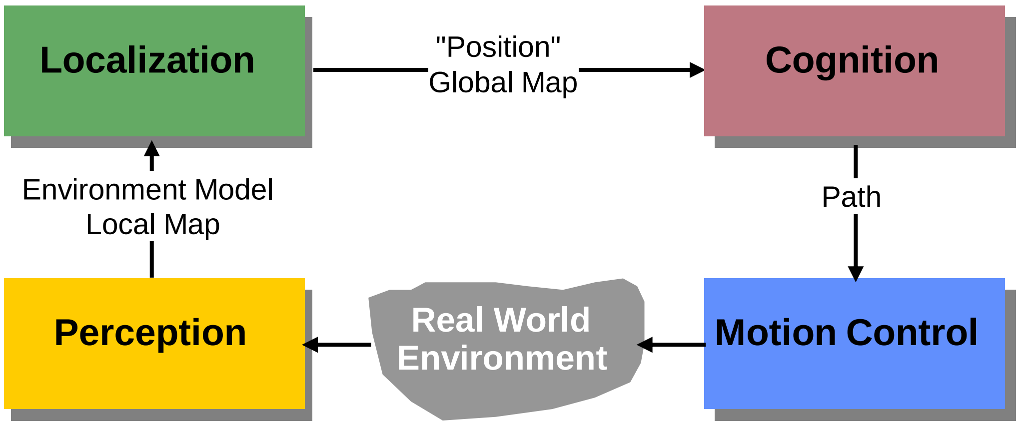


Perception

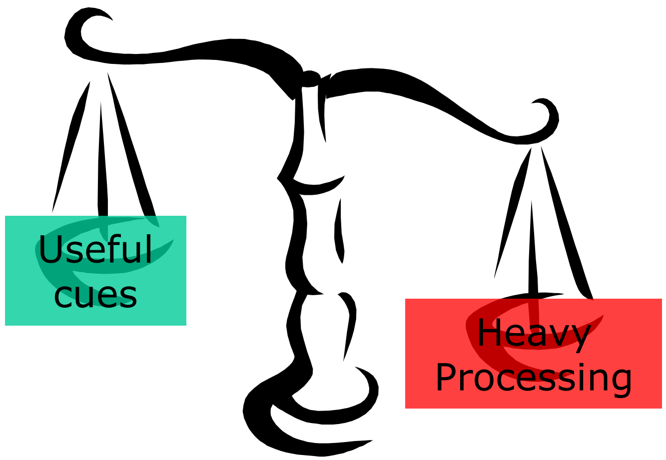
Sensors

Vision

Uncertainties, Line extraction from laser scans

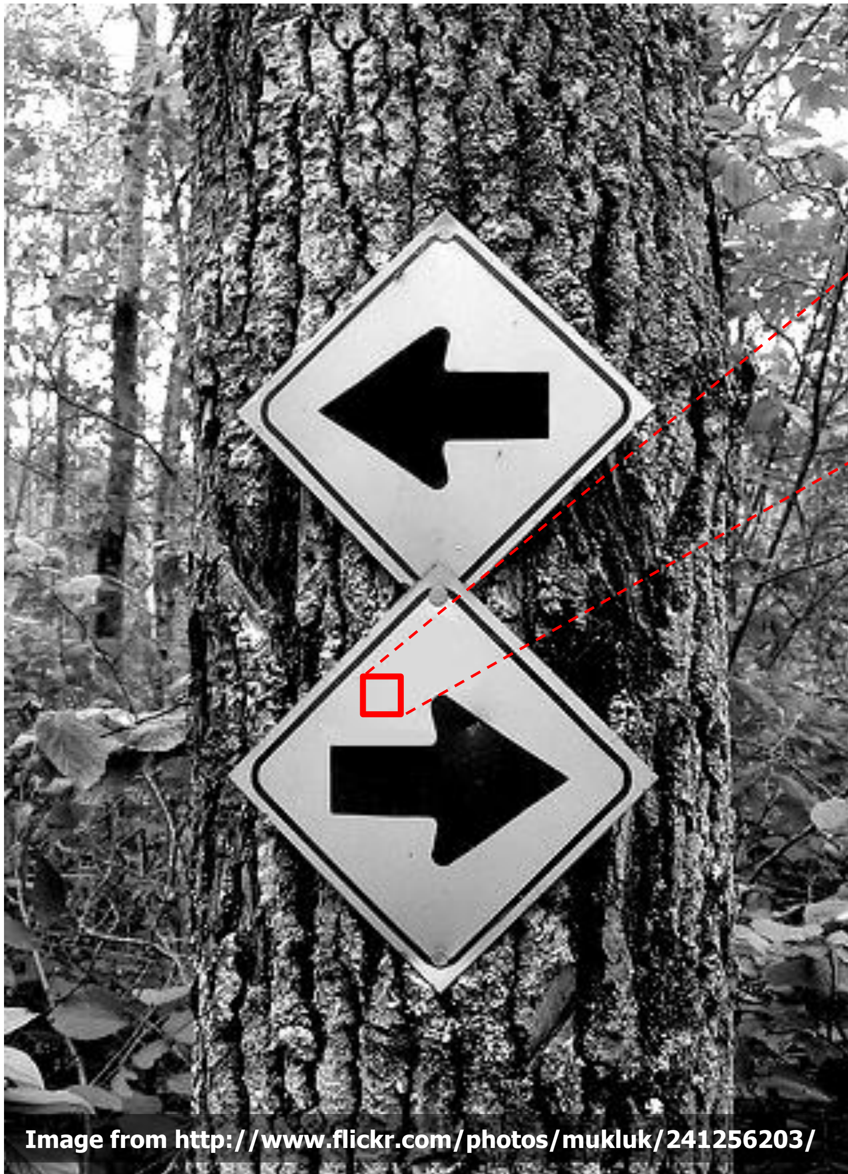
Image Intensities & Data reduction

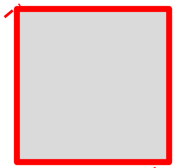
- Monochrome image ⇨ matrix of intensity values
