



上海科技大学  
ShanghaiTech University

## CS283: Robotics Fall 2019: Vision II

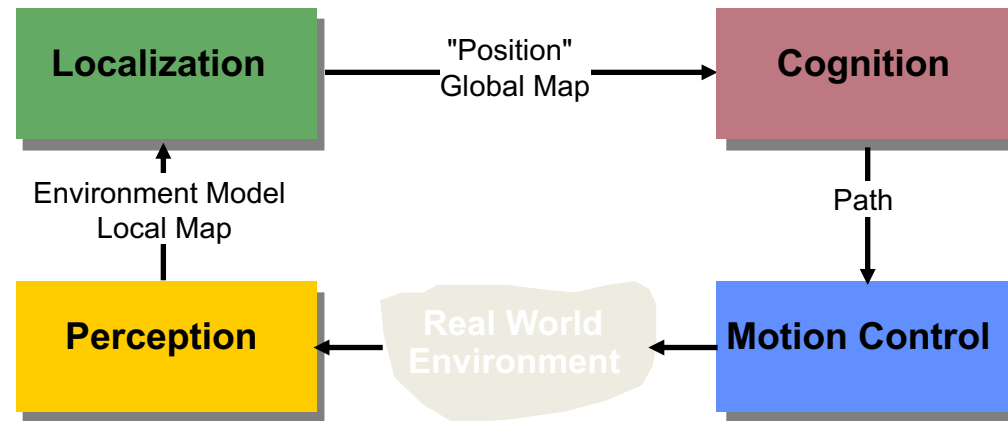
---

Sören Schwertfeger / 师泽仁

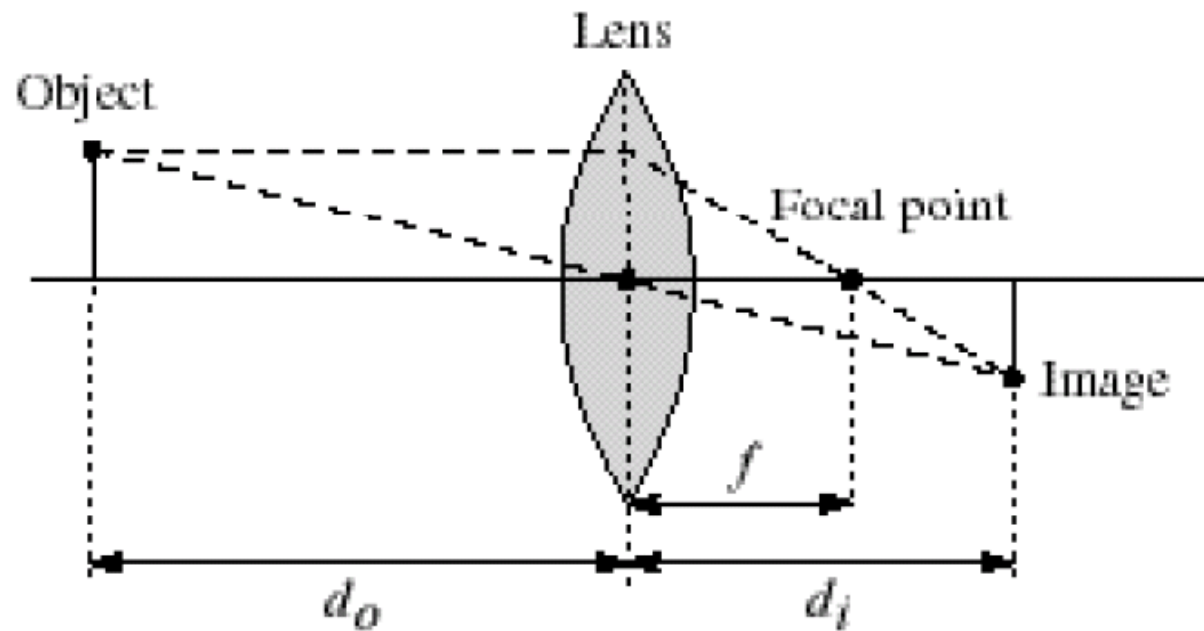
ShanghaiTech University

# VISION

---



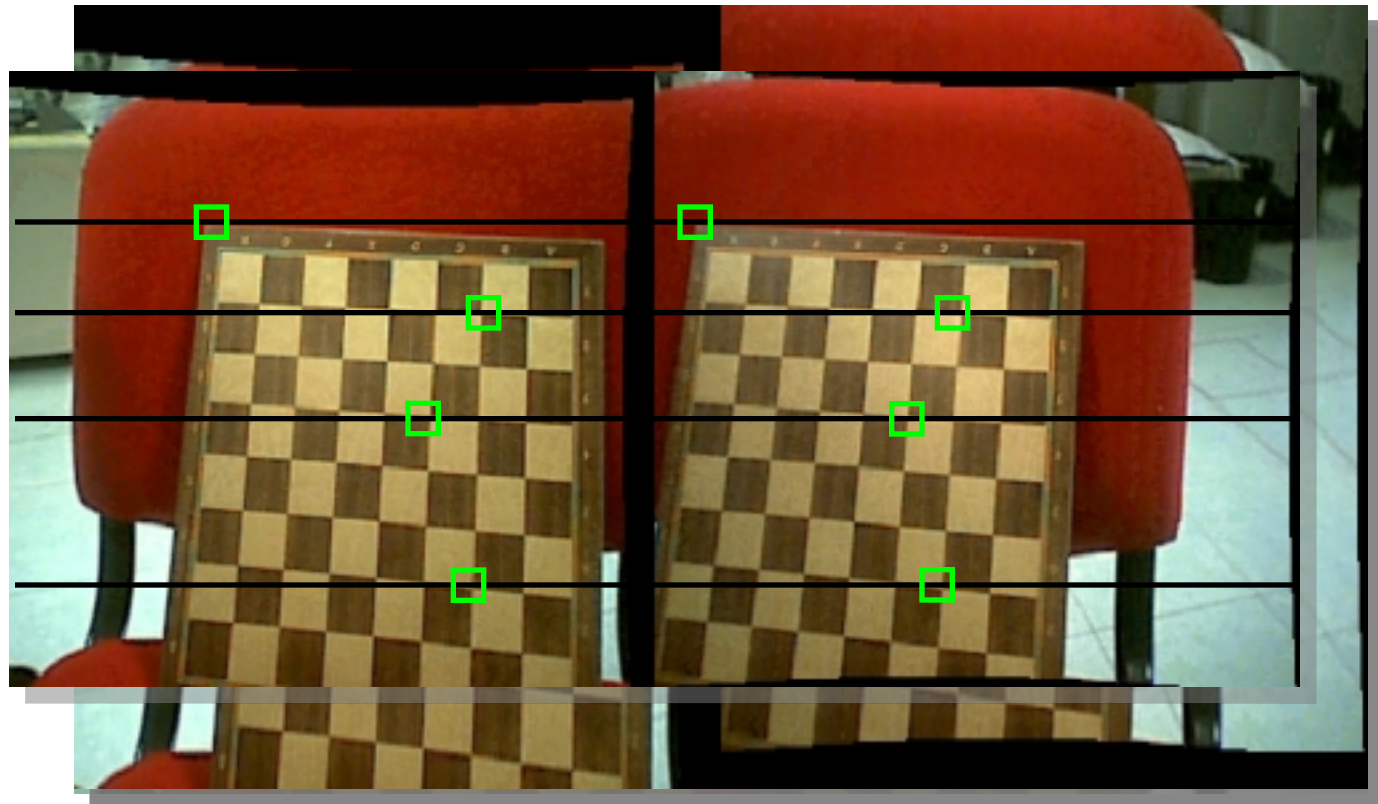
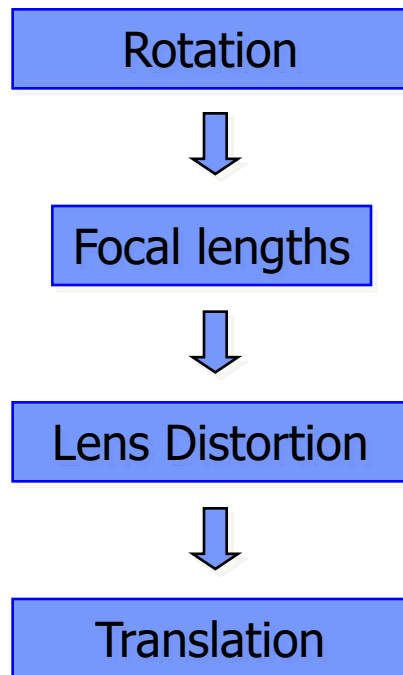
# Thin lenses



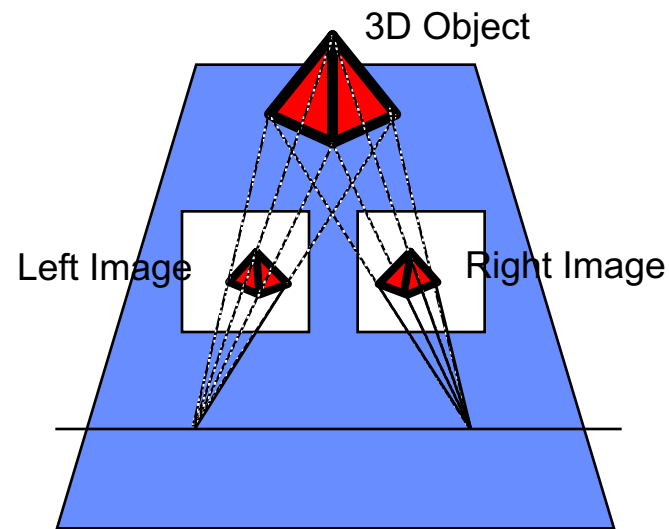
- Thin lens equation:  $\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$ 
  - Any object point satisfying this equation is in focus
  - This formula can also be used to estimate roughly the distance to the object ("Depth from Focus")

# 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)



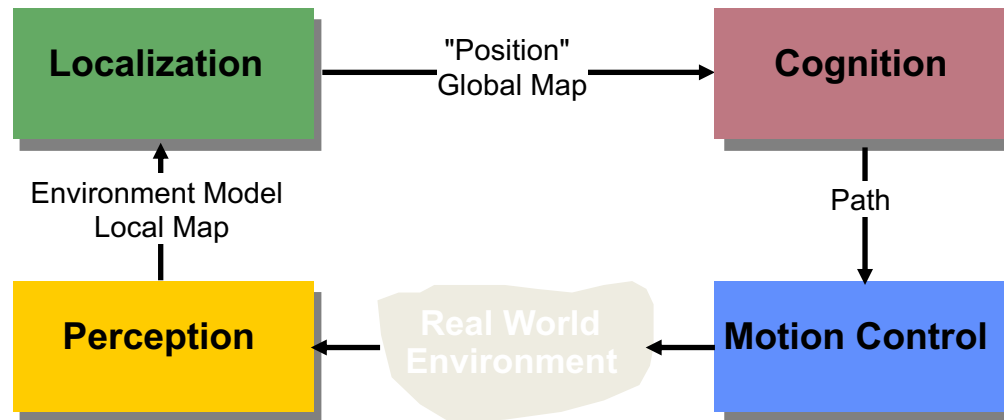
# Stereo Vision - summary



1. Stereo camera calibration -> compute camera relative pose
2. Epipolar rectification -> align images
3. Search correspondences
4. Output: compute stereo triangulation or disparity map
5. Consider baseline and image resolution to compute accuracy!

