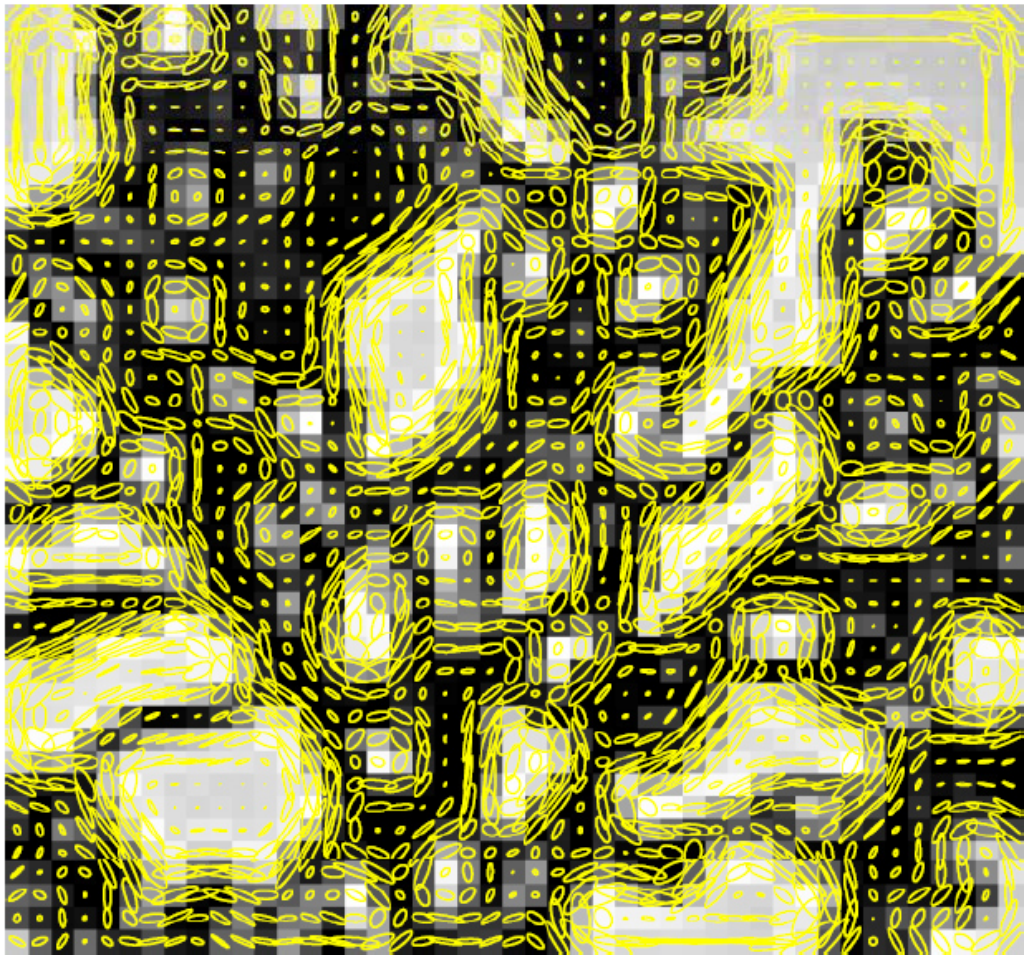
Ellipse equation:

$$[u \quad v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



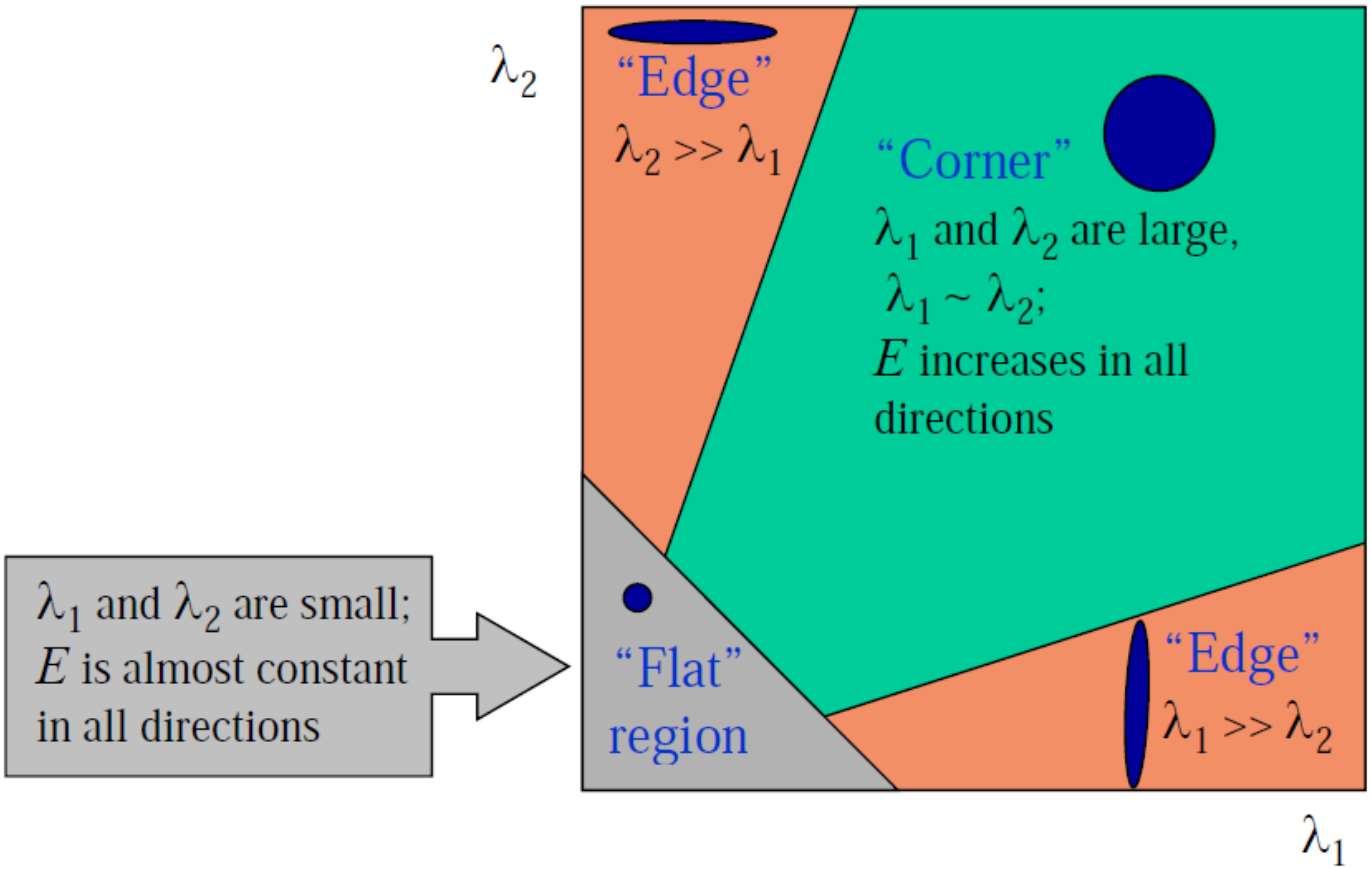
Example

Example



Classification

Classification of image points using eigenvalues of M:



Corner Response Function

Harris and Stephens note that exact computation of the eigenvalues is computationally expensive, since it requires the computation of a square root, and instead suggest the following function Mc, where α is a tunable sensitivity parameter:

$$M_\alpha = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2 = \det(M) - \alpha \, \text{trace}^2(M)$$

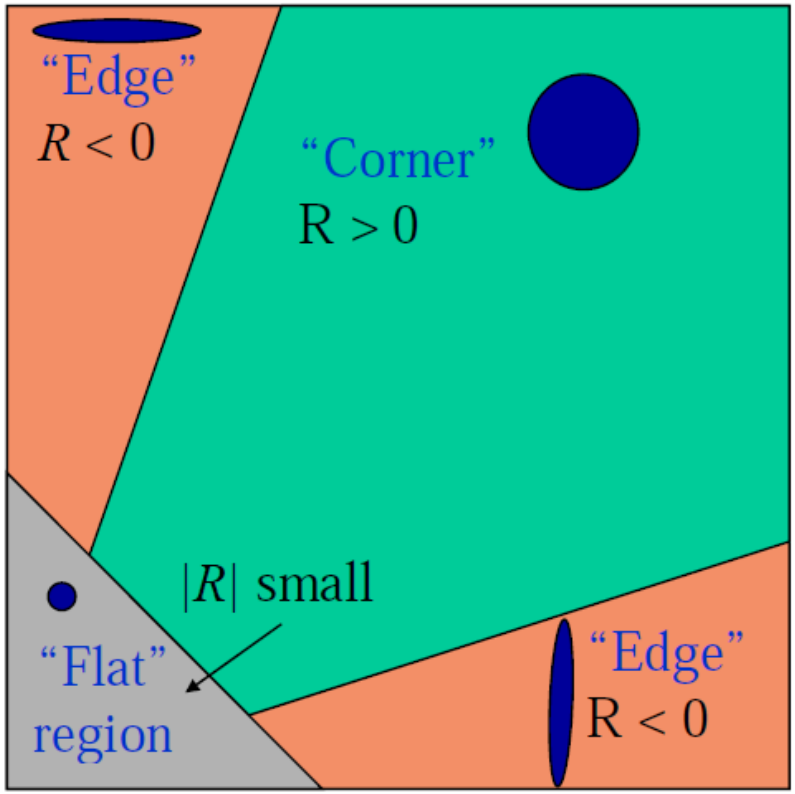
Therefore, the algorithm does not have to actually compute the eigenvalue decomposition of the matrix M and instead it is sufficient to evaluate the determinant and trace of M to find corners, or rather interest points in general.

The value of α has to be determined empirically, and in the literature values in the range 0.04 - 0.15 have been reported as feasible.

Classification via CRF

$$R = \det(M) - \alpha \,\mathrm{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha (\lambda_1 + \lambda_2)^2$$

α: constant (0.04 to 0.06)



Noise

In practice, since derivatives are computed, smoothing is applied to the image prior to computation of I_x and I_y.