- Typical sizes:
  - 320 x 240 (QVGA)
  - 640 x 480 (VGA)
  - 1280 x 720 (HD)
- Intensities sampled to 256 grey levels ⇨ 8 bits
- Images capture a lot of information



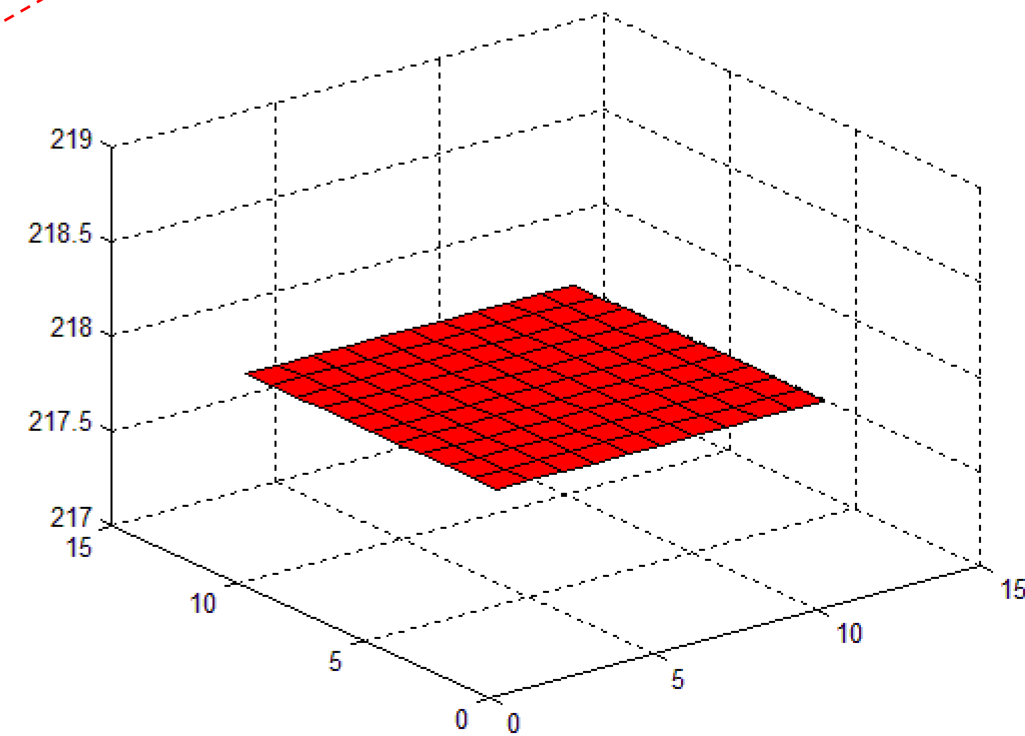
⇨ Reduce the amount of input data:
preserving useful info & discarding redundant info

What is **USEFUL**, What is **REDUNDANT** ?