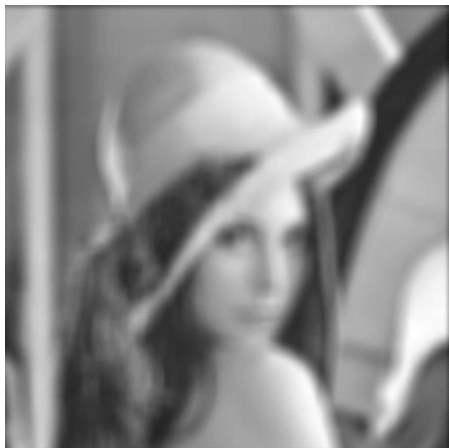
# VISION

## Image Processing

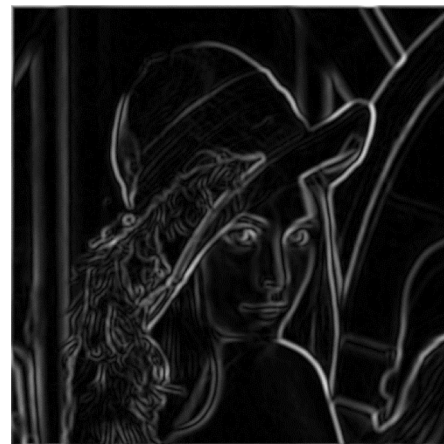


# Image filtering

- Filter: frequency domain processing where “filtering” refers to the process of accepting or rejecting certain frequency components. E.g.:
  - Lowpass filter: pass only low frequencies => blur (smooth) an image
  - **spatial filters** (also called masks or kernels): same effect



Lowpass filtered image



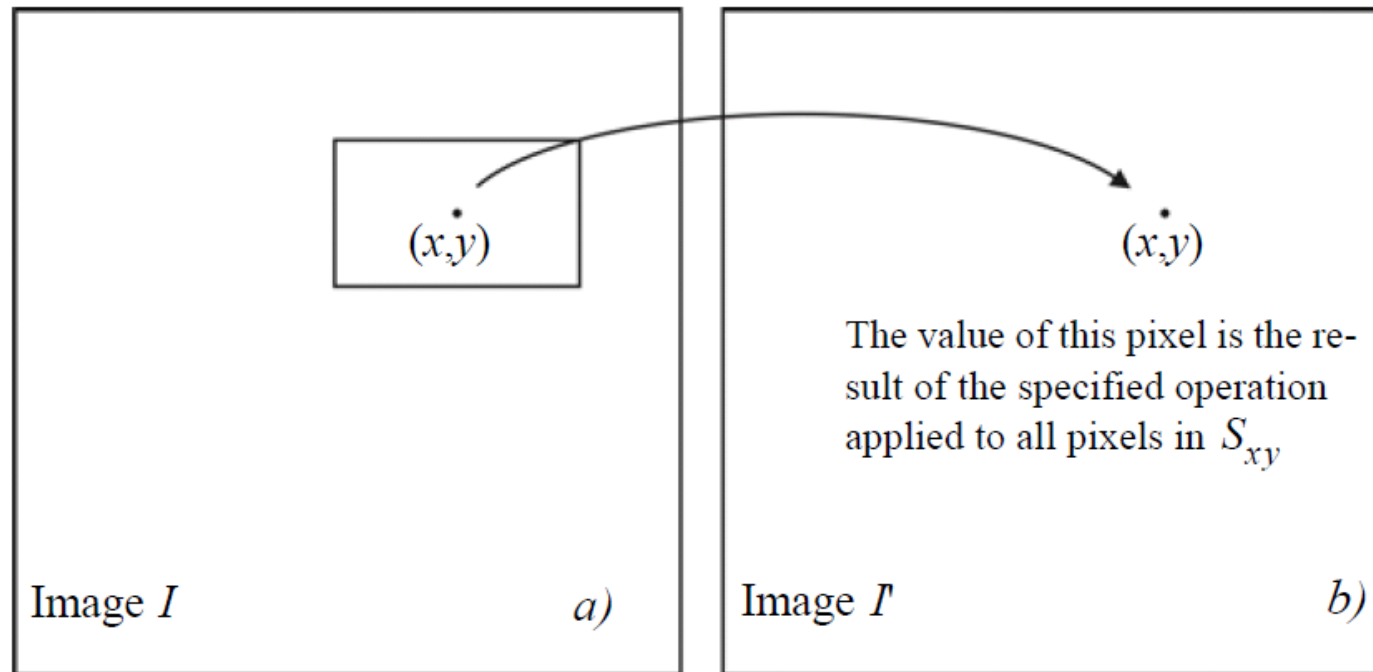
Highpass filtered image



Lena: Image processing standard test picture (512x512) since 1972

# Spatial filters

- Let  $S_{xy}$  denote the set of coordinates of a neighborhood centered on an arbitrary point  $(x,y)$  in an image  $I$
- Spatial filtering generates a corresponding pixel at the same coordinates in an output image  $I'$  where the value of that pixel is determined by a specified operation on the pixels in  $S_{xy}$



- For example, an averaging filter is:
- $$I' = \frac{1}{mn} \sum_{(r, c) \in S_{xy}} I(r, c)$$

# Linear filters

- In general, linear spatial filtering of an image with a filter  $w$  of size  $m \times n$  is given by the expression

$$I'(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) \cdot I(x+s, y+t)$$

where  $m=2a+1$  and  $n=2b+1$  are usually assumed odd integers. The filter  $w$  is also called **kernel**, **mask**, or **window**.

- As observed in this formula, linear filtering is the process of moving a filter mask over the entire image and computing the sum of products at each location. In signal processing, this particular operation is also called correlation with the kernel  $w$  or alternatively, convolution with the kernel  $w$
- The operation of convolution with the kernel  $w$  can be written in a more compact way as

$$I'(x, y) = w(x, y) * I(x, y)$$

where  $*$  denotes the operator of convolution

# Smoothing filters (1)

- A constant averaging filter yields the standard average of all the pixels in the mask. For a 3x3 mask this writes:

$$w = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- where notice that all the coefficients sum to 1. This normalization is important to keep the same value as the original image if the region by which the filter is multiplied is uniform.



This example was generated with a 21x21 mask

# Smoothing filters (2)

- A Gaussian averaging write as

$$G_{\sigma}(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

- To generate, say, a 3x3 filter mask from this function, we sample it about its center. For example, with  $\sigma=0.85$ , we get

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

- Very popular: Such low-pass filters effectively removes high-frequency noise =>
- First derivative and especially the second derivative of intensity far more stable
- Gradients and derivatives very important in image processing =>
- Gaussian smoothing preprocessing popular first step in computer vision algorithms

# Smoothing Example: SIST Lobby Scan

- Scanner: Faro Focus 3D X330
- 1 scan 20265 x 8533 pixel – about 173 million pixel
- Including color information
- Scan time: about 15 minutes





# ShanghaiTech University

8 Scans of the lobby of the new SIST building

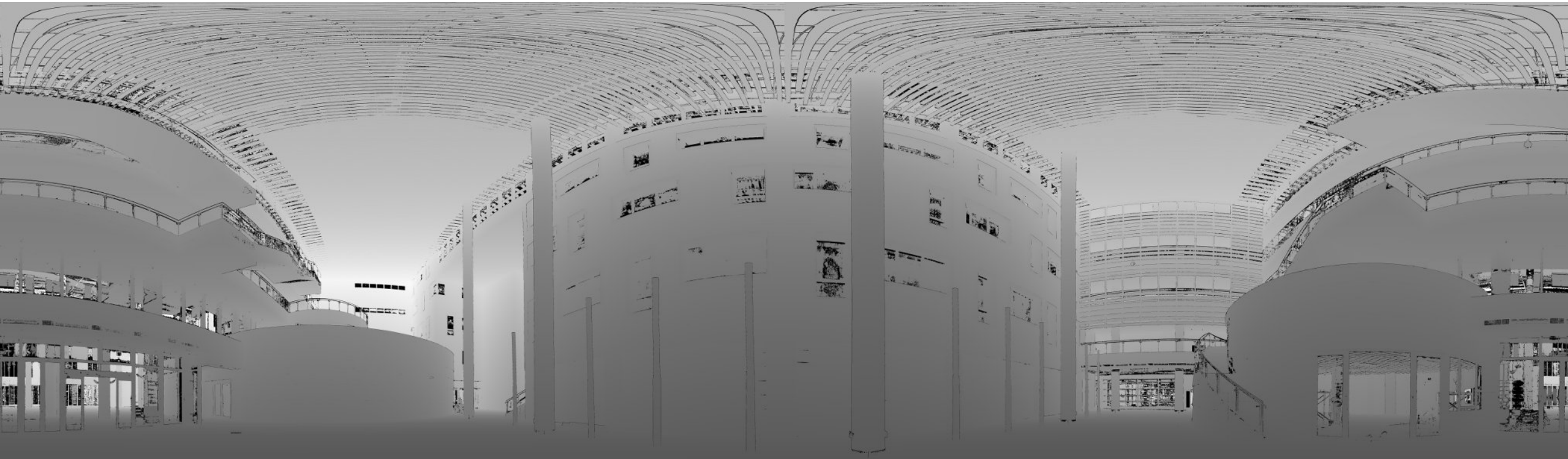
More than 400 million colored points

Rough alignment by hand and fine registration using ICP

# SIST Lobby: RGB image 360 degrees



# SIST Lobby: Depth image 360 degrees



# Depth image smoothing: kernel size 15, std. deviation 4

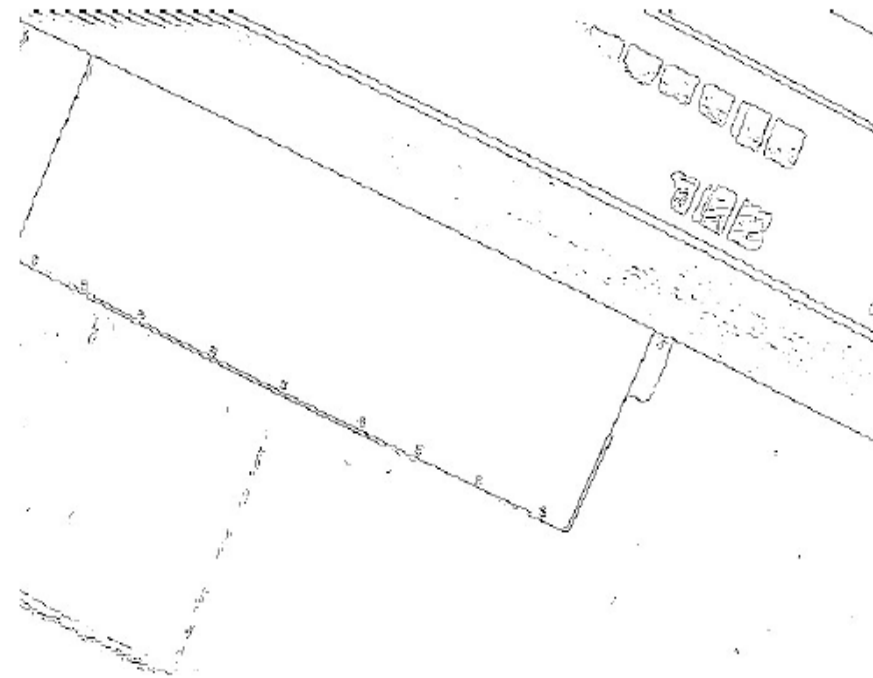
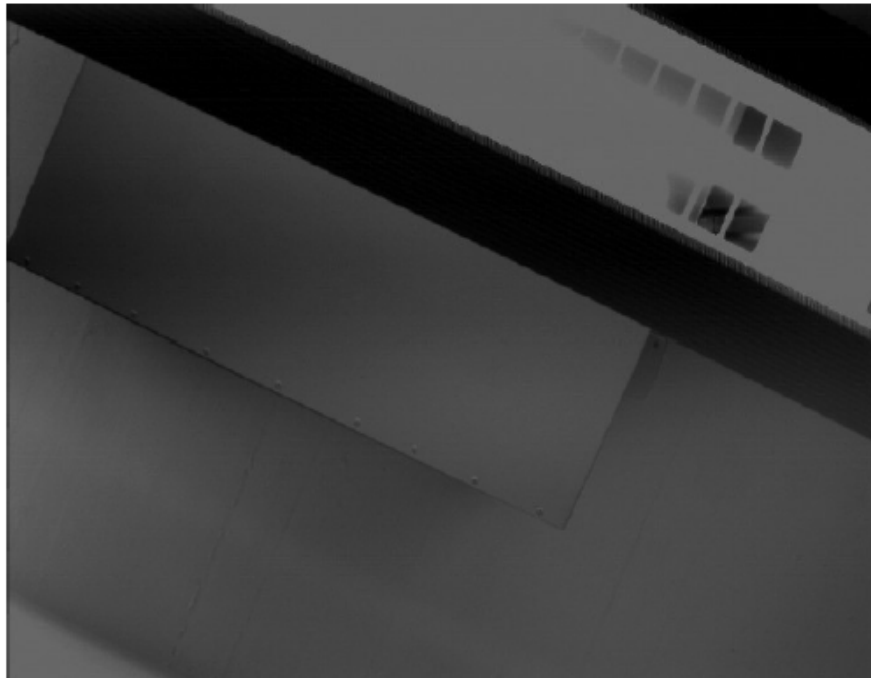


# Depth image smoothing: kernel size 255, std. deviation 20



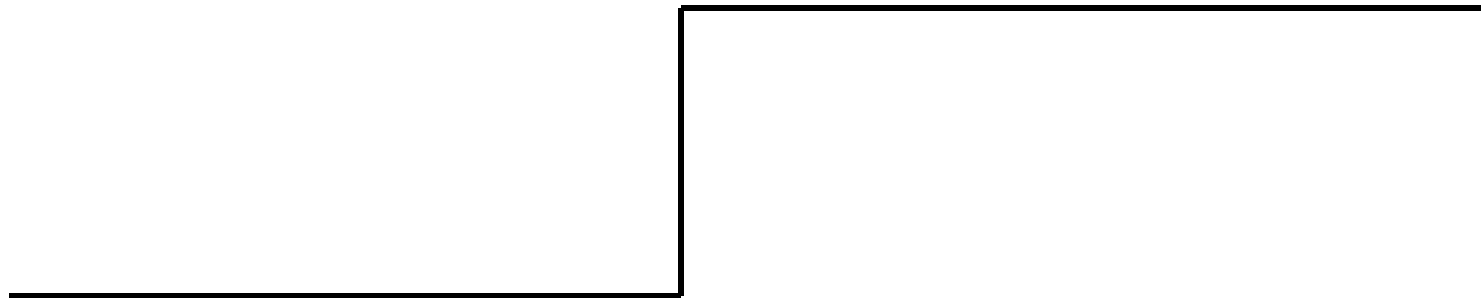
# Edge Detection

- Ultimate goal of edge detection
  - an idealized line drawing.
- Edge contours in the image correspond to important scene contours.