This is normally a Gaussian and application is the convolution process described earlier: $L = G * I$, so then the structure tensor,

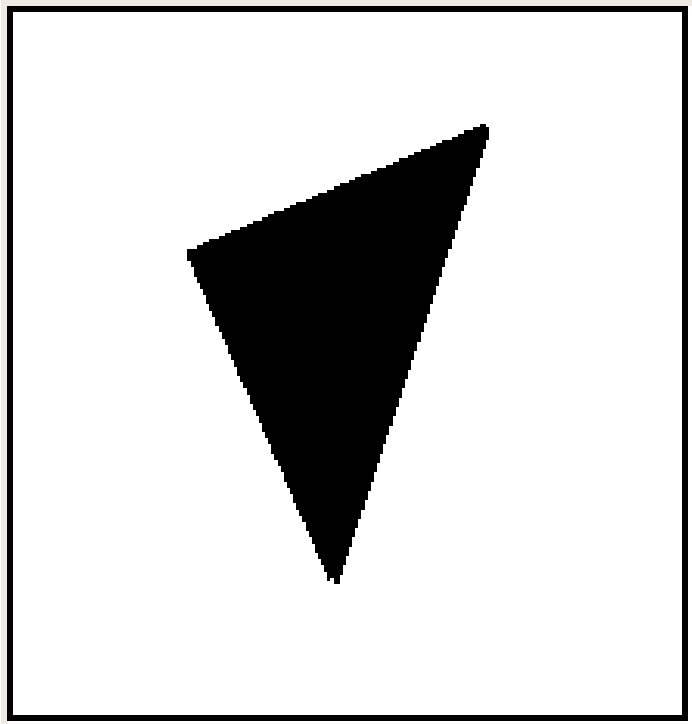
$$S = \begin{bmatrix} \sum_x \sum_y L_u^2 & \sum_x \sum_y L_u L_v \\ \sum_x \sum_y L_u L_v & \sum_x \sum_y L_v^2 \end{bmatrix}$$

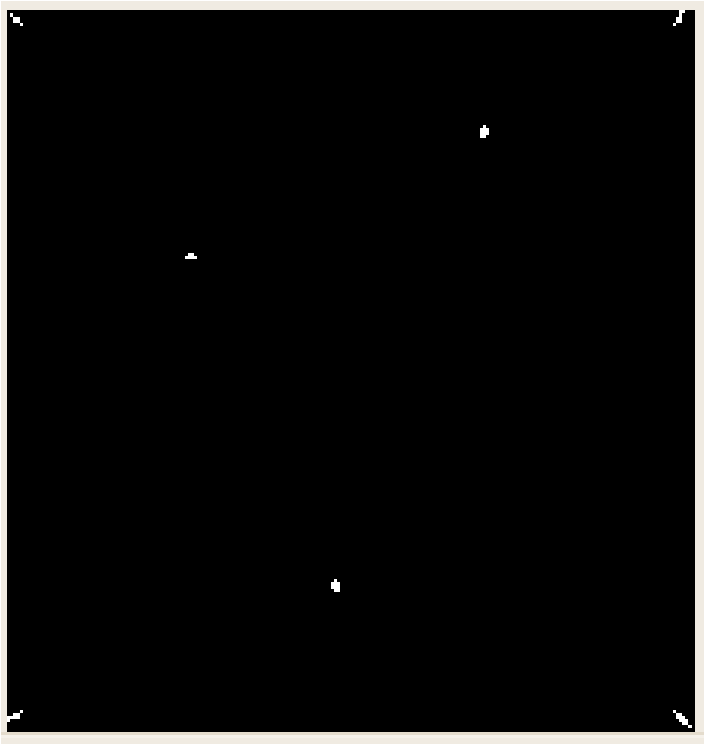
and the Corner Response Function would be applied:

$$S_\alpha = \det(S) - \alpha \ \text{trace}^2(S)$$

Simple Example

Return to our original figure and see how Harris does:

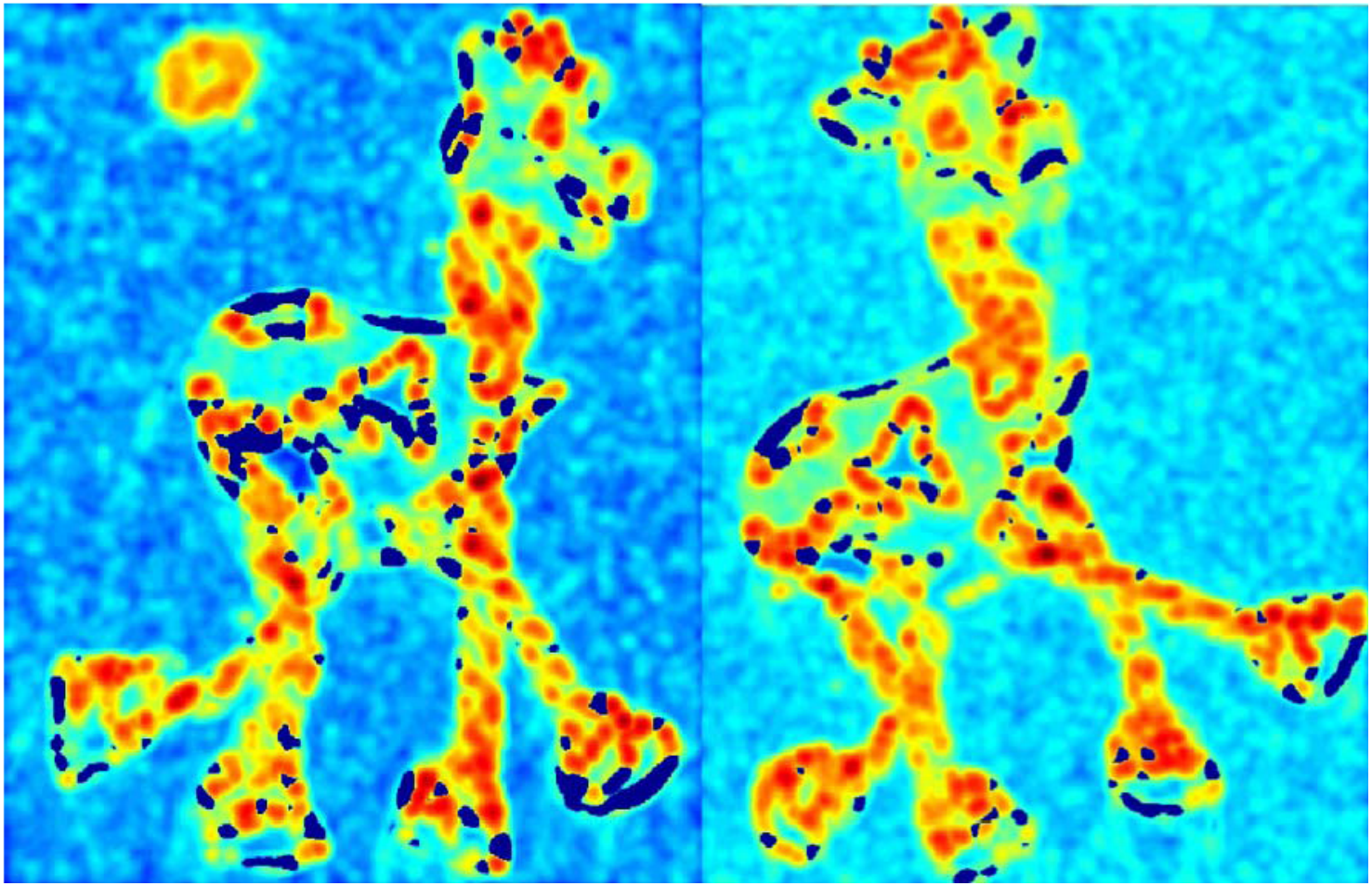




Example

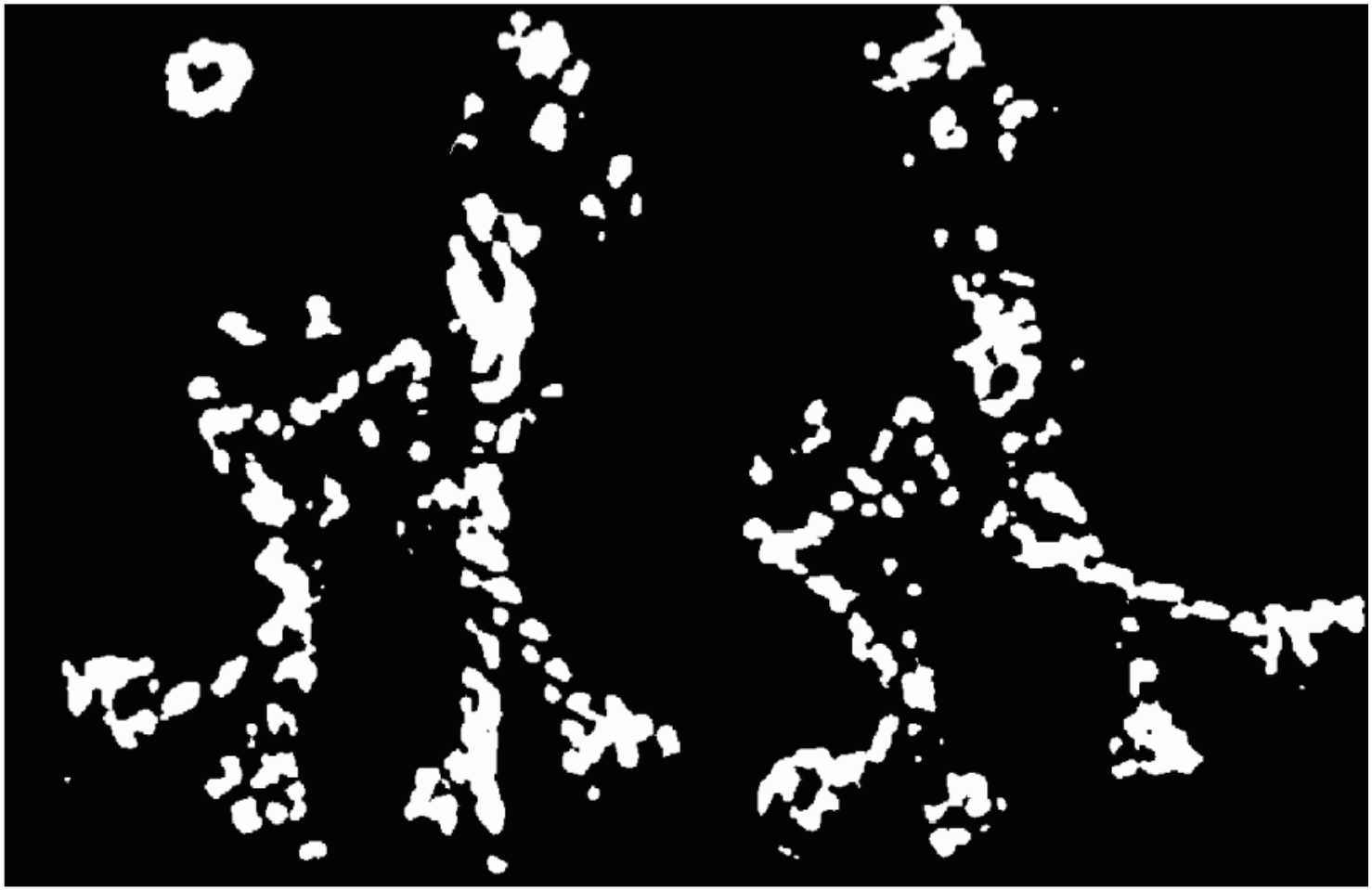
Example

- Compute corner response R



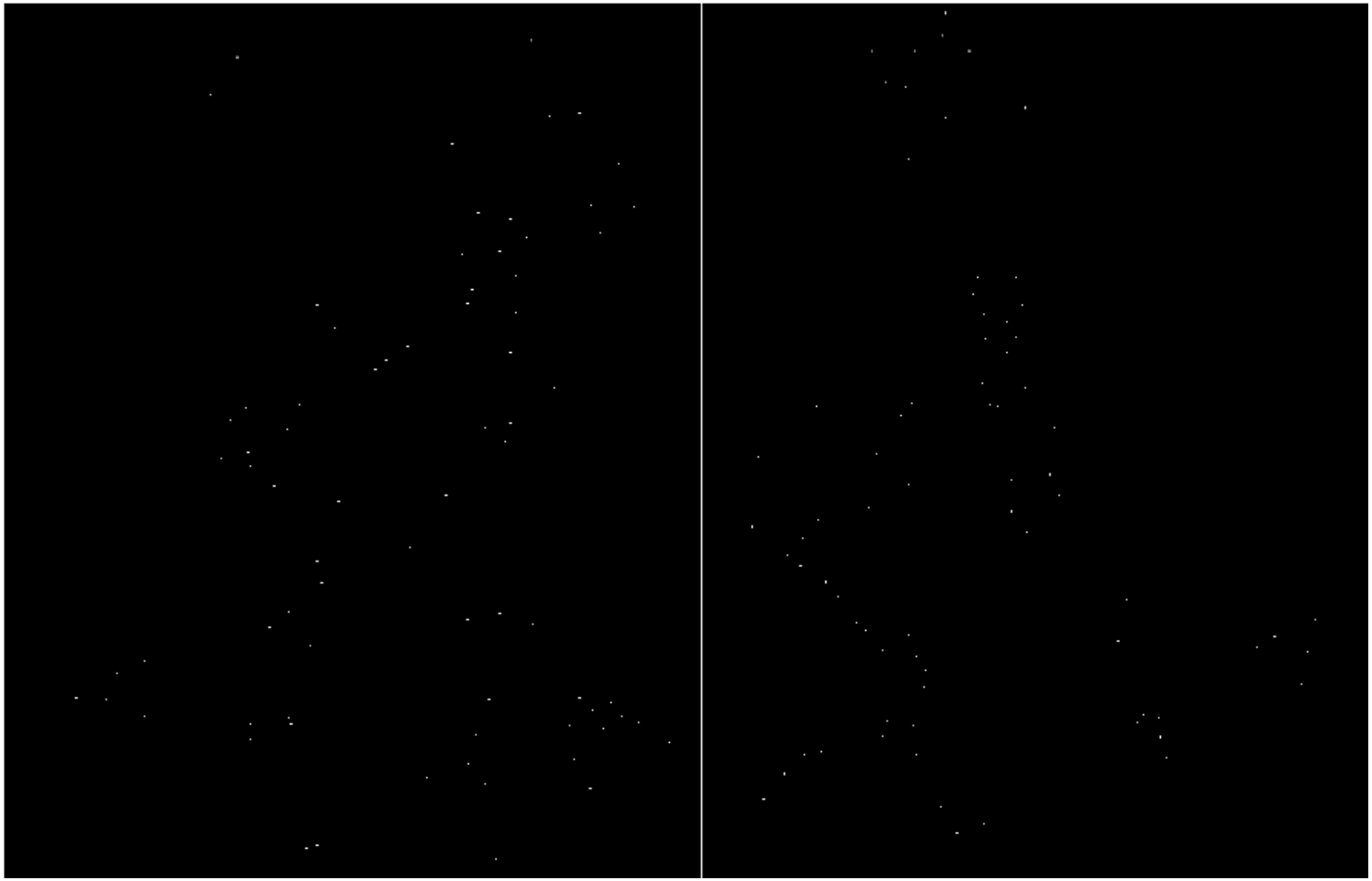
Example

- Find points with large corner response: $R >$ threshold



Example

- Take only the points of local maxima of R



Example

http://www.cim.mcgill.ca/ dparks/CornerDetector/harris.htm

Matching

Matching up Harris Points is the last step in the correspondence problem.

The most popular approach is Template Matching:

- Select a neighborhood (9 or 25 pixel region)
- Select a Harris point to match in image 1
- For all Harris points in image 2:
  - Compute the match correspondence
  - Select point using the best correspondence
- Often done in gray scale, but RGB variants possible

Correspondence

SSD:
The neighborhood will be reflected in the window function:
$w(u,v) = w_1(i,j)w_2(i+u,j+v)$, so

$$E_{SSD}(u,v) \equiv \sum_{i,j} w(u,v)\left[I_1(i+u,j+v) - I_0(i,j)\right]^2$$

NCC:
This is the normalized cross-correlation:

$$E_{SSD}(u,v) \equiv \frac{\sum_{i,j} w(u,v)\left[I_0(i,j) - \mu_0\right]\left[I_1(i+u,j+v) - \mu_1\right]}{\sqrt{\sum_{i,j} w(u,v)\left[I_0(i,j) - \mu_0\right]^2\left[I_1(i+u,j+v) - \mu_1\right]^2}}$$

where μ_0 and μ_1 are the mean intensity values.

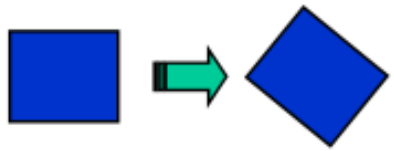
Correspondence

Not robust to all types of geometric changes.

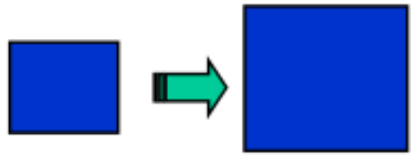
We want features to be detected despite geometric or photometric changes in the image: if we have two transformed versions of the same image, features should be detected in corresponding locations.

Invariance

- Geometric
  - **Rotation**



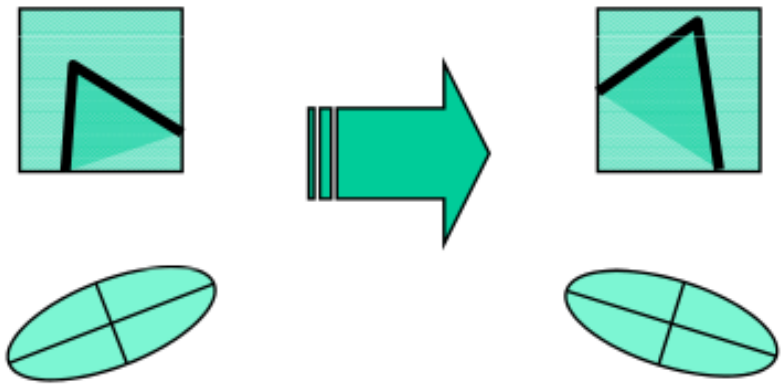
  - **Scale**



  - **Affine**
    valid for locally planar object

- Photometric
  - **Affine intensity change** ($I \rightarrow a\,I + b$)

Invariance

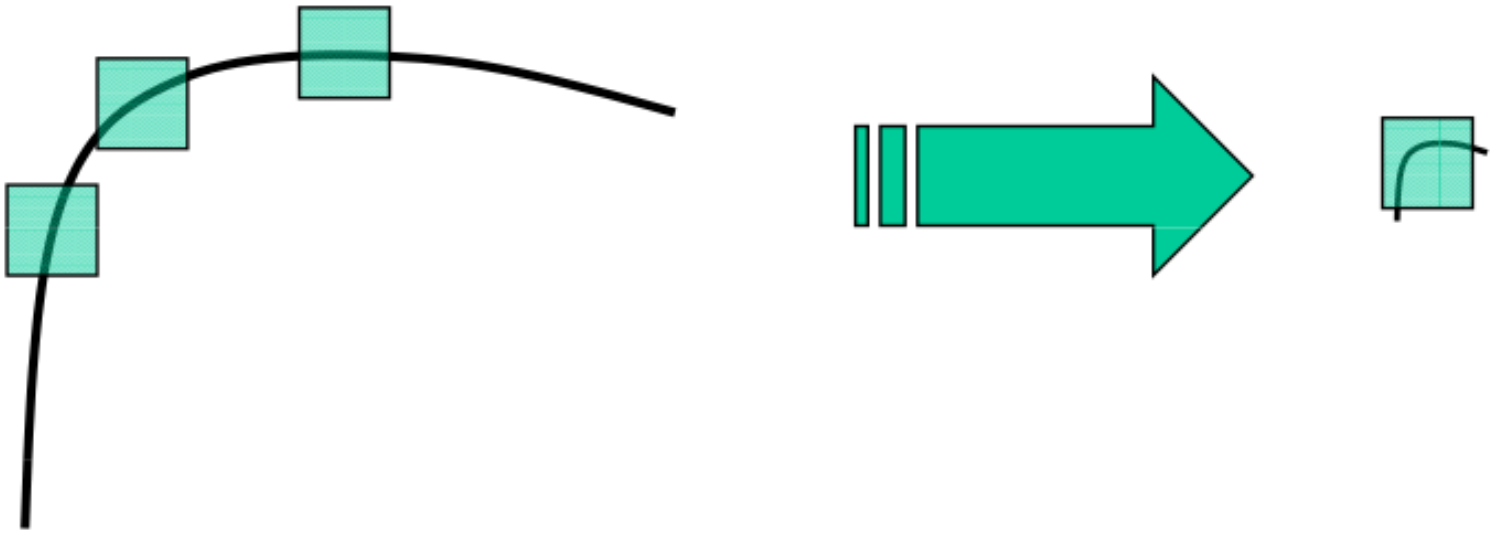


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response R is invariant to image rotation

Invariance

- But: non-invariant to *image scale*!



All points will be classified as edges

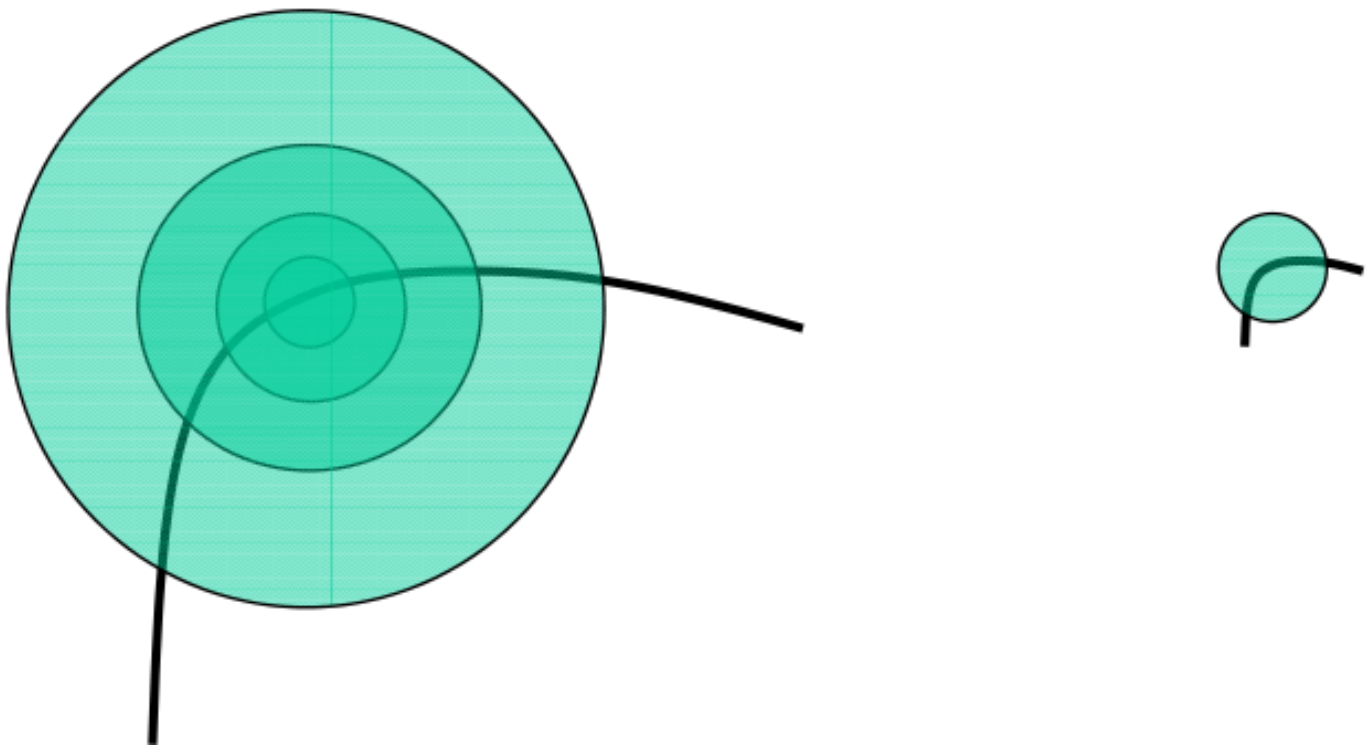
Corner !