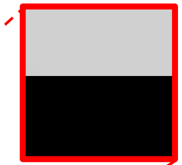


218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218

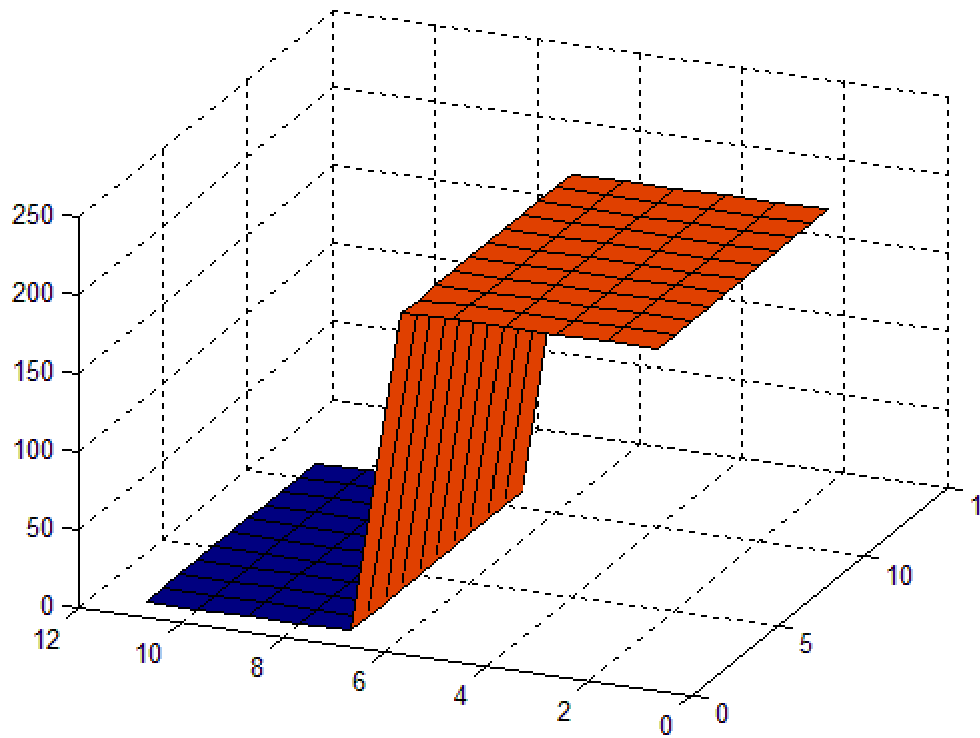


What is **USEFUL**, What is **REDUNDANT** ?