# Edge is Where Change Occurs

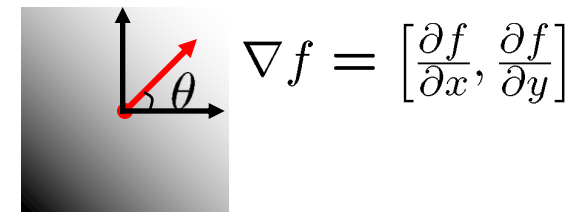
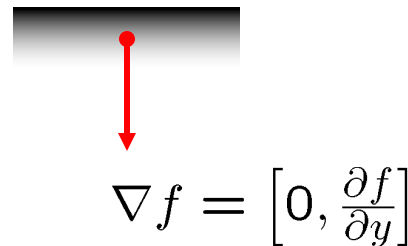
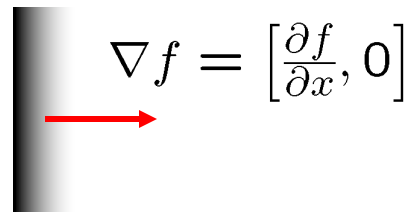
- Edges correspond to sharp changes of intensity
- Change is measured by 1<sup>st</sup> derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2<sup>nd</sup> derivative is zero.



# Image gradient

- The gradient of an image:  $\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$

- The gradient points in the direction of most rapid change in intensity



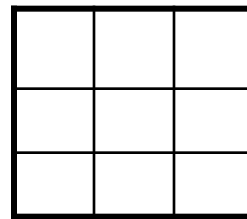
- The gradient direction is:  $\theta = \tan^{-1} \left( \frac{\partial f / \partial y}{\partial f / \partial x} \right)$
- The gradient magnitude is:  $\|\nabla f\| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2}$

# The discrete gradient

- How can we differentiate a *digital* image  $f[x,y]$ ?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx f[x + 1, y] - f[x, y]$$

- How to implement this as a spatial filter?



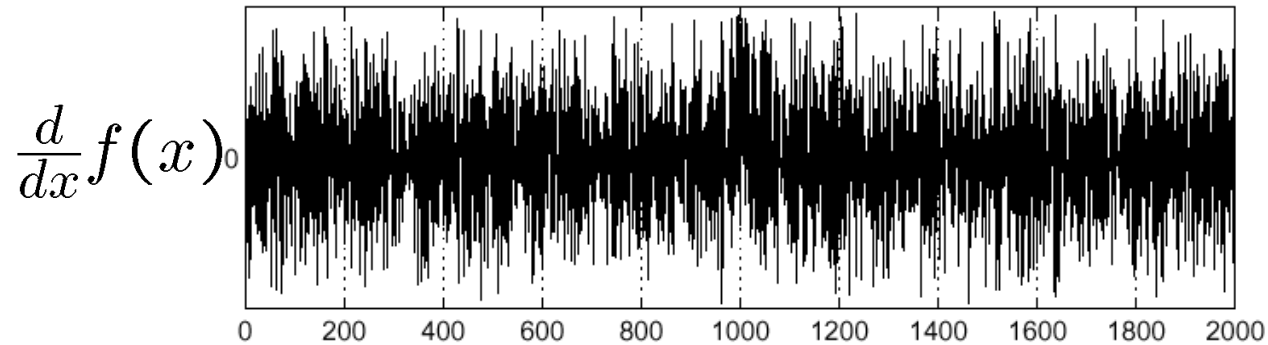
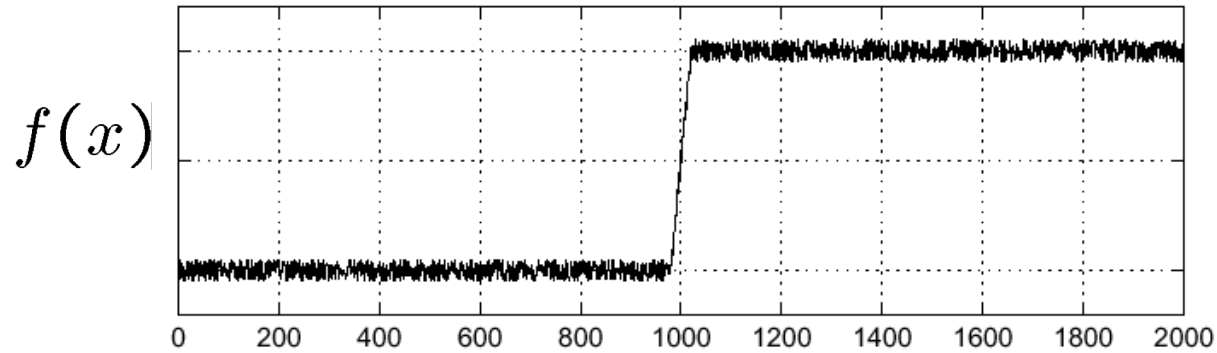
$W$

# Gradient Edge Detectors

- **Roberts**  $|G| \cong \sqrt{r_1^2 + r_2^2}$  ;  $r_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$  ;  $r_2 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$
- **Prewitt**  $|G| \cong \sqrt{p_1^2 + p_2^2}$  ;  $\theta \cong \text{atan}\left(\frac{p_1}{p_2}\right)$  ;  $p_1 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$  ;  $p_2 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$
- **Sobel**  $|G| \cong \sqrt{s_1^2 + s_2^2}$  ;  $\theta \cong \text{atan}\left(\frac{s_1}{s_2}\right)$  ;  $s_1 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$  ;  $s_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

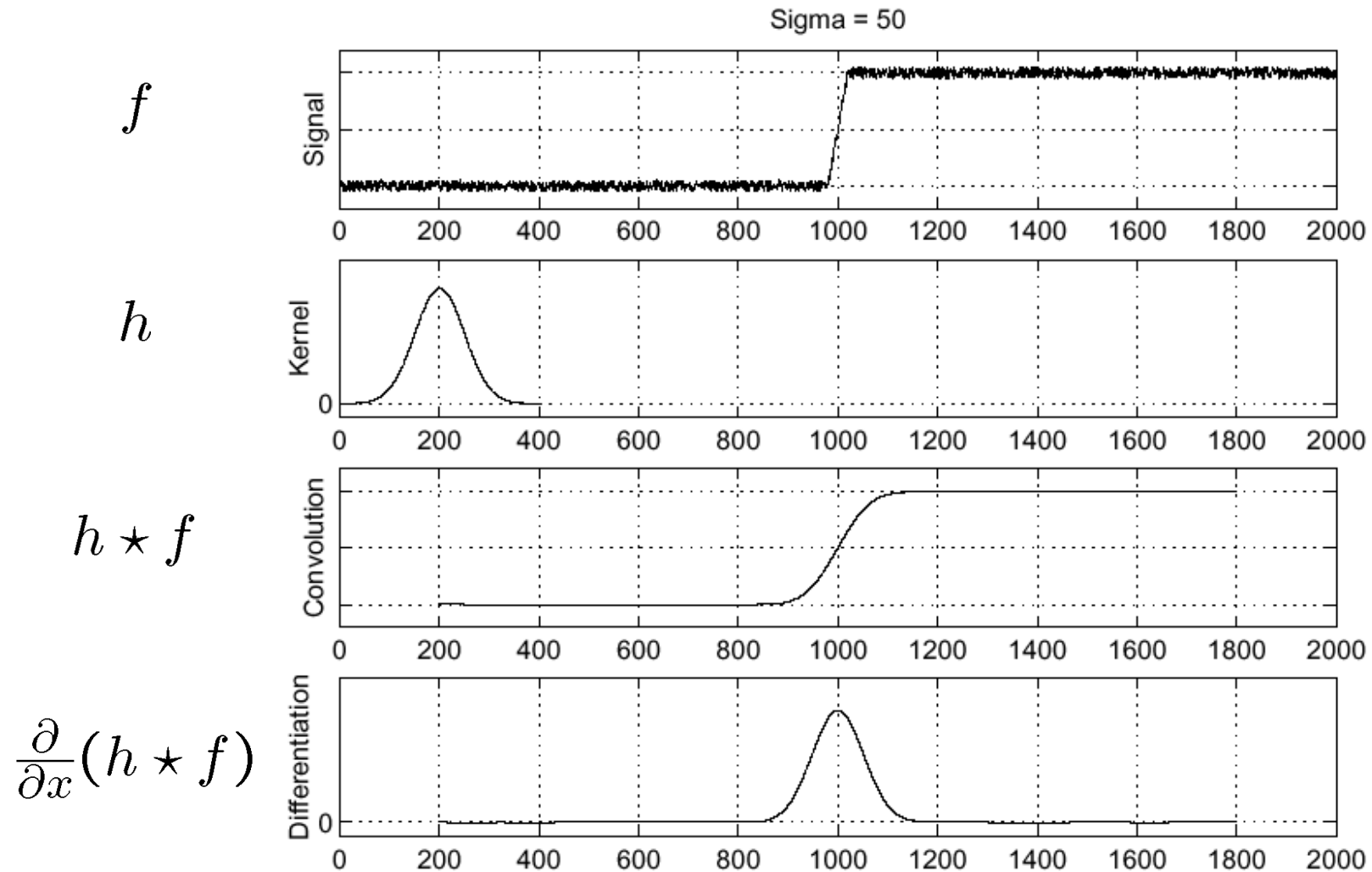
# Effects of noise

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



- Where is the edge?

# Solution: smooth first



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

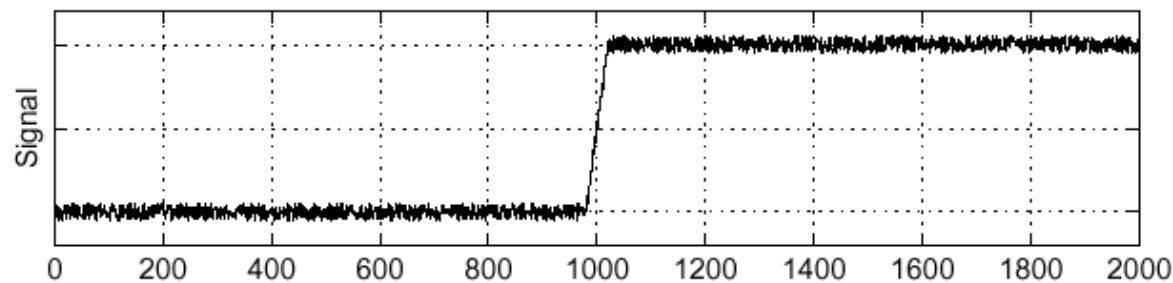
# Derivative theorem of convolution

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

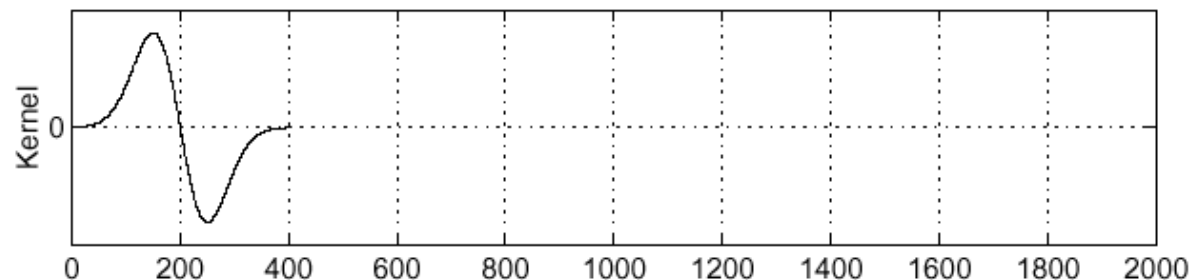
- This saves us one operation:

Sigma = 50

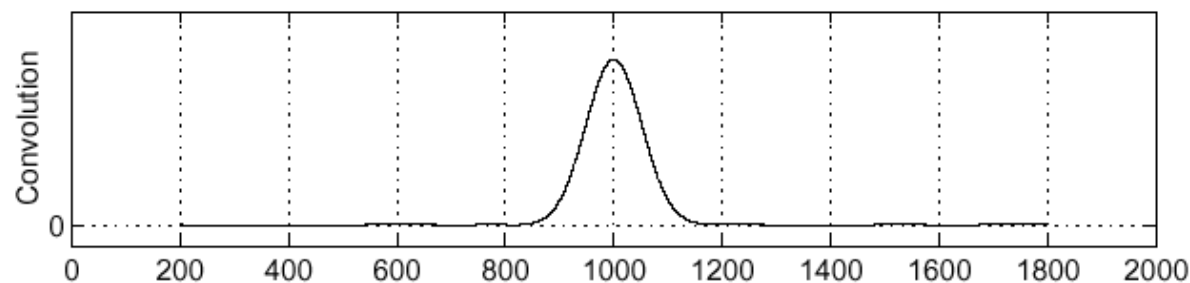
$f$



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



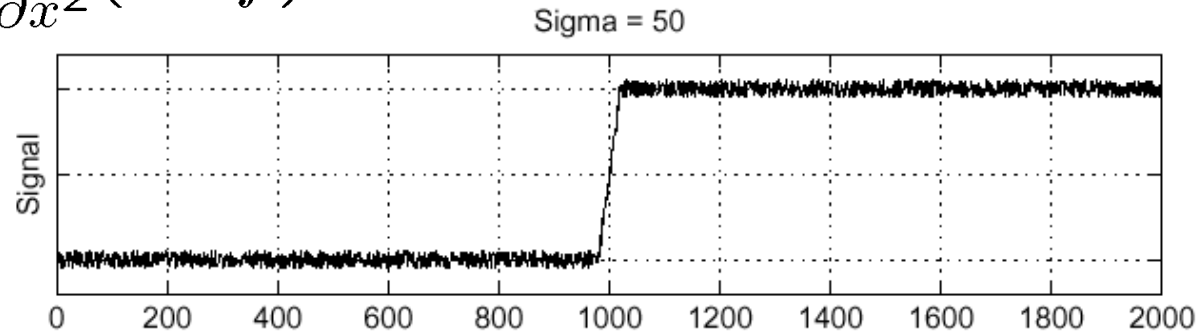
$\left(\frac{\partial}{\partial x}h\right) \star f$