Harris Summary

- Harris detector is an approach for detecting and extracting corners (i.e. points with high intensity changes in all directions)
- The detection is Invariant to Rotation
- Matching of Harris corners is done using Sum of Squared Differences, which is NOT rotation invariant !!!
- It is also NOT invariant to
  - scaling
  - small changes in viewpoint
  - illumination

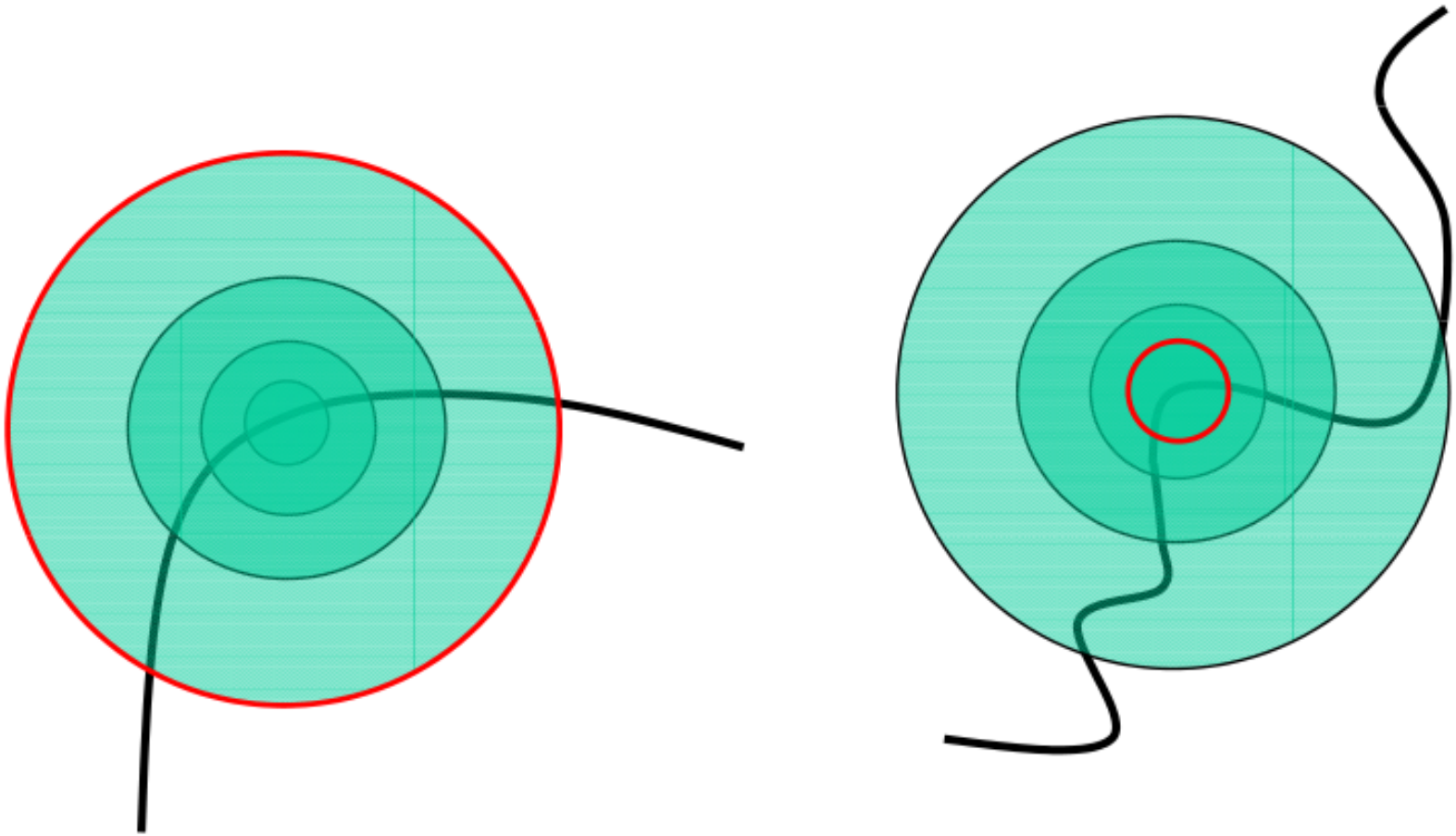
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection

The problem: how do we choose corresponding circles independently in each image?

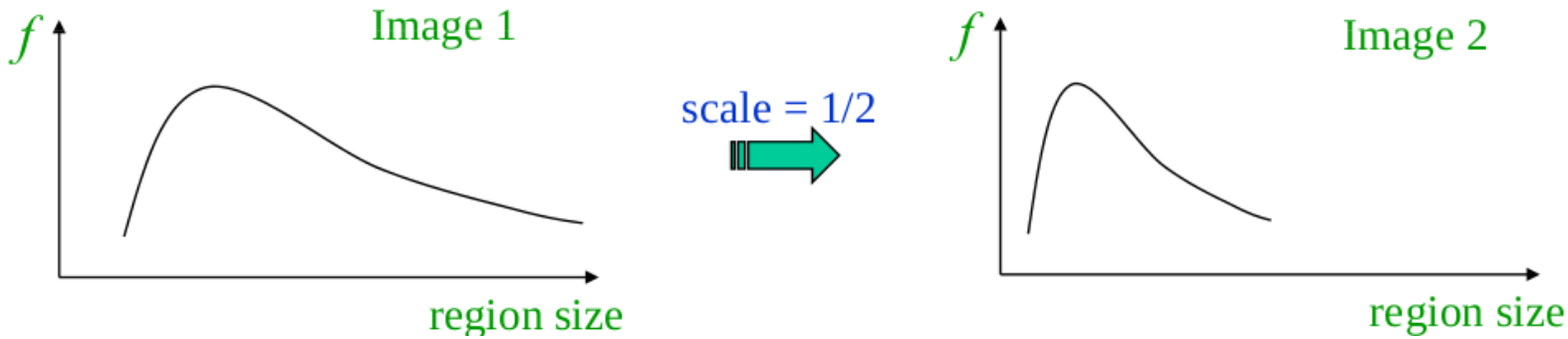


Scale Invariant Detection - Solution

Solution Overview:
Design a function on the region (circle), which is scale invariant (the same for corresponding regions, even if they are at different scales)

Example: average intensity. For corresponding regions (even of different sizes) it will be the same.

For a point in one image, we can consider it as a function of region size (circle radius)



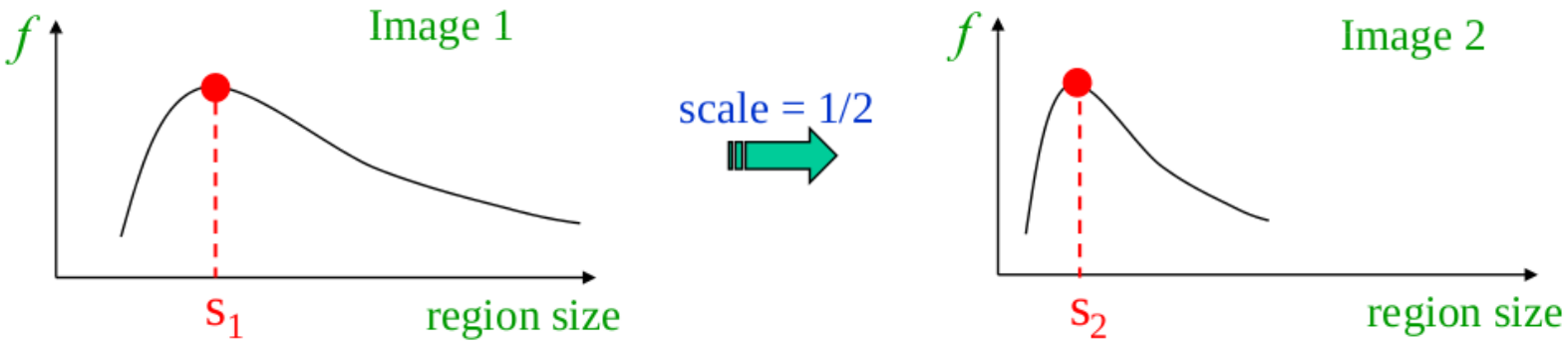
Scale Invariant Detection - solution

Common approach:

Take a local maximum of this function

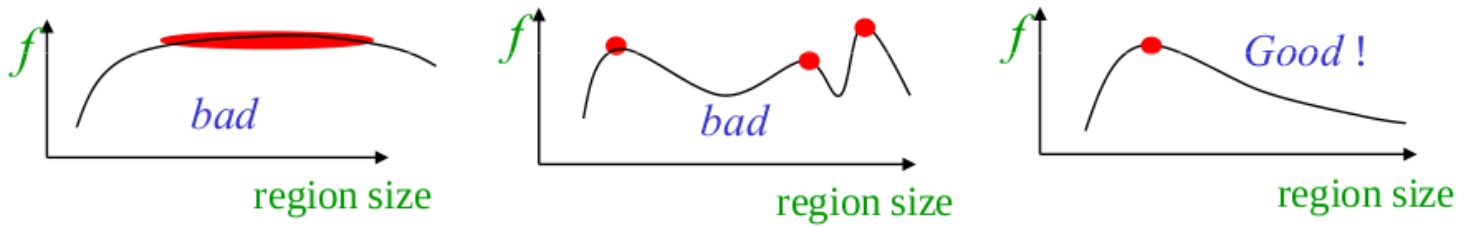
Observation: region size, for size which the maximum is achieved, should be invariant to image scale.