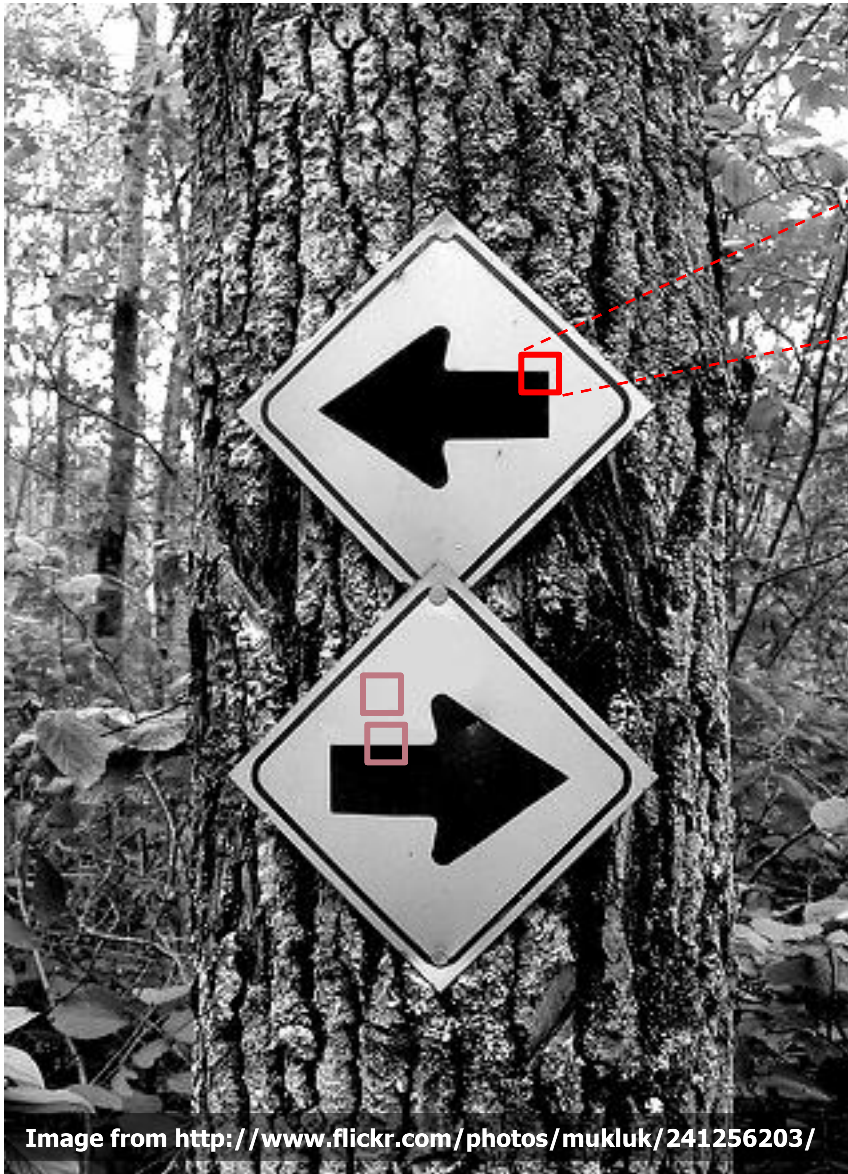
Image from http://www.flickr.com/photos/mukluk/241256203/

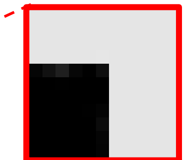


208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	208	208	208	208	208	208	208	208	208	208
208	207	208	208	208	208	208	208	208	208	208
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

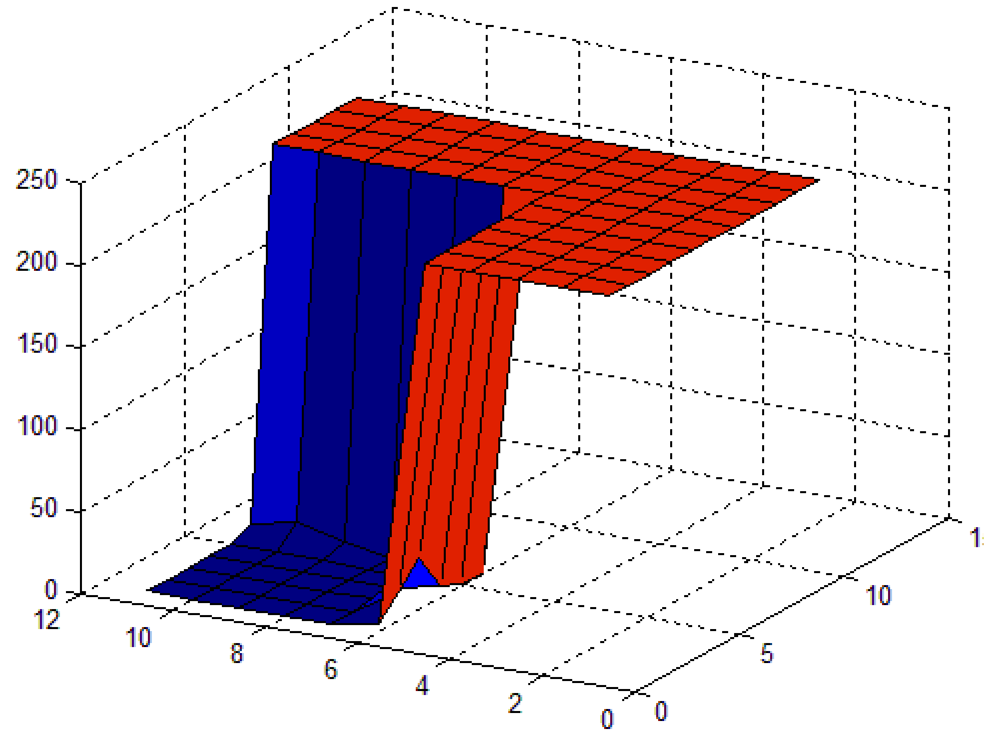


What is **USEFUL**, What is **REDUNDANT** ?