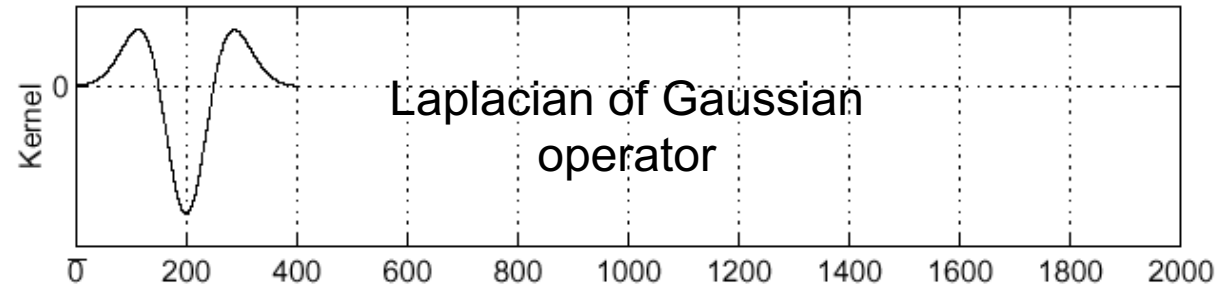
# The Canny Edge Detector

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

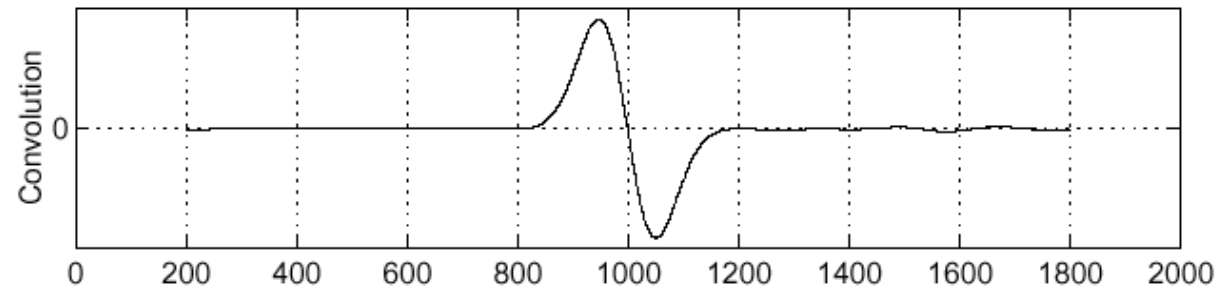
$f$



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

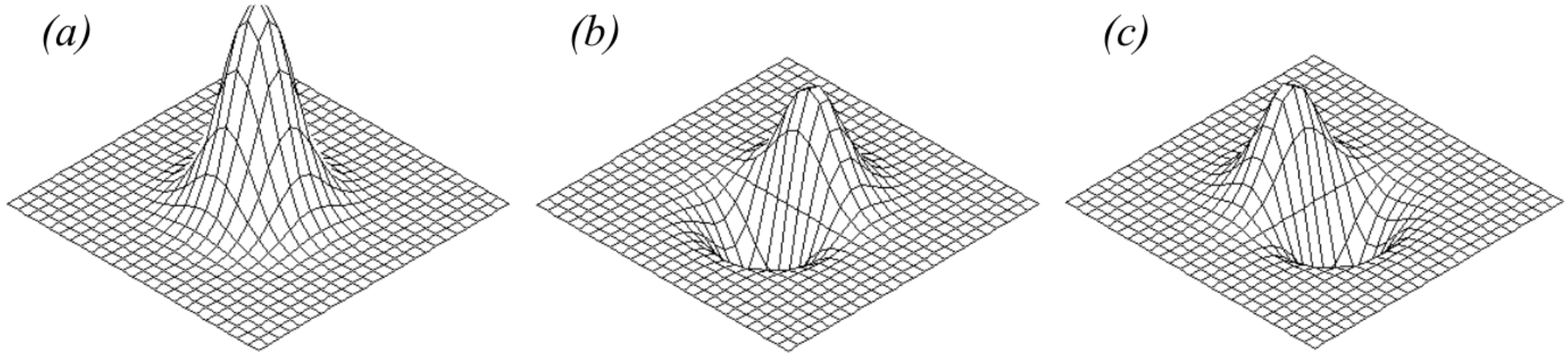


$(\frac{\partial^2}{\partial x^2}h) \star f$



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

# 2D Canny edge detector



$$G_\sigma(x, y) = G_\sigma(x)G_\sigma(y) \quad f_V(x, y) = G'_\sigma(x)G_\sigma(y) \quad f_H(x, y) = G'_\sigma(y)G_\sigma(x)$$

- Two perpendicular filters:
  - Convolve image  $I(x, y)$  with  $f_V(x, y)$  and  $f_H(x, y)$  – obtaining  $R_V(x, y)$  and  $R_H(x, y)$
  - Use square of gradient magnitude:  $R(x, y) = R_V^2(x, y) + R_H^2(x, y)$
  - Mark peaks in  $R(x, y)$  above a threshold

# The Sobel edge detector



original image (Lena image)

# The Sobel edge detector



norm of the gradient

# The Sobel edge detector



thresholding

# The Sobel edge detector



thinning  
(non-maxima suppression)

# IMAGE FEATURES

---

- Lines
- Points
  - Harris
  - SIFT

# Example: Build a Panorama



This panorama was generated using **AUTOSTITCH** (freeware), available at <http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html>

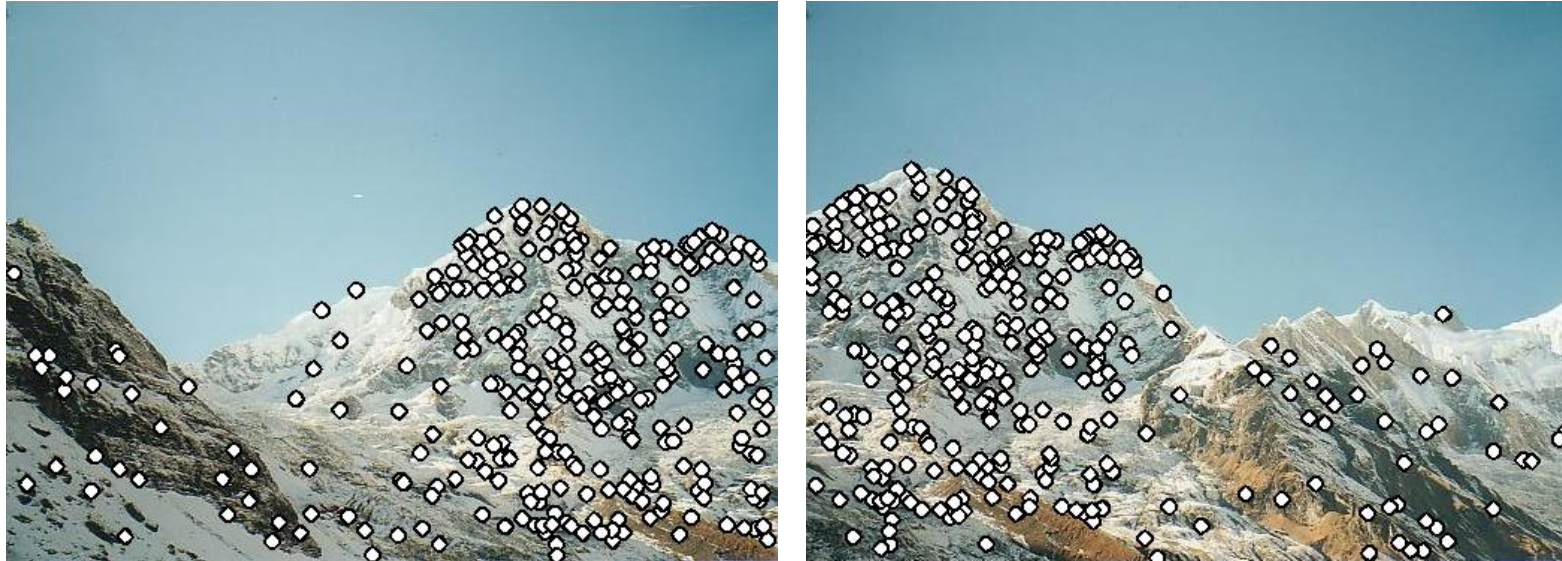
# How do we build panorama?

- We need to match (align) images



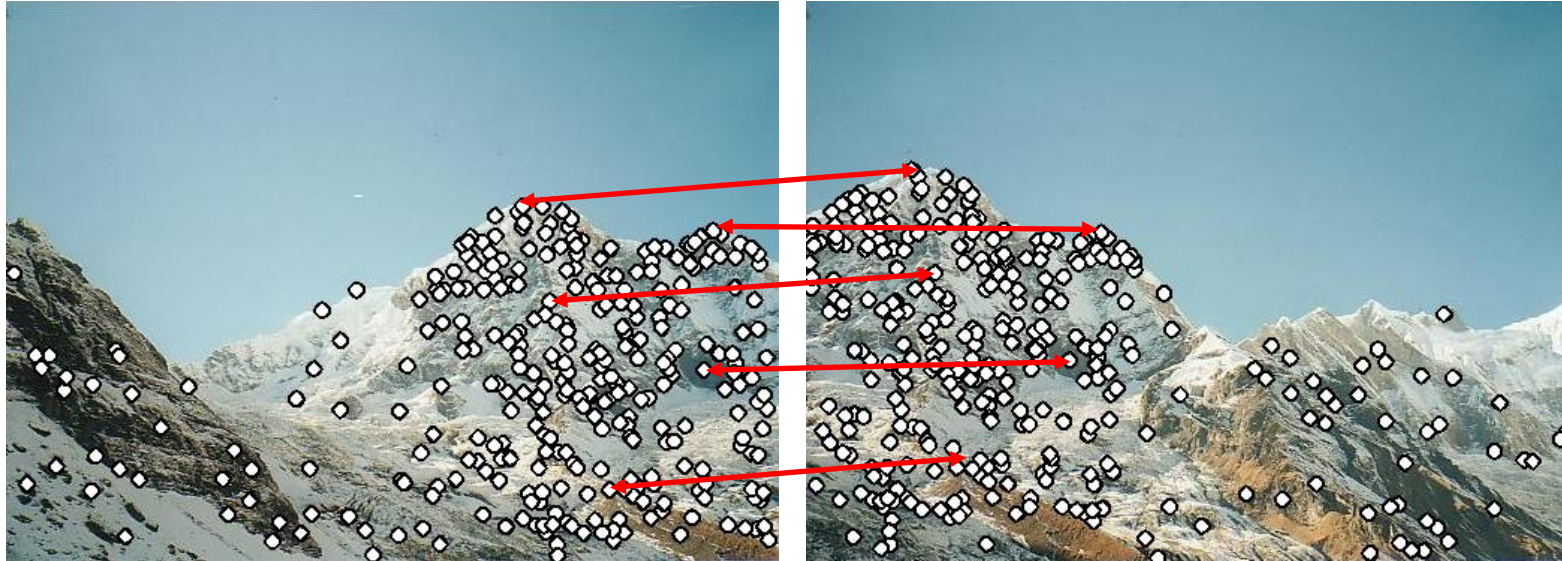
# Matching with Features

- Detect feature points in both images



# Matching with Features

- Detect feature points in both images
- Find corresponding pairs
  - Use RANSAC to find inliers and outliers



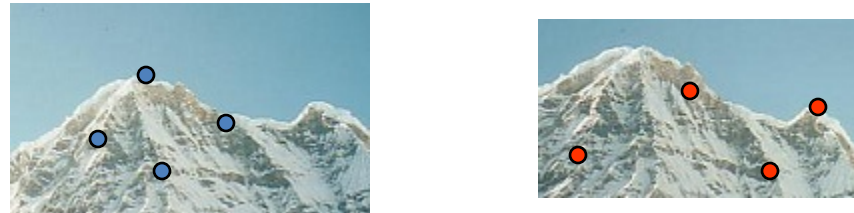
# Matching with Features

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



# Matching with Features

- Problem 1:
  - Detect the *same* point *independently* in both images

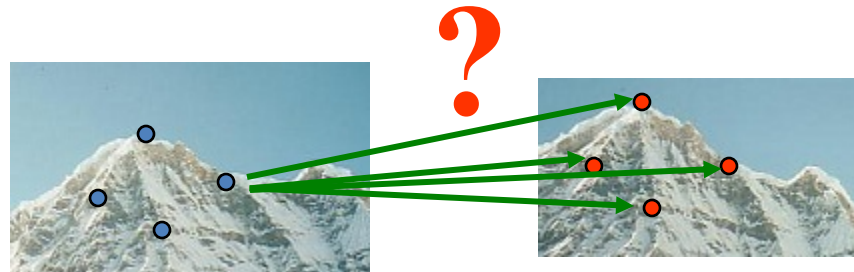


no chance to match!