Important: this scale invariant region size is found in each image independently!



Scale Invariant Detection - solution

A good function for scale detection: has one stable sharp peak



For usual images: a good function would be a one which responds to contrast (sharp local intensity change)

Scale Invariant Detection - solution

- Function for determining scale

$$f = \text{Kernel} * \text{Image}$$

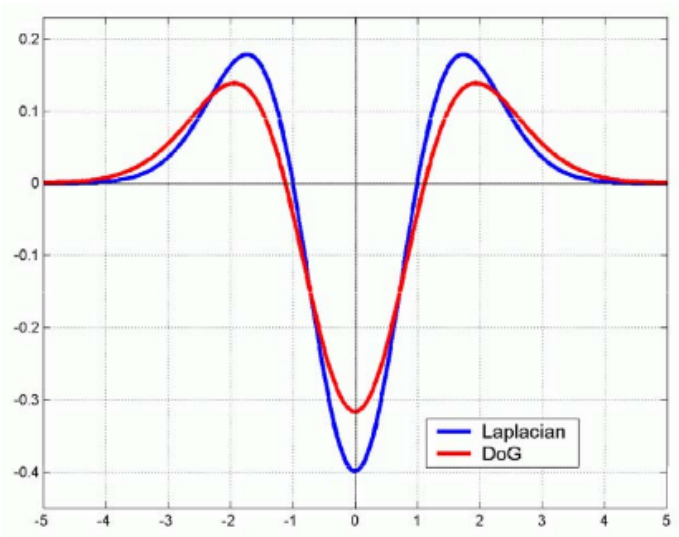
Kernel:

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

where Gaussian

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



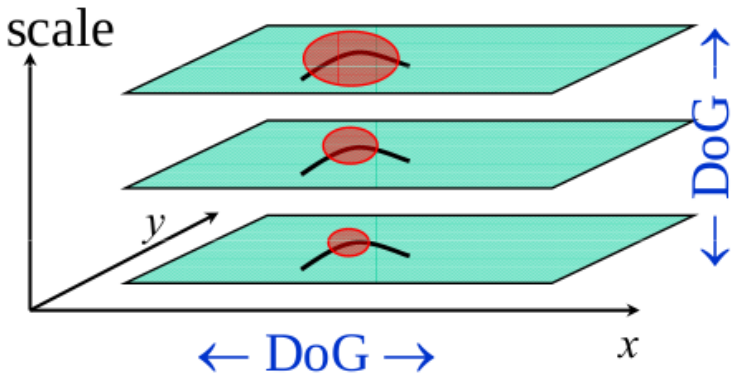
Note: this kernel is invariant to *scale* and *rotation*

Scale Invariant Detection

- SIFT

  Find local maximum of:

  – Difference of Gaussians in space and scale

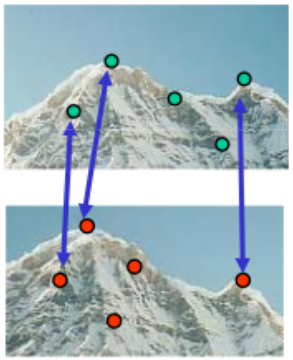


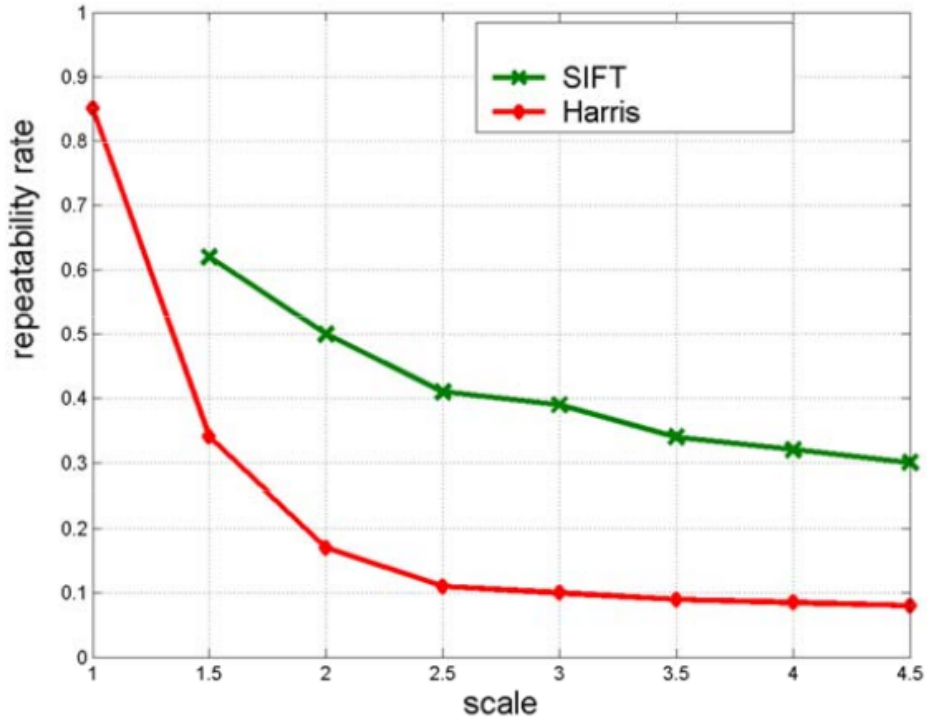
Scale Invariant Detection

- Experimental evaluation of detectors w.r.t. scale change

Repeatability rate:

$$\frac{\text{\# correspondences}}{\text{\# possible correspondences}}$$





Look at SIFT carefully ...

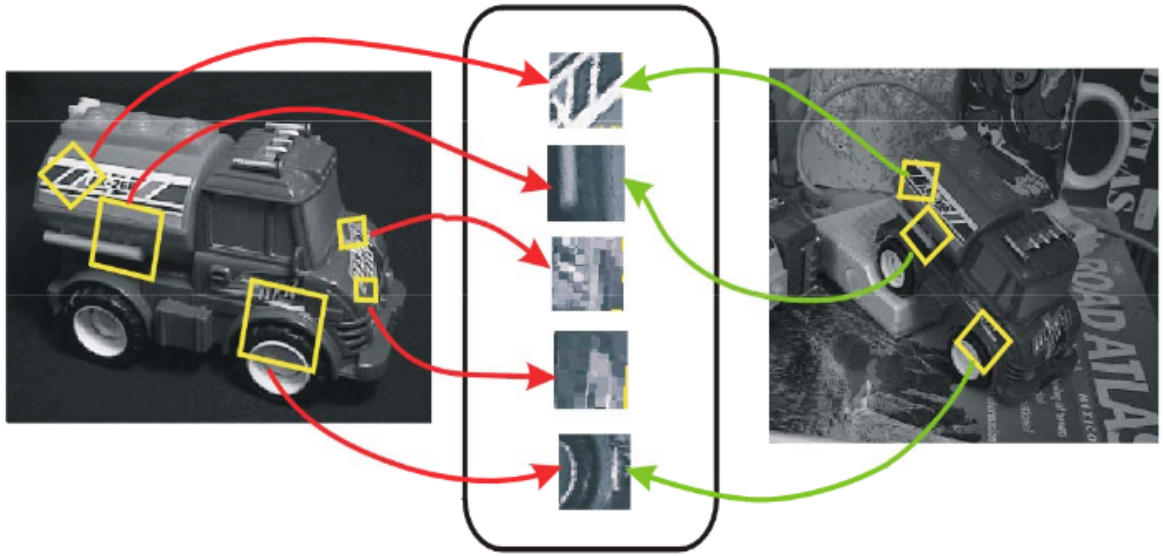
Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) is an approach for detecting and extracting local feature descriptors that are reasonably invariant to changes in:

- rotation
- scaling
- small changes in viewpoint
- illumination
- image noise

Invariant Local Features

Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and slight view-point and illumination



David Lowe's Method: SIFT

Scale Invariant Feature Transform

SIFT Transforms an image into a large collection of feature vectors each invariant to translation, rotation, scaling and illumination.

SIFT Keypoints extracted from a set of reference images and stored in a database.

An object is recognized by comparing each feature in the new image to the database.

Detection stages for SIFT features

- Scale-space extrema detection
- Keypoint localization
- Orientation assignment
- Generation of keypoint descriptors.

From Lowe's Paper:

1. **Scale-space extrema detection:** The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.

2. **Keypoint localization:** At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.

3. **Orientation assignment:** One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

4. **Keypoint descriptor:** The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

Some definitions

Recall the Gaussian Kernel

$$g_t(x, y) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t}$$

we define scale space kernel

$$L(x, y; t) = (g_t * f)(x, y)$$

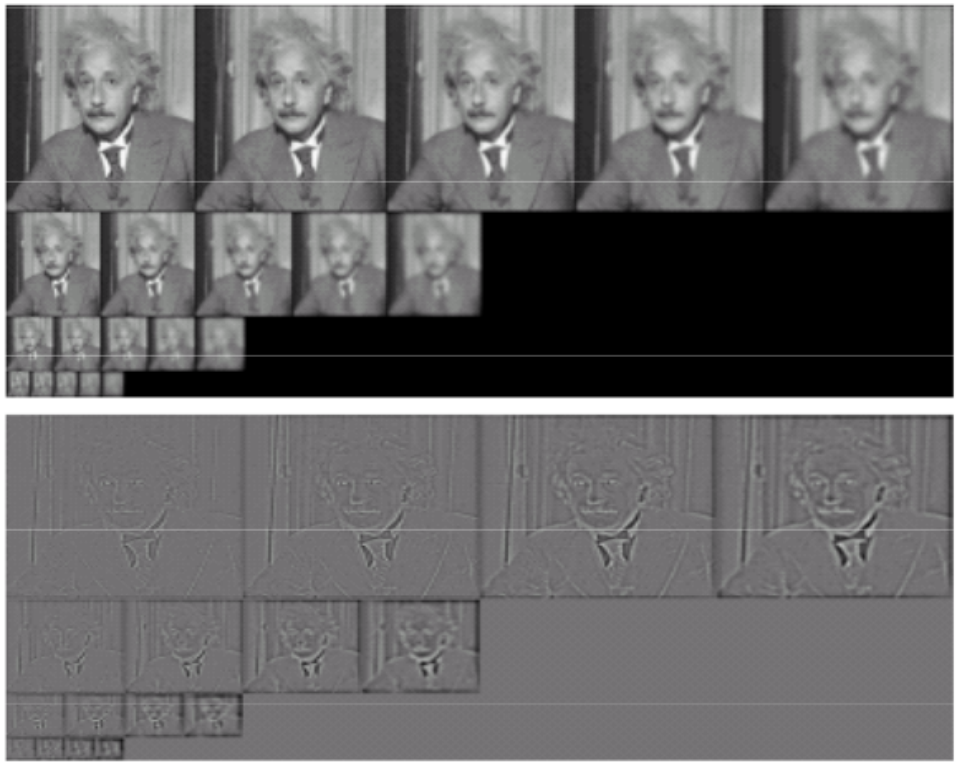
This provides a variable scale at a particular resolution. Define

$$\underbrace{D(x, y, \sigma) \equiv L(x, y, k\sigma) - L(x, y, \sigma)}_{\text{Difference of Gaussians appplied to the image}}$$

We define factor of two changes in scale as different Octaves. This is efficiently implemented using down-sampling.