229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	229	229	229	229	229	229
229	229	229	229	229	230	229	229	229	229	229
5	17	31	7	1	0	229	229	229	229	229
0	0	1	0	0	0	229	229	229	229	229
0	0	0	0	0	0	229	229	229	229	229
0	0	0	0	1	4	229	229	229	229	229
0	0	0	0	0	11	229	229	229	229	229
0	0	0	0	0	5	229	229	229	229	229



Today's Topics

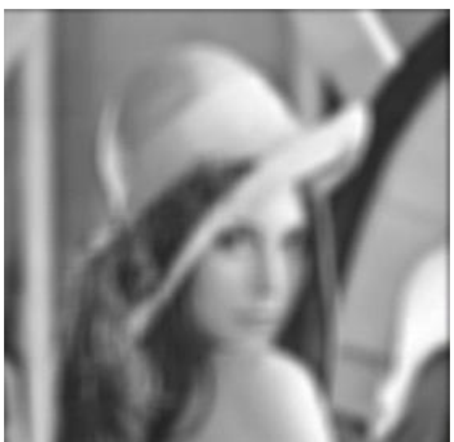
Sections **4.3 – 4.5** of the book

- Image Filtering
  - Correlation
  - Convolution
- Edge / Corner Detection
- Image Features
  - Harris corners
  - SIFT features

Image filtering

- “filtering”: accept / reject certain components
- Example: a lowpass filter allows low frequencies

  ⇨ blurring (smoothing) effect on an image – used to reduce image noise
- Smoothing can also be achieved by **spatial filters** instead of **frequency filters**.



Lowpass filtered image

Highpass filtered image

Spatial filters

- S_{xy} : neighborhood of pixels around the point (x,y) in an image I
- Spatial filtering operates on S_{xy} to generate a new value for the corresponding pixel at output image J