We need a repeatable detector

# Matching with Features

- Problem 2:
  - For each point correctly recognize the corresponding one



We need a reliable and distinctive descriptor

# More motivation...

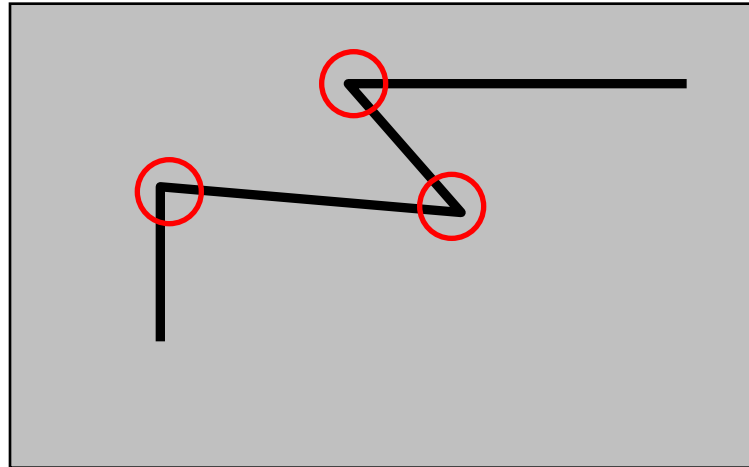
- Feature points are used also for:
  - Robot navigation
  - Object recognition
  - Image alignment (panoramas)
  - 3D reconstruction
  - Motion tracking
  - Indexing and database retrieval -> Google Images
  - ... other

# Most Famous Feature Detectors

- Overview
  - Harris Detector (1988)
  - SIFT Detector (2004)

# HARRIS CORNER DETECTOR

---



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

---

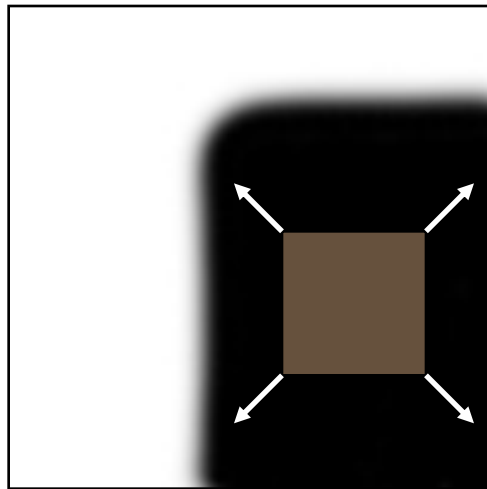
# Finding Corners



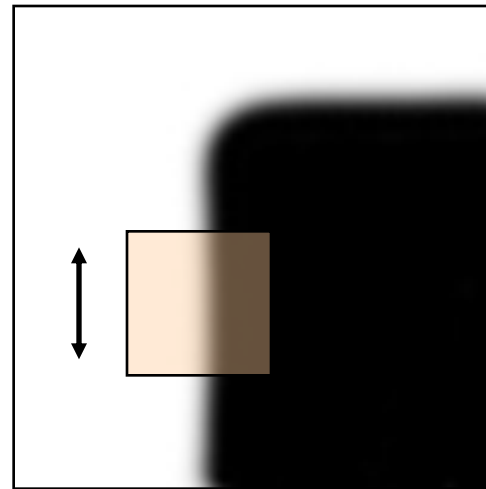
- Key property: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and distinctive

# The basic idea

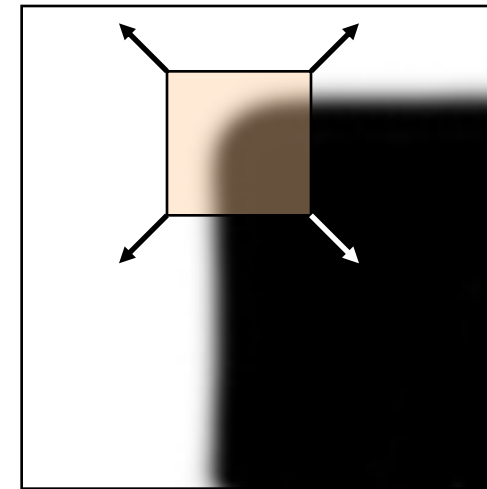
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give a *large change* in intensity
- => define a corner response function



“flat” region:  
no change in all  
directions



“edge”:  
no change along the  
edge direction



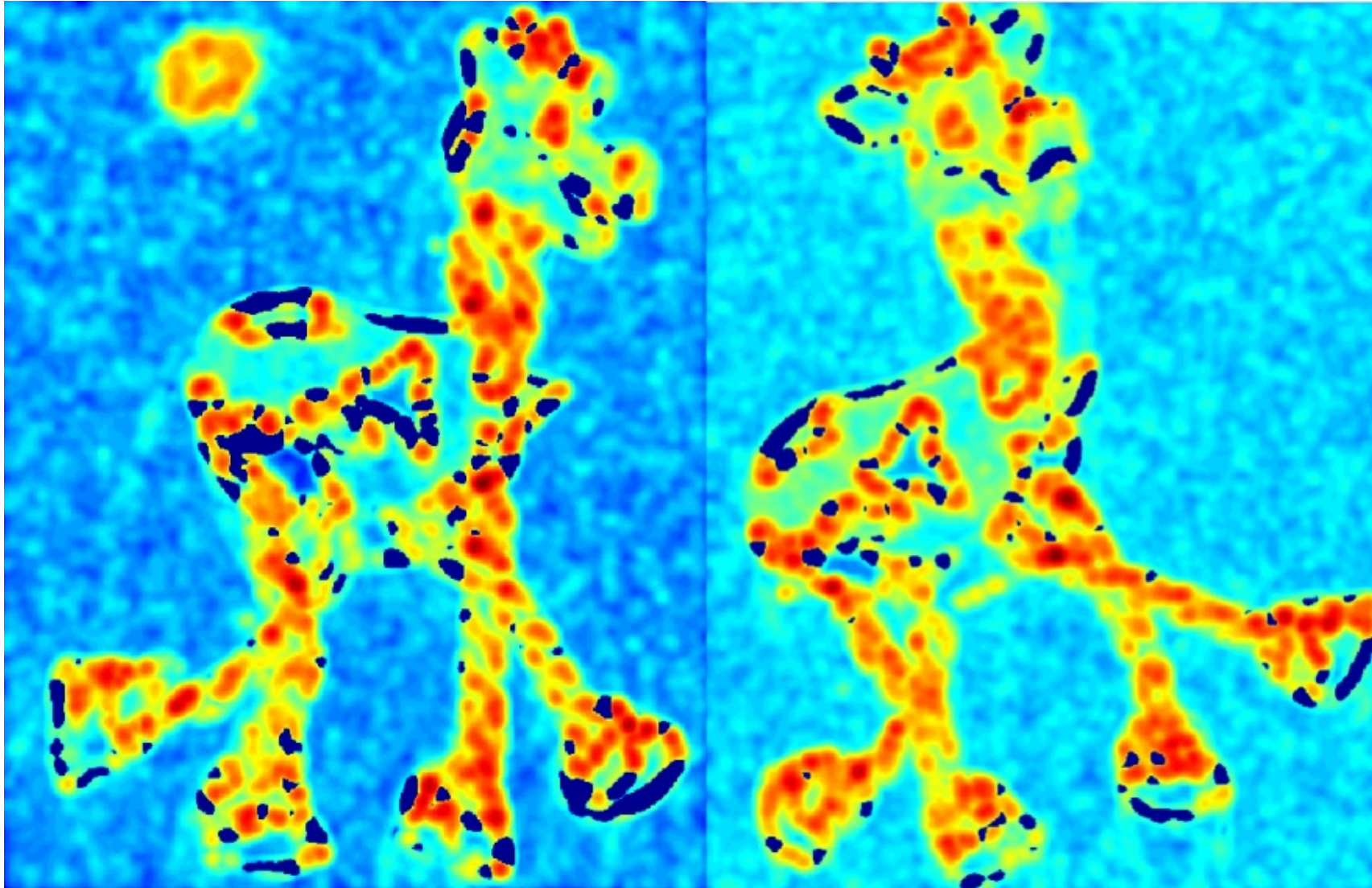
“corner”:  
significant change in  
all directions

# Harris Detector: Workflow



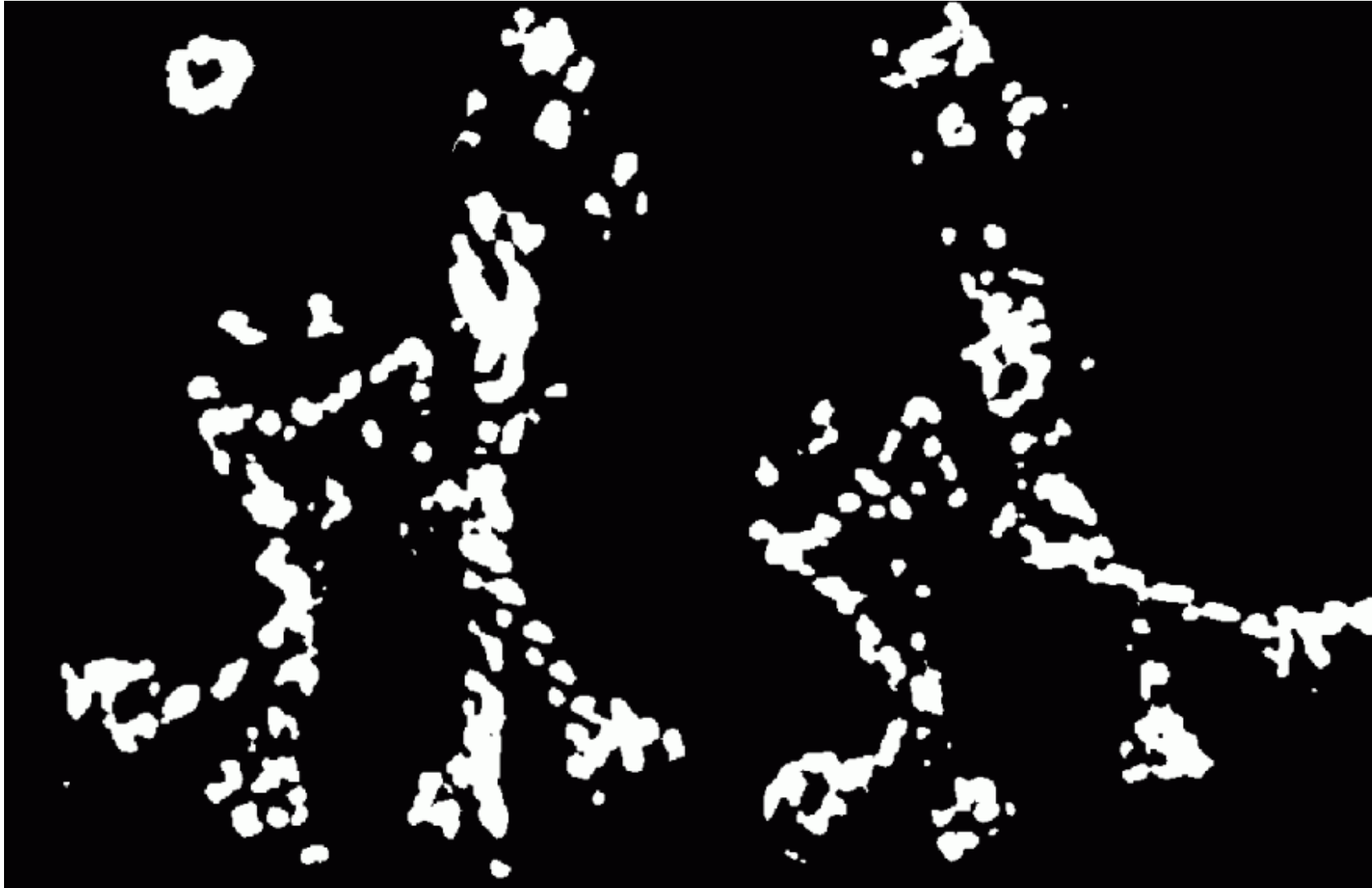
# Harris Detector: Workflow

- Compute corner response  $R$



# Harris Detector: Workflow

- Find points with large corner response:  $R > \text{threshold}$



# Harris Detector: Workflow

- Take only the points of local maxima of  $R$



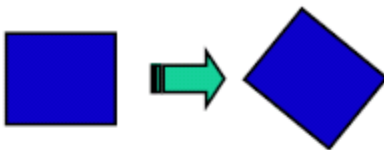
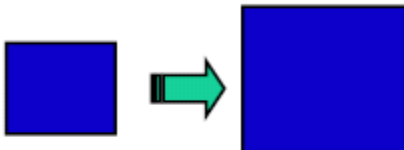


# Harris Detector: Workflow



# Harris detector: properties

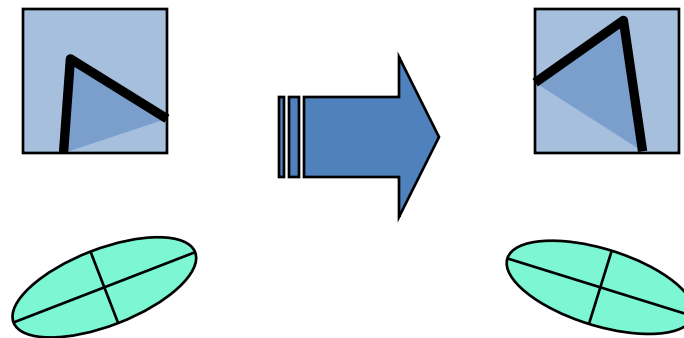
- How does the Harris detector behaves to image transformations?
- Will we be able to re-detect the same corners under
  - Rotations
  - View-point changes
  - Zoom changes
  - Illumination changes?
- In order to answer these questions, we need a model of these transformations.
- The detector can then be modified to be invariant to such transformations