Scale space extrema detection

- Gaussian blurred images grouped by octave
- Difference of Gaussian images (DoG) grouped by octave



Scale space extrema detection

Scale-space representation $L(x,y;t)$ at scale $t = 0$, corresponding to the original image f

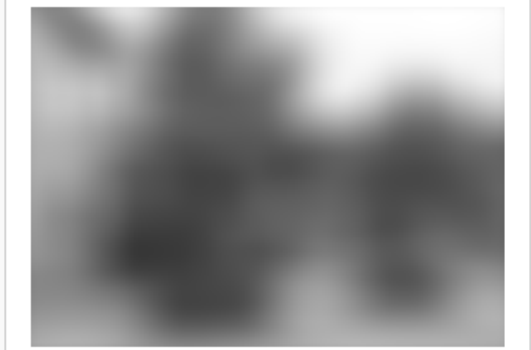
Scale-space representation $L(x,y;t)$ at scale $t = 1$

Scale-space representation $L(x,y;t)$ at scale $t = 4$

Scale-space representation $L(x,y;t)$ at scale $t = 16$

Scale space extrema detection

Scale-space representation $L(x,y;t)$ at scale $t = 64$



Scale-space representation $L(x,y;t)$ at scale $t = 256$

Again recall that the scale space is produced via:

$$L(x, y; t) = (G_t * f)(x, y)$$

Again, different octaves are factor of two changes in σ and in practice are downsamples.

Scale Space and Diffusion

Gaussian

$$G(x, y, t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t}$$

Difference of Gaussians

$$\frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \approx \frac{\partial G}{\partial \sigma} = \sigma \left(\frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right) \equiv \sigma \nabla^2 G$$

From:

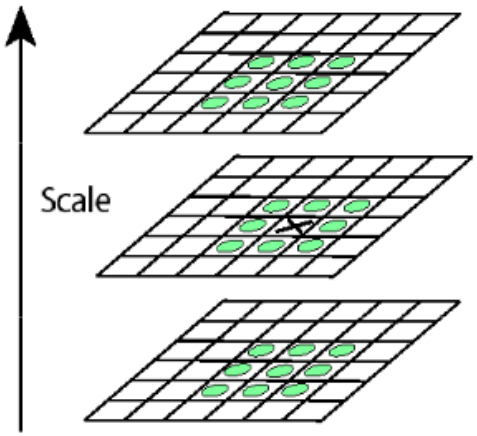
$$L(x, y; t) = (G_t * f)(x, y)$$

we have

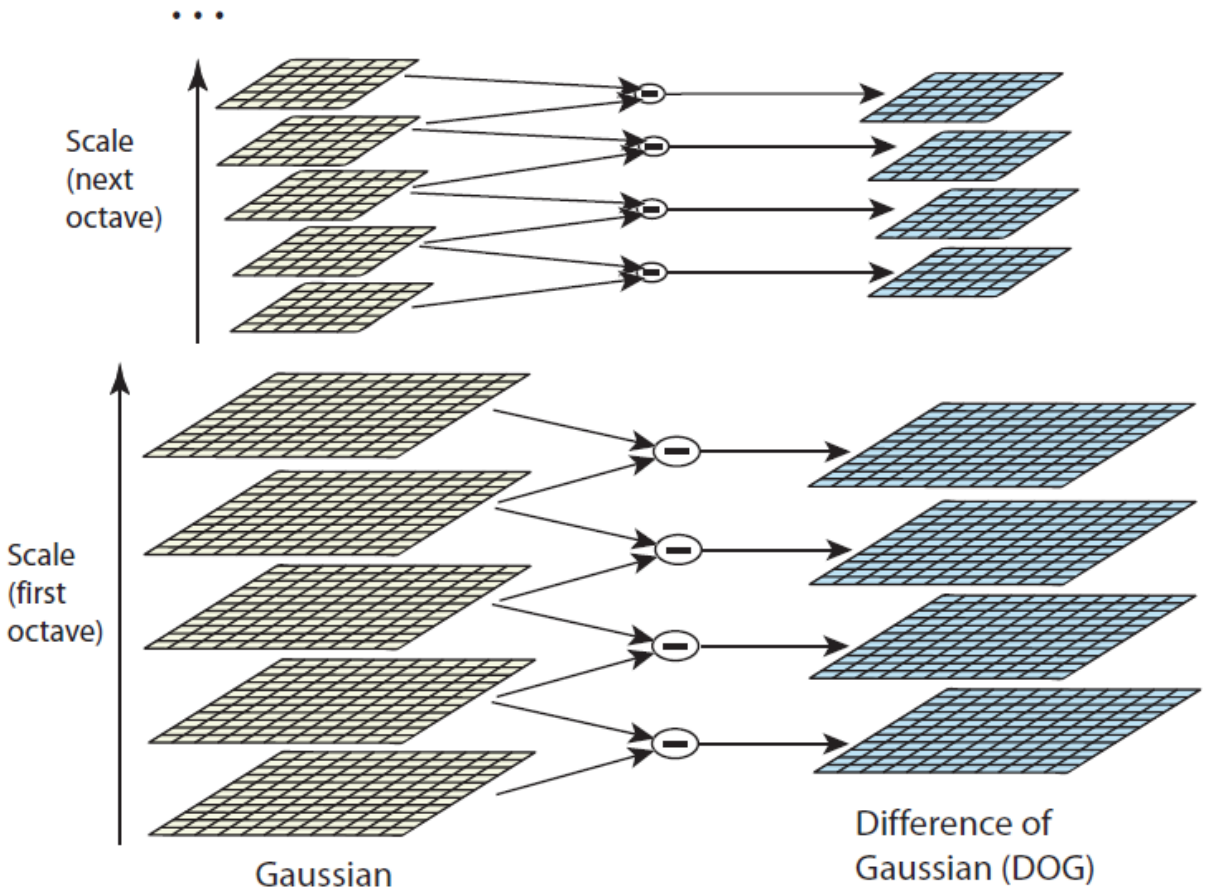
$$\underbrace{D(x, y, \sigma) \equiv L(x, y, k\sigma) - L(x, y, \sigma)}_{\text{Difference of Gaussians appplied to the image}} \approx (k-1)\sigma^2(\nabla^2 G * f)(x, y)$$

Scale space extrema detection

- SIFT keypoints are identified as local maxima or minima of the DoG images across scales.
- Each pixel in the DoG images is compared to its 8 neighbors at the same scale, plus the 9 corresponding neighbors at neighboring scales.
- If the pixel is a local maximum or minimum, it is selected as a candidate keypoint.



Scale space extrema detection



Local Extremals

Local Max and Min of $D(x, y, \sigma)$:

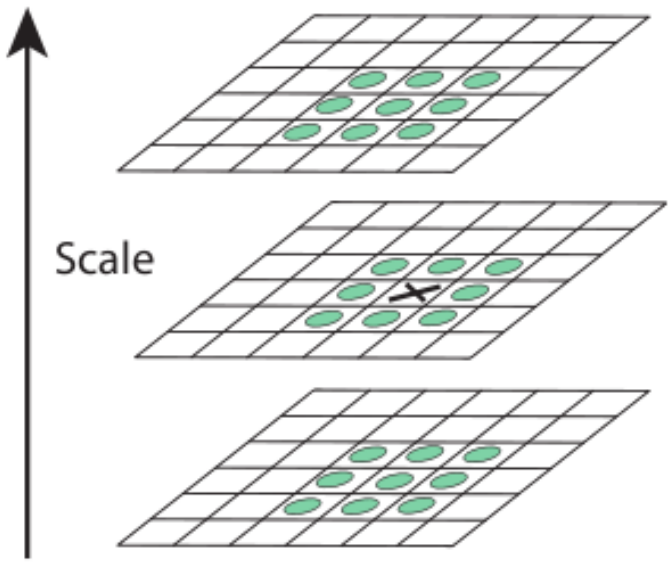
- Each sample point is compared to its eight neighbors in the current image scale and the nine neighbors in the scale above and below.
- No min spacing that will detect all extemals.

Sample details:

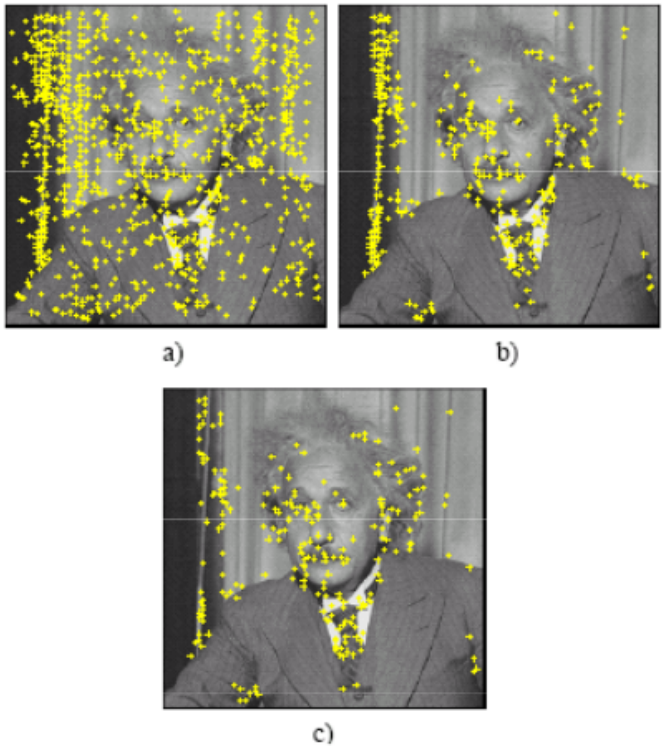
- Vary σ and determine the number of repeated keypoints.
- 3 scales per octave.
- See Lowe's paper for details.

Keypoint localization:

- Fit a quadratic to $D(x, y, \sigma)$
- $\hat{x} = -D_{xx}^{-1} D_x$



Keypoint localization



- (a) Maxima of DoG across scales
- (b) Remaining keypoints after removal of low contrast points
- (c) Remaining keypoints after removal of edge reponses

Keypoint removal

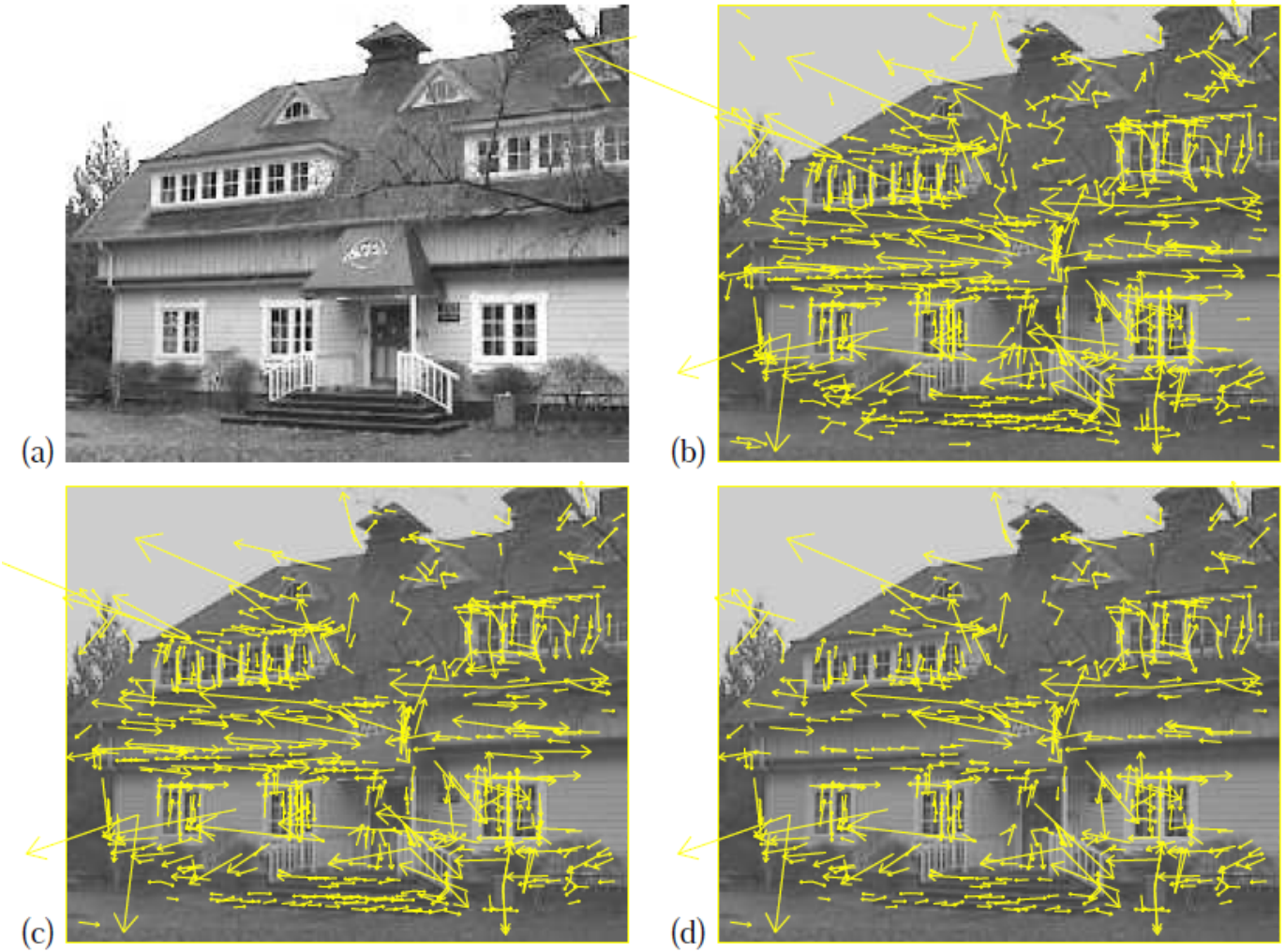


Figure 5: This figure shows the stages of keypoint selection. (a) The 233x189 pixel original image. (b) The initial 832 keypoints locations at maxima and minima of the difference-of-Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. (c) After applying a threshold on minimum contrast, 729 keypoints remain. (d) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures.

Keypoint orientation assignment

For orientation, we set σ to the be value at the keypoint.

Compute the gradient magnitude and orientation:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \arctan\left[\frac{(L(x,y+1) - L(x,y-1))}{(L(x+1,y) - L(x-1,y))}\right]$$

An orientation histogram is formed from the gradient orientations of sample points within a region around the keypoint.

Keypoint orientation assignment

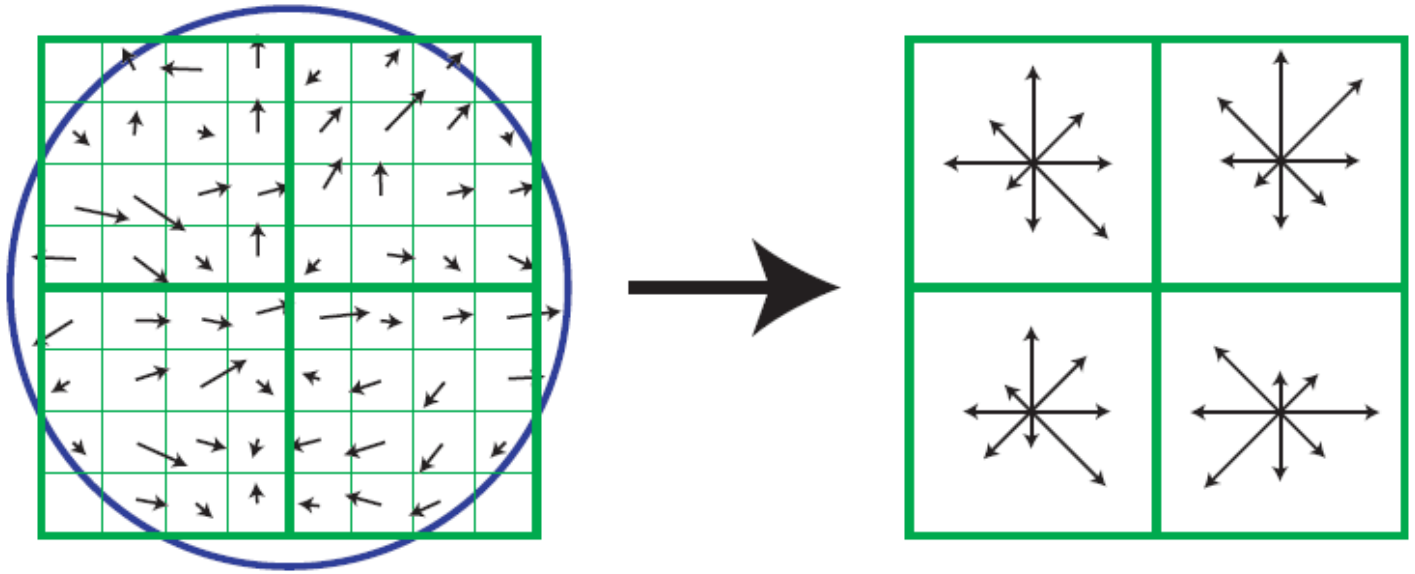
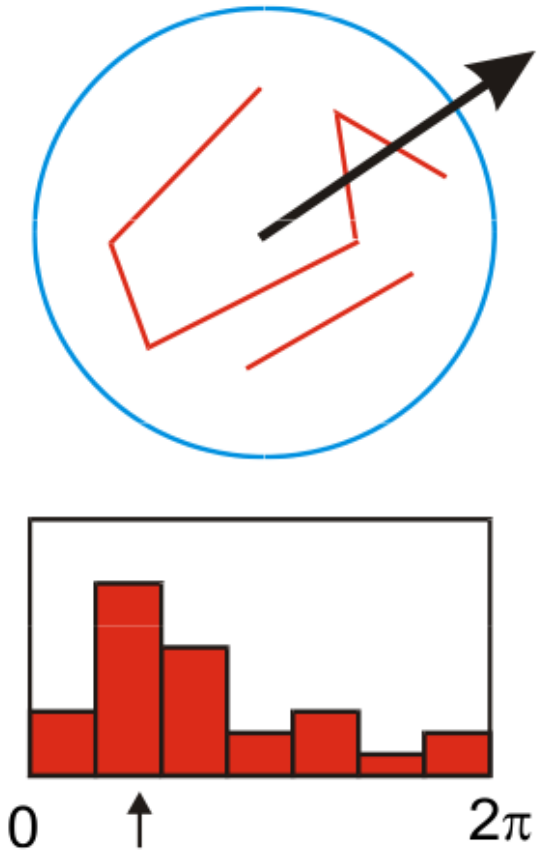


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 descriptors computed from a 16x16 sample array.

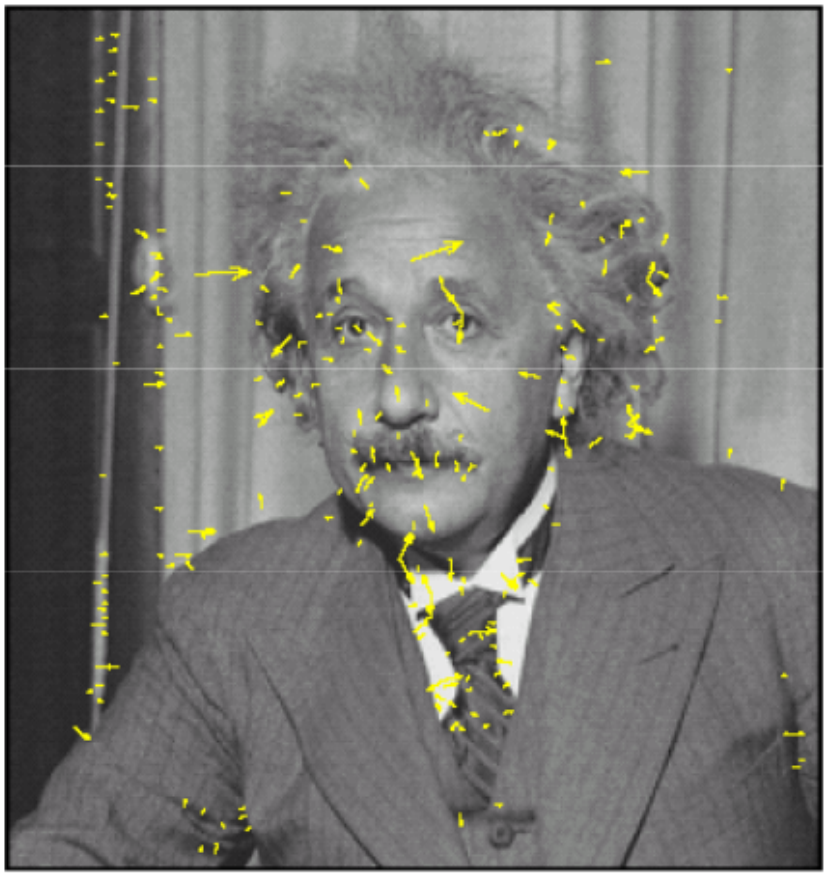
Keypoint orientation assignment

- To determine the keypoint orientation, a gradient orientation histogram is computed in the neighborhood of the keypoint.
- Peaks in the histogram correspond to dominant orientations. If more than one peak is found, a separated feature is assigned to the same point location.
- All the properties of the keypoint are measured relative to the keypoint orientation, this provides invariance to rotation.