# Models of Image Change

- Geometric
  - Rotation 
  - Scale 
  - Affine   
valid for locally planar object
- Photometric
  - Affine intensity change ( $I \rightarrow a I + b$ ) 

# Harris Detector: Some Properties

- Rotation invariance

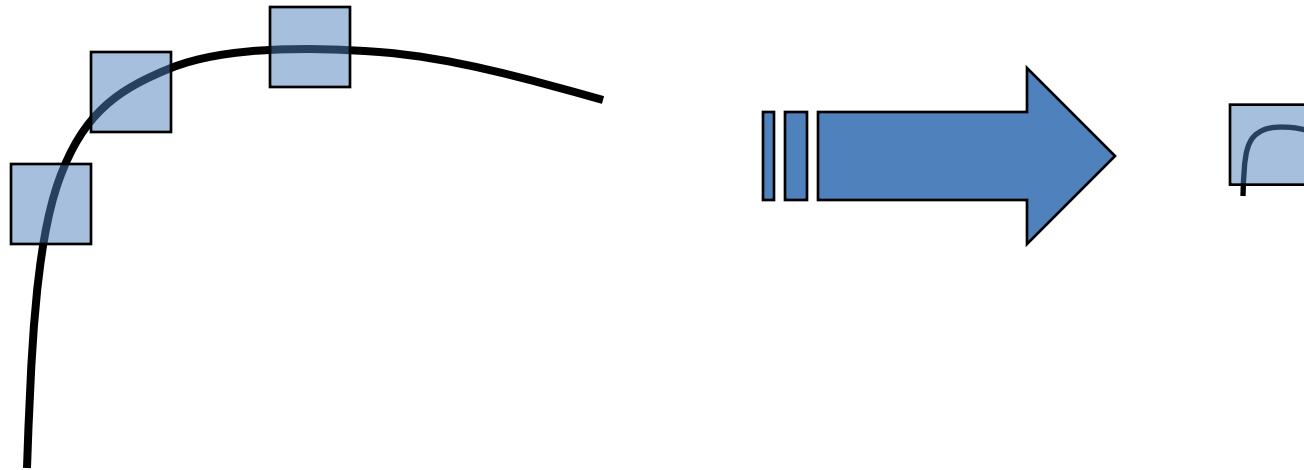


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

*Corner response  $R$  is invariant to image rotation*

# Harris Detector: Some Properties

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



All points will be  
classified as **edges**

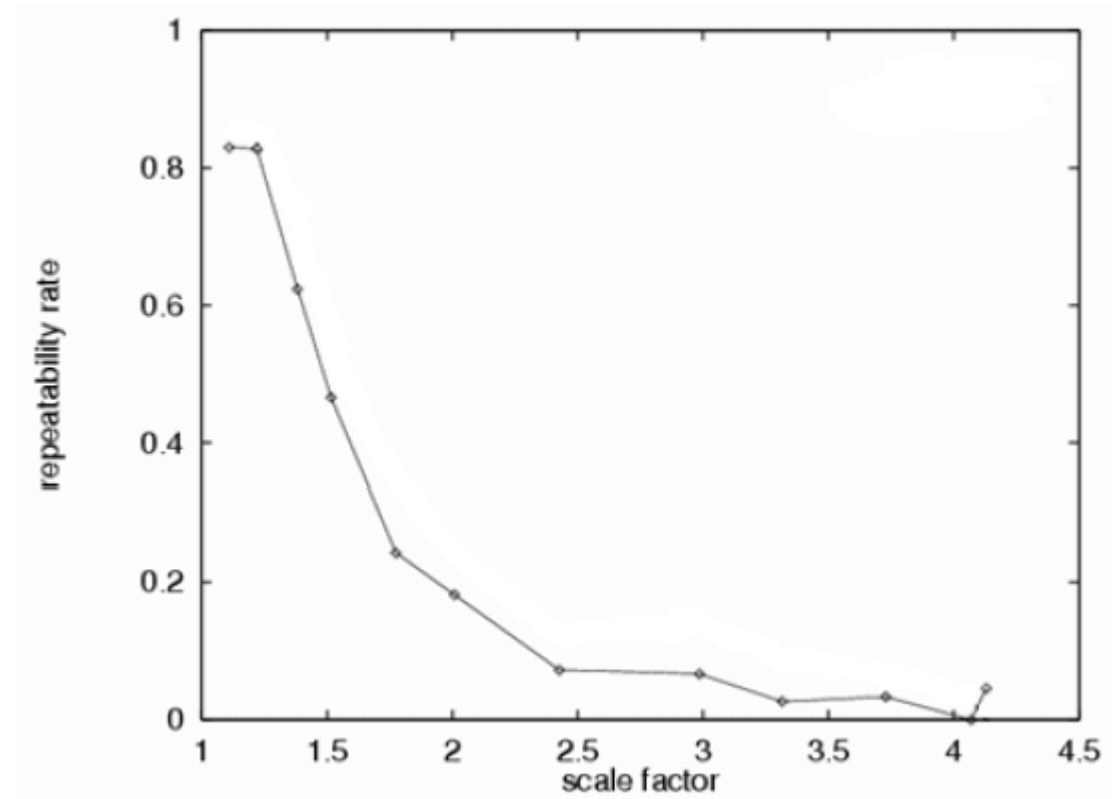
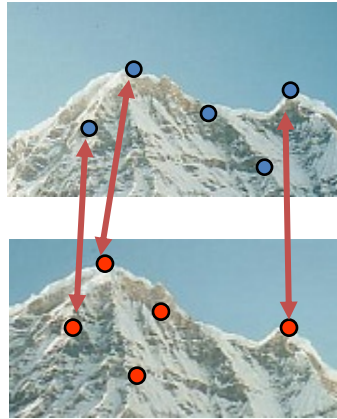
**Corner !**

# Harris Detector: Some Properties

- Quality of Harris detector for different scale changes

Repeatability rate:

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

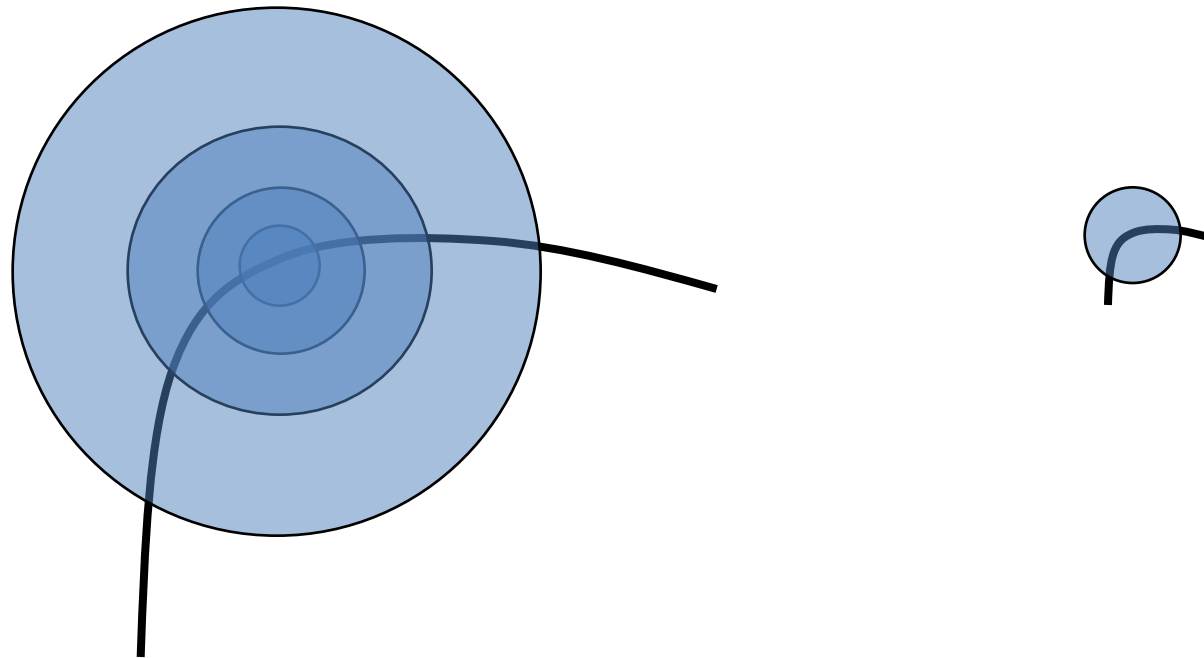


# Summary on Harris properties

- 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
  - Linear intensity changes
  - However, to make the matching invariant to these we need an appropriate matching criterion (for example, SSD is not Rotation nor affine invariant!)
- The detection is NOT invariant to
  - Scale changes
  - Geometric affine changes (Intuitively, an affine transformation distorts the neighborhood of the feature along the x and y directions and, accordingly, a corner can get reduced or increased its curvature)

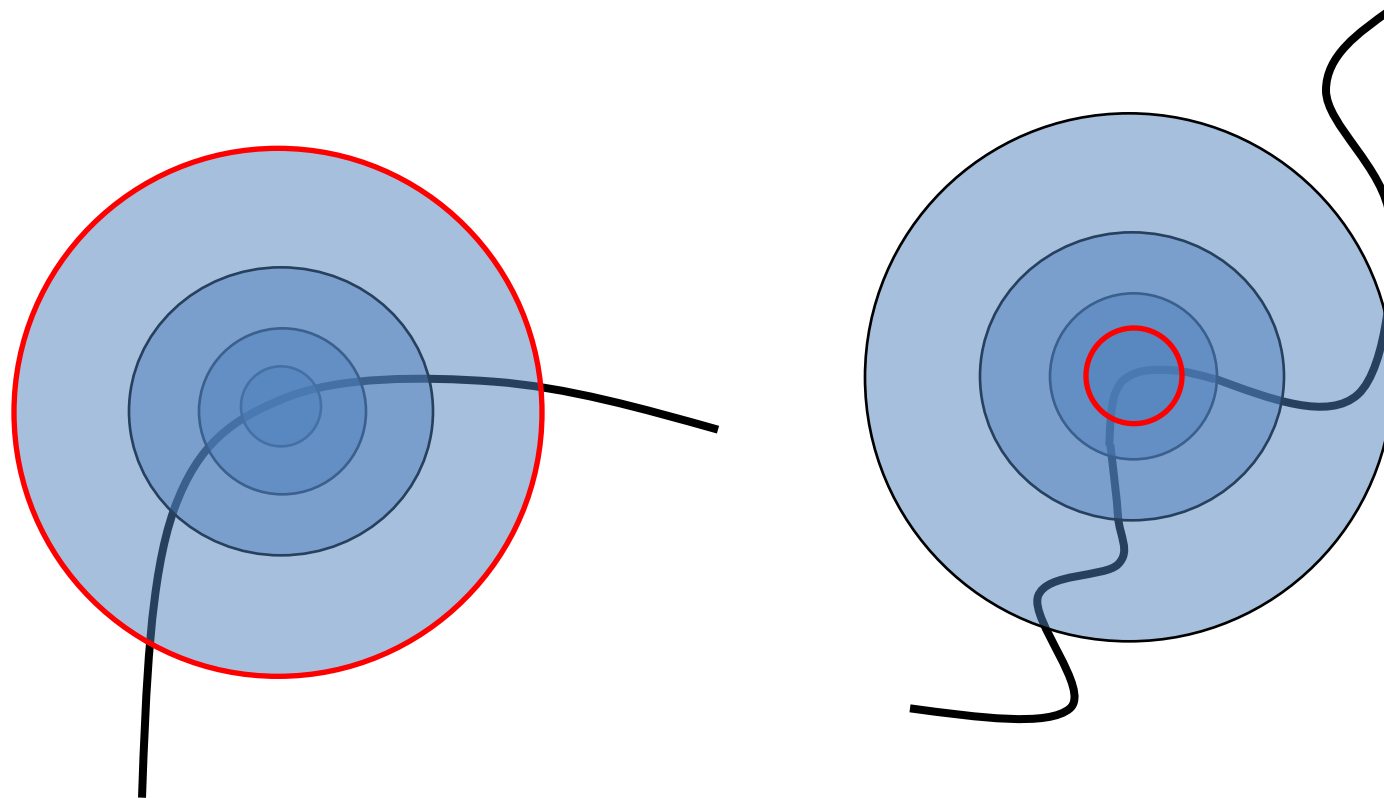
# Scale Invariant Detection

- 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?