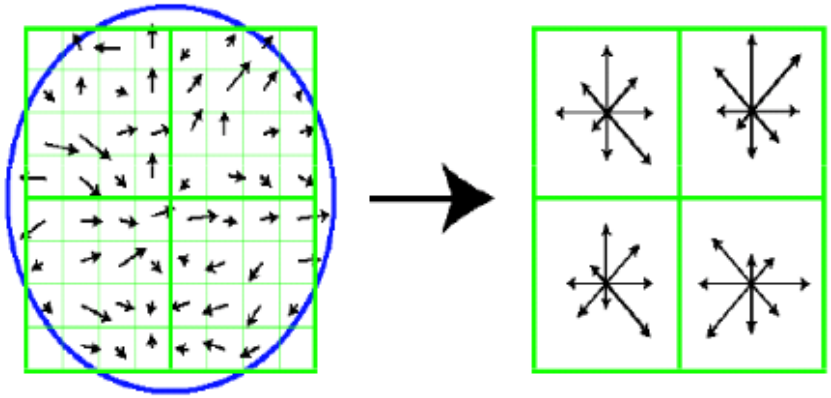
Keypoint localization

- Final keypoints with selected orientation and scale



Keypoint Descriptor

- A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left.



- These samples are then accumulated into orientation histograms summarizing regions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region.
- The descriptor is formed from a vector containing the values of all the orientations histogram entries, corresponding to the arrows of the right side.
- This vector is then normalized to enhance invariance to changes in illumination.

Advantages of SIFT features

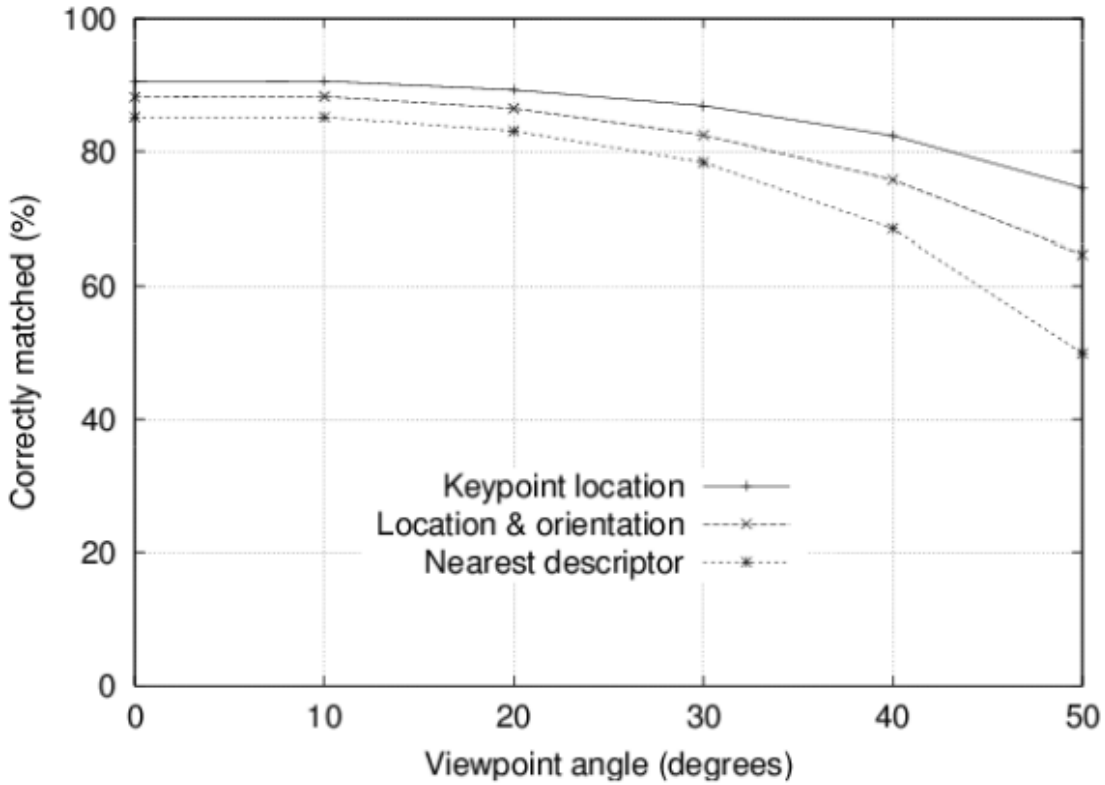
- Locality: features are local, so robust to occlusion and clutter (no prior segmentation)
- Distinctiveness: individual features can be matched to a large database of objects
- Quantity: many features can be generated for even small objects
- Efficiency: close to real time performance

Object Recognition

Now that you can gleen features, how can you apply this?

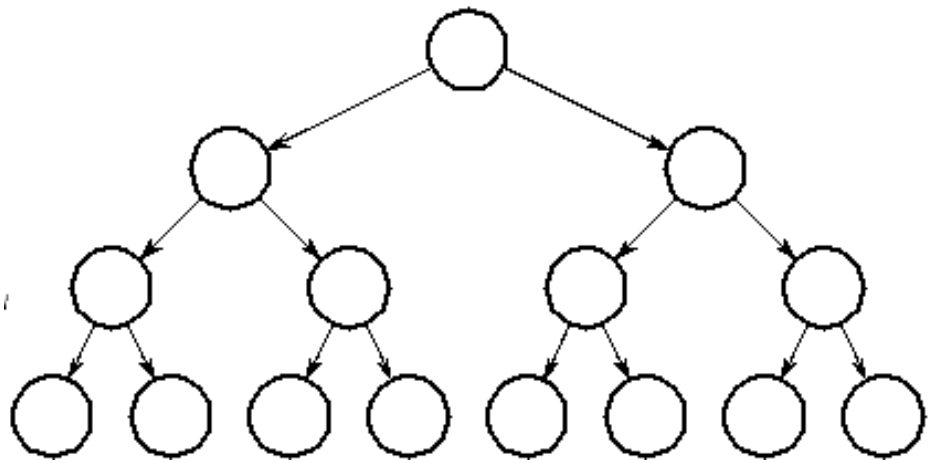
Object Recognition

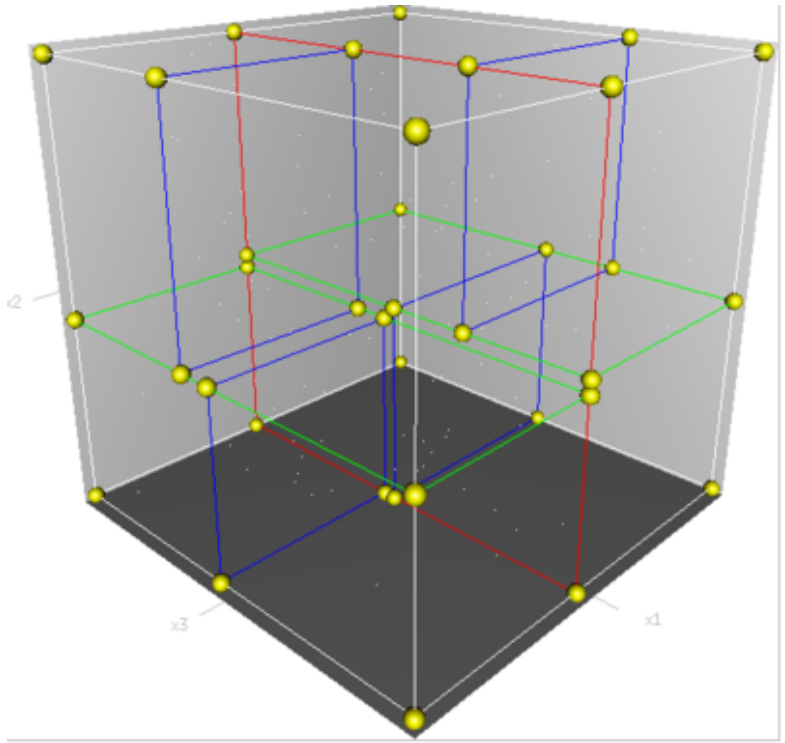
- Match features after random change in image scale & orientation, with 2% image noise, and affine distortion
- Find nearest neighbor in database of 30,000 features



Searching

- Best Bin First Search Method (BBF)
- Modification of k-d tree algorithm - a space partitioning data structure.





Feature matching

- Best Bin First Search Method (BBF)
- Modification of k-d tree algorithm a space partitioning data structure.

Given a list of *n* points, the following algorithm will construct a balanced *k*d-tree containing those points.

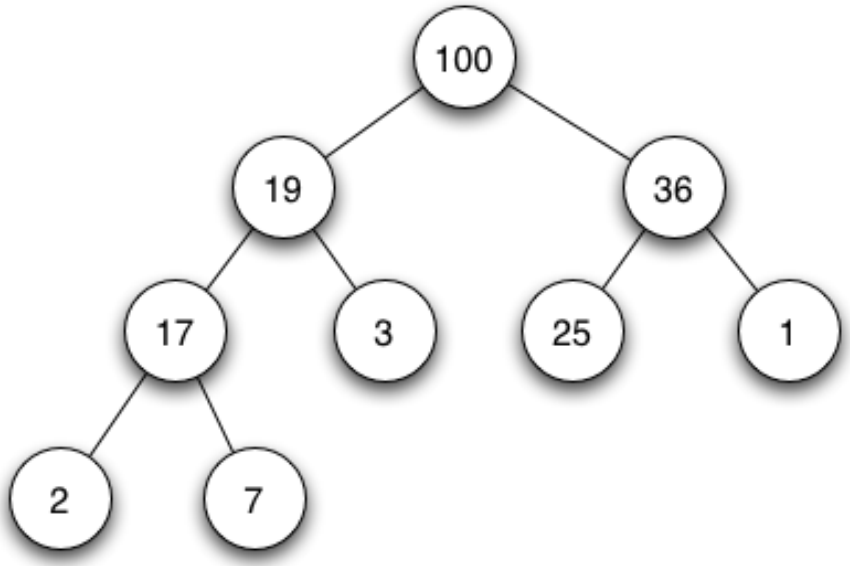
```
function kdtree (list of points pointList, int depth)
{
    if pointList is empty
        return nil;
    else
    {
        // Select axis based on depth so that axis cycles through all valid values
        var int axis := depth mod k;

        // Sort point list and choose median as pivot element
        select median by axis from pointList;

        // Create node and construct subtrees
        var tree_node node;
        node.location := median;
        node.leftChild := kdtree(points in pointList before median, depth+1);
        node.rightChild := kdtree(points in pointList after median, depth+1);
        return node;
    }
}
```

Feature matching

- Best Bin First Search Method (BBF)
- Modification of k-d tree algorithm a space partitioning data structure.
- Requires the use of a heap based priority queue. Heap: tree based data structure with the heap property.



Feature matching

- Best Bin First Search Method (BBF)
- Modification of k-d tree algorithm a space partitioning data structure.
- Requires the use of a heap based priority queue.
- Nearest neighbor in the database of key points
  - Nearest in the sense of Euclidean distance.
  - Distance ratio $= 0.8$
  - Only 200 nearest

Cluster Identification

- Hough Transform Voting
  - Bins
- Each feature votes for object poses
  - Location
  - Orientation
  - Scale
- Hashing used for search.
  - At least three entries per bin

Model Verification

Affine transformation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Solve for values

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \end{bmatrix}$$

Solve via pseudo-inverse

$$A^T A x = A^T b \quad \Rightarrow \quad x = (A^T A)^{-1} A^T b$$

Vision

SIFT is only the beginning...