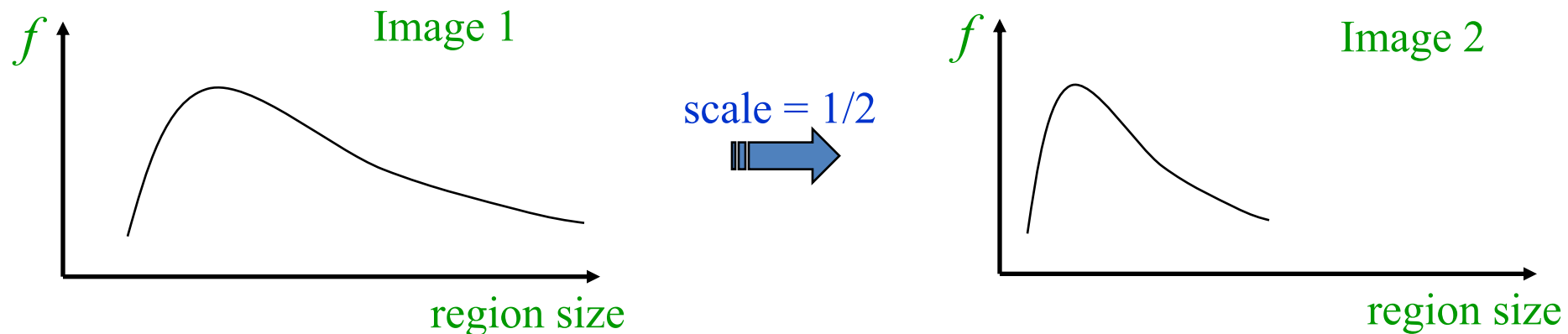


# Scale Invariant Detection

- Solution:
  - 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)



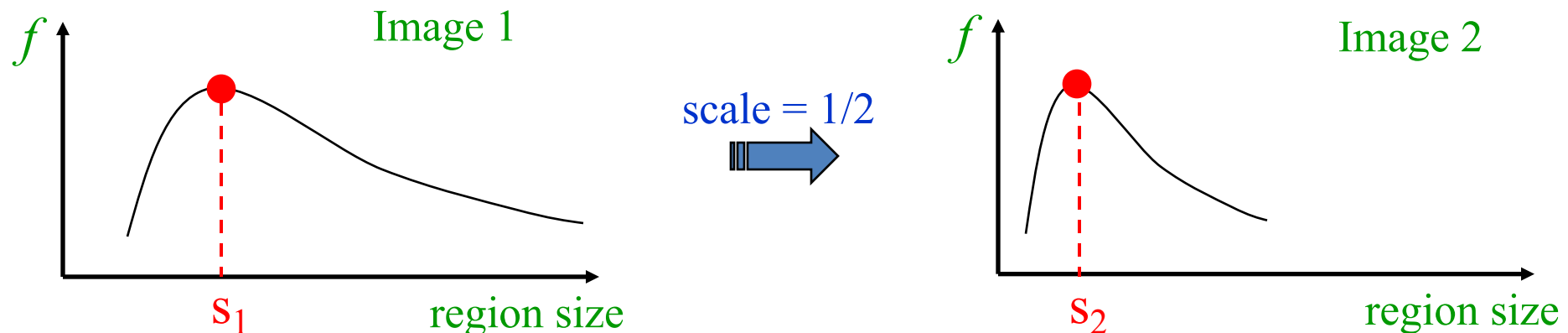
# Scale Invariant Detection

- Common approach:

Take a local maximum of this function

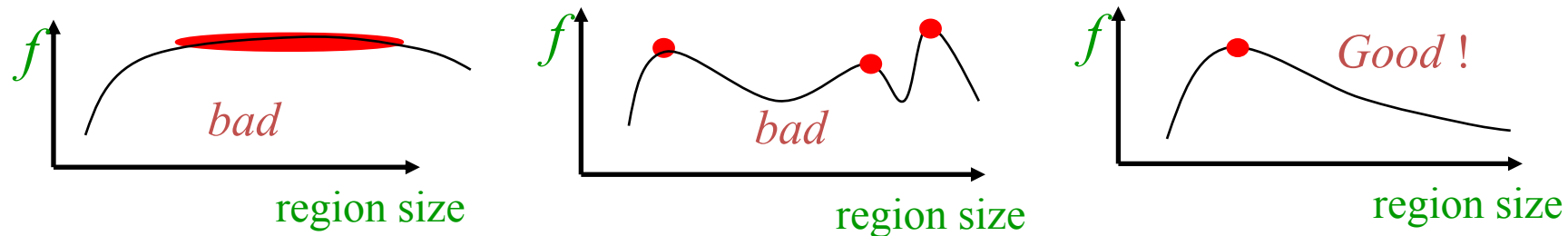
Observation: region size, for which the maximum is achieved, should be *invariant* to image scale.

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



# Scale Invariant Detection

- 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

- Functions for determining scale

Kernels:

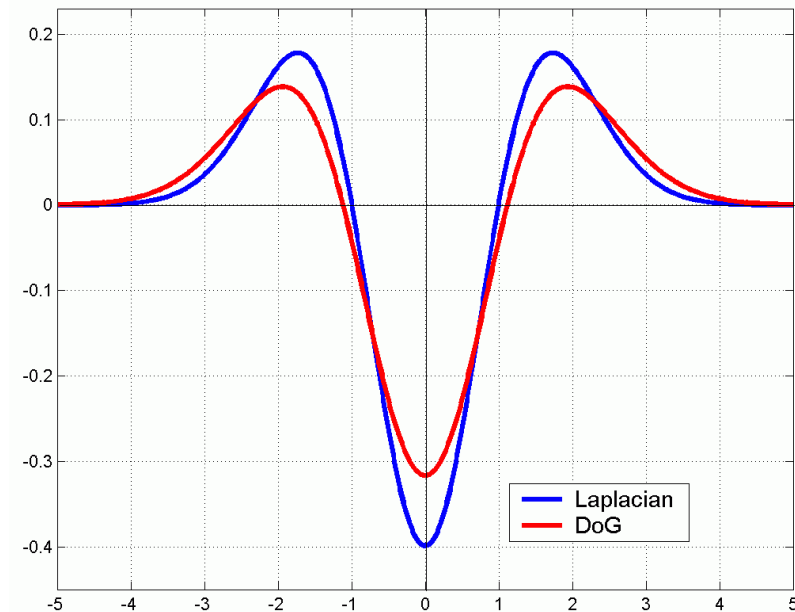
$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

where Gaussian

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

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



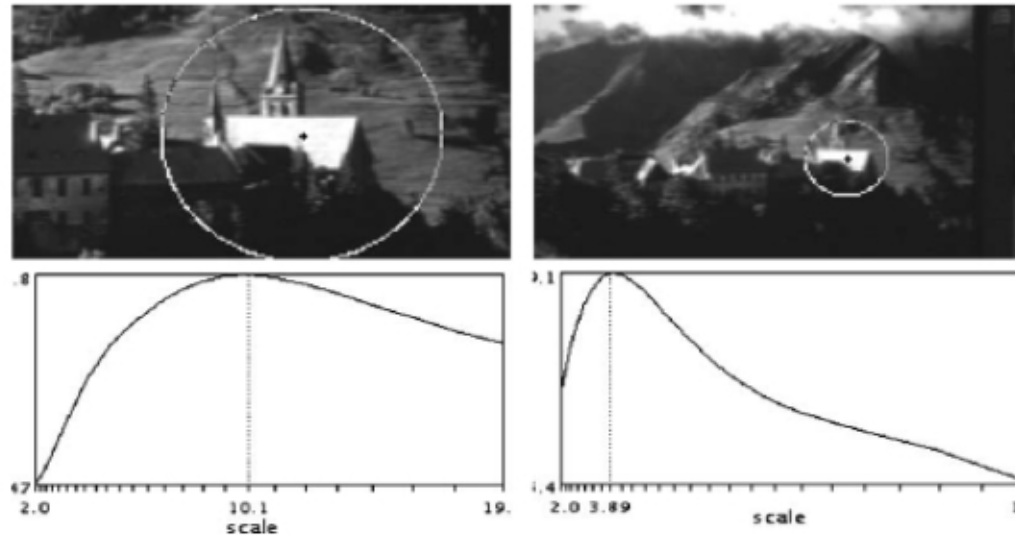
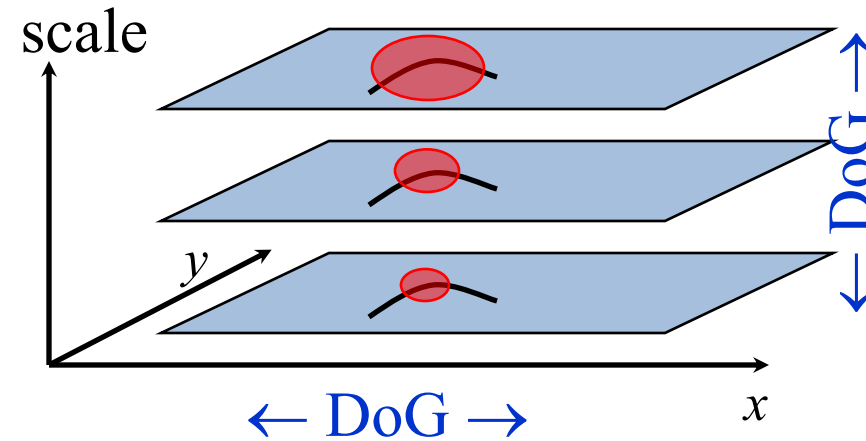
Note: This kernel is invariant to  
*scale* and *rotation*

# Scale Invariant Detectors

- **SIFT**

*Find local maximum of:*

- Difference of Gaussians in space and scale

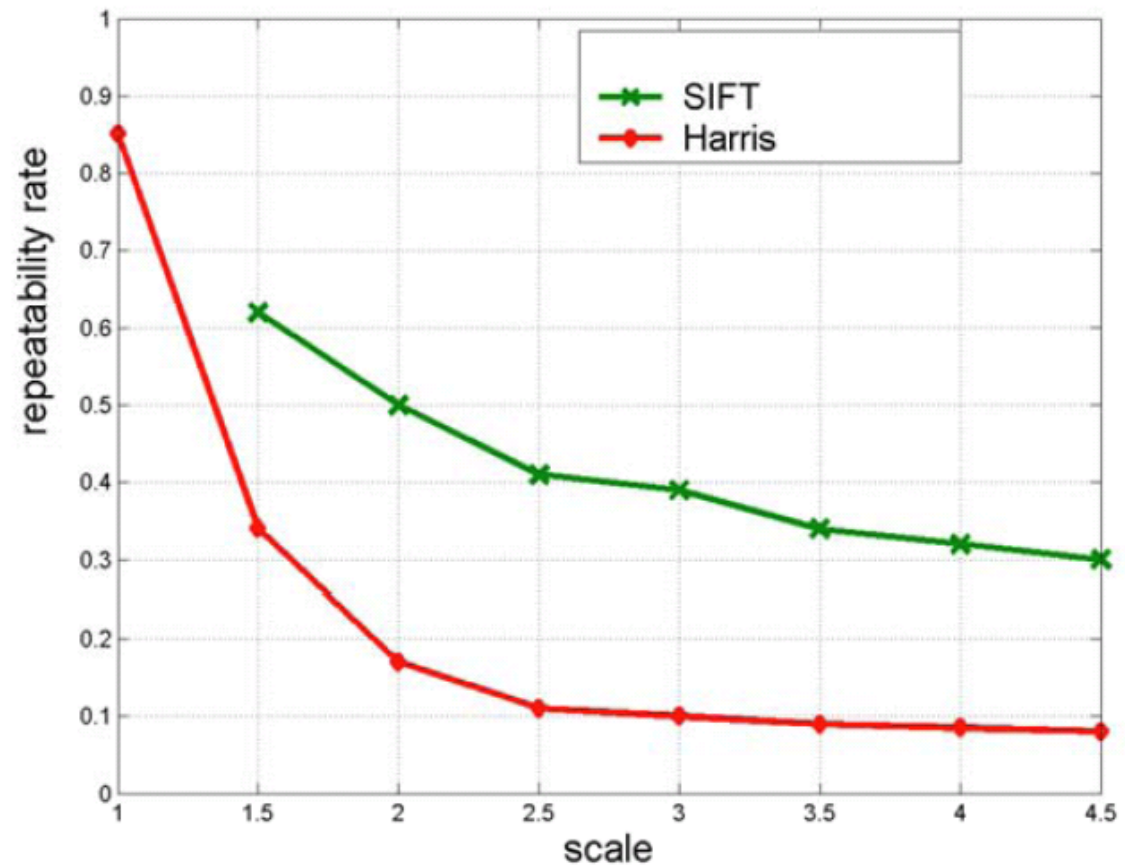
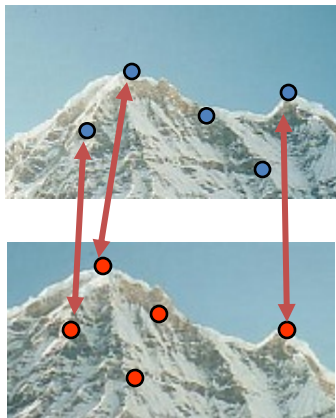


# Scale Invariant Detectors

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

Repeatability rate:

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

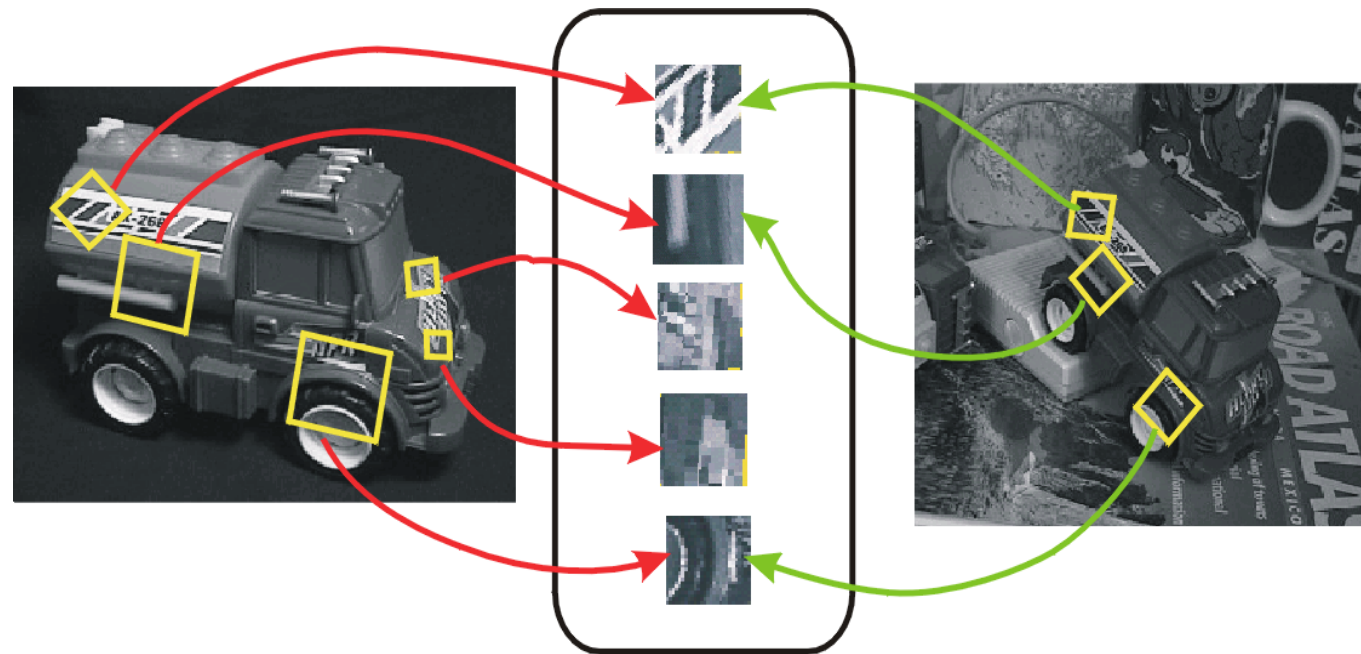


# What is SIFT ?

- 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

# Invariant Local Features

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



# SIFT Descriptor

- SIFT matching features between different images invariant to:
  - Rotation
  - Small image scale
  - Small view point changes
- This is made possible by the use of an opportune feature descriptor (remember, Harris does not use a descriptor for matching)
- What is a descriptor? A descriptor is a “description”, identity card, which allows to recognize a given feature uniquely among many others!

# Generation of Keypoint Descriptor

- Keypoint descriptor: computing gradient magnitude and orientation at each image sample point in a region around the keypoint location
- Samples are accumulated into orientation histograms with the length of each bin corresponding to the sum of the gradient magnitudes near that direction within the region.
- The descriptor is therefore a vector containing the values of all the orientations histogram entries
- This vector is normalized to enhance invariance to changes in illumination
- 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.

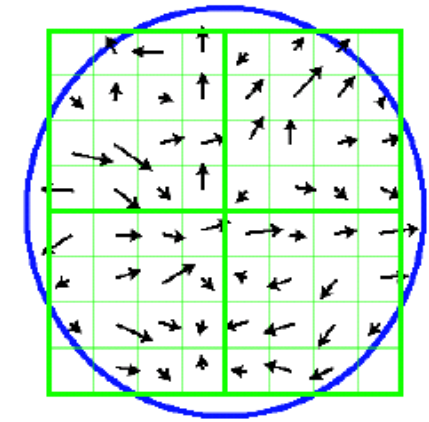
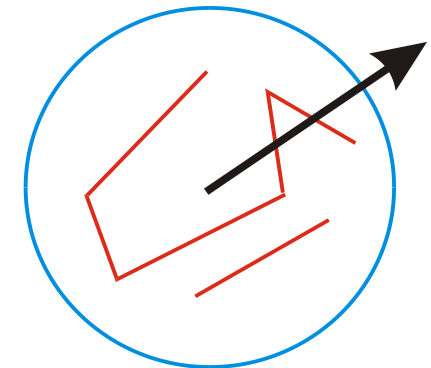
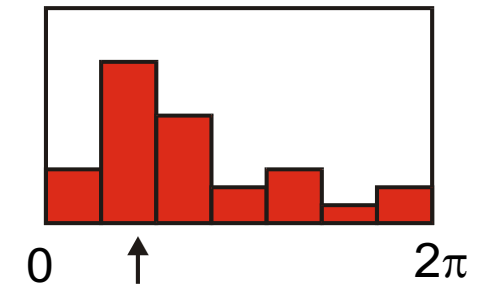
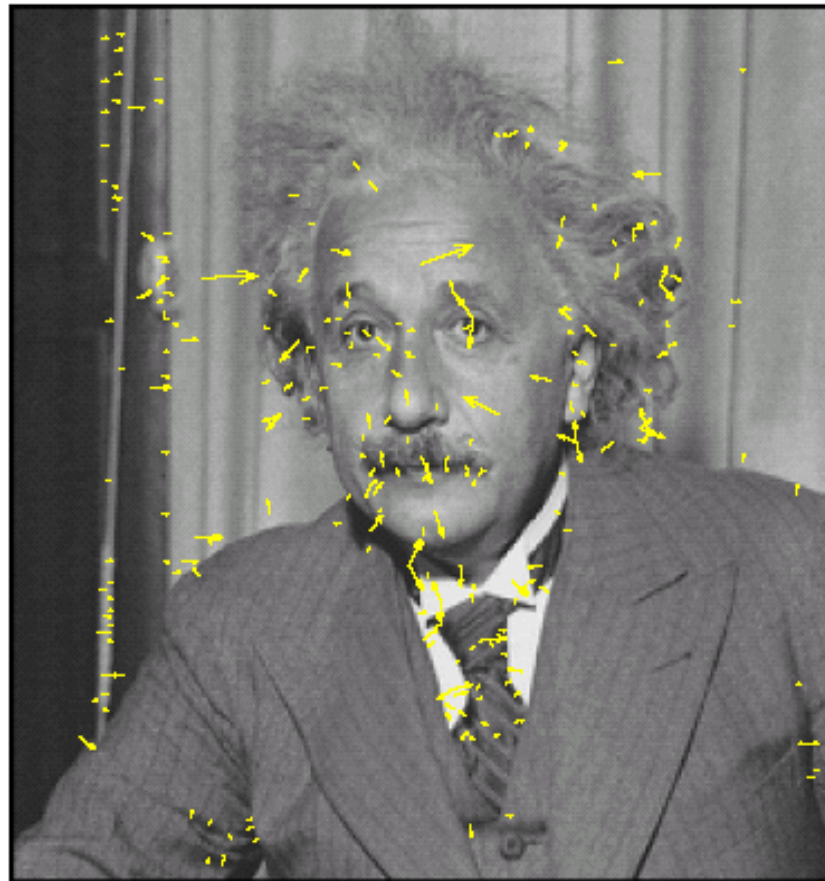


Image gradients

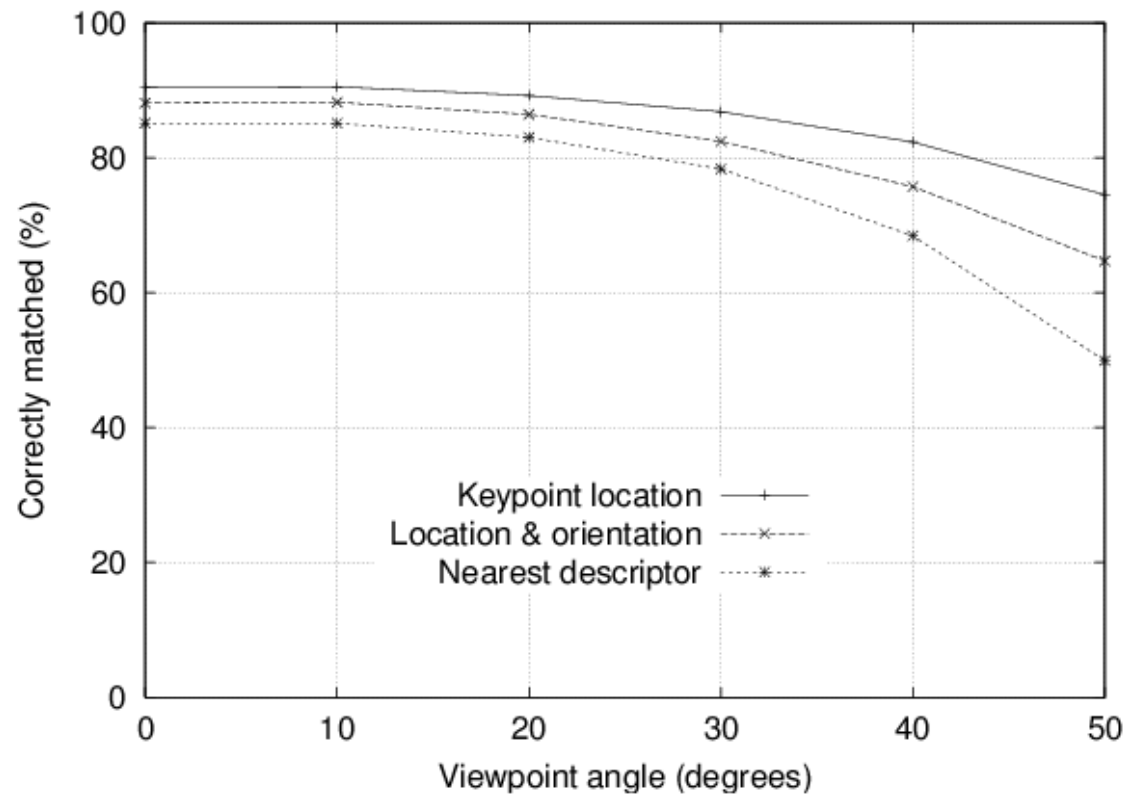


# Key point localization

- Final keypoints with selected orientation and scale



# Feature stability to view point change

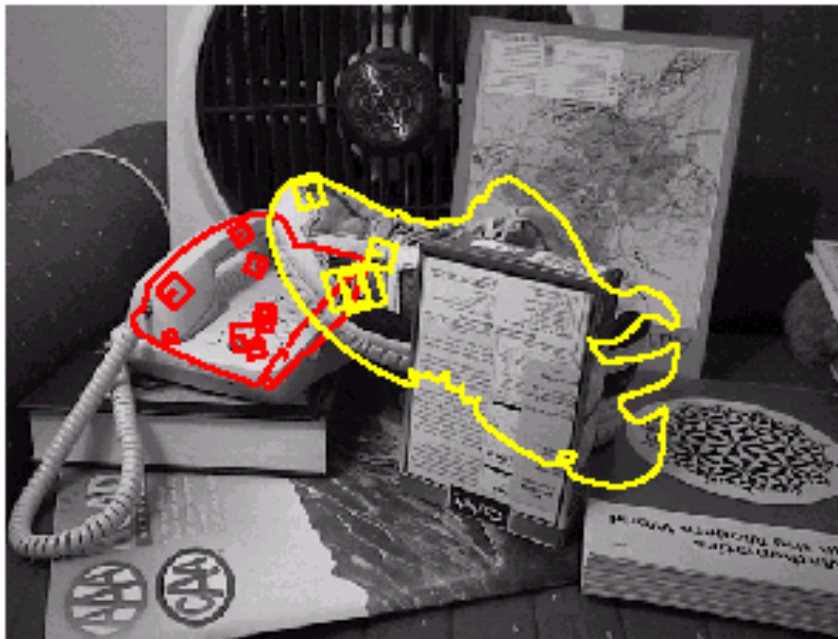


# Planar recognition

- Planar surfaces can be reliably recognized at a rotation of  $60^\circ$  away from the camera
- Only 3 points are needed for recognition
- But objects need to possess enough texture (i.e. no monochrome objects!)



# Recognition under occlusion

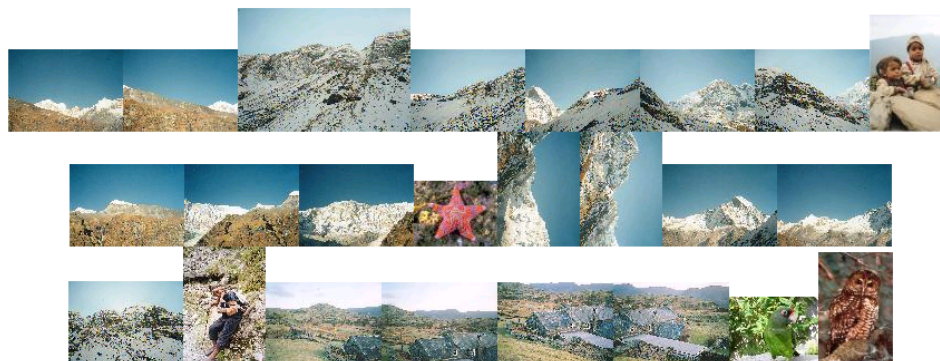


# Place recognition



# Multiple panoramas from an unordered image set

SIFT is used in current consumer cameras and smart phones to build panoramas from multiple shots!



*Input images*



*Output panorama 1*