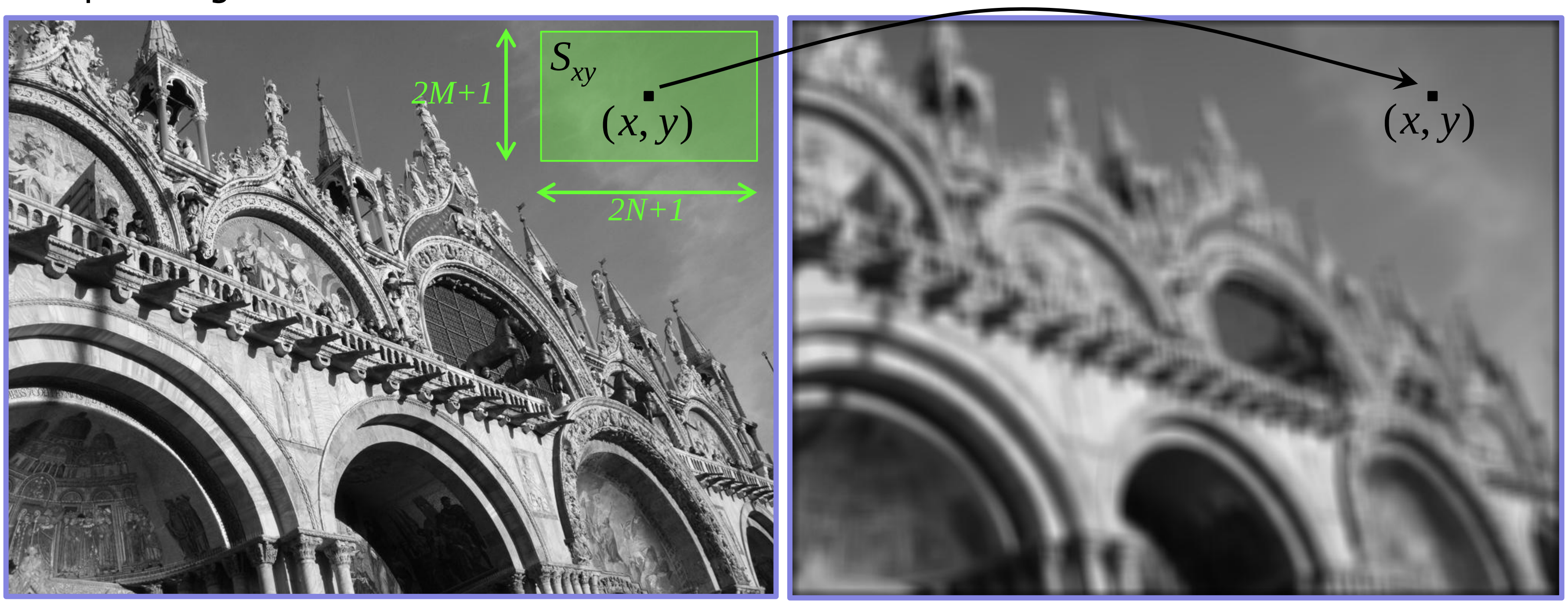


Image I Filtered Image $J = F(I)$

- For example, an averaging filter is: $J(x,y) = \dfrac{\sum_{(r,c)\in S_{xy}} I(r,c)}{(2M+1)(2N+1)}$

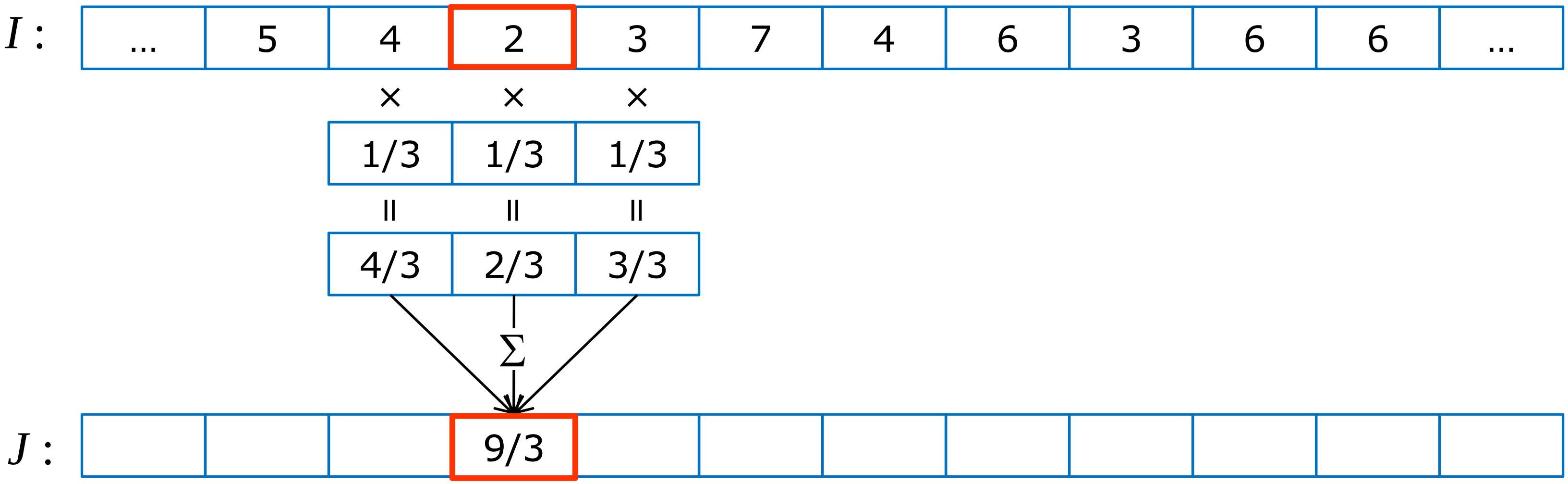
Linear, Shift-invariant filters

- **Linear**: every pixel is replaced by a linear combination of its neighbours
- **Shift-invariant**: the same operation is performed on every point on the image
- Basic & very useful filtering operations:
  - Correlation
  - Convolution
- Brief study of these filters in the simplest case of 1D images (i.e. row of pixels) & their extension to 2D

Optional, further reading here: 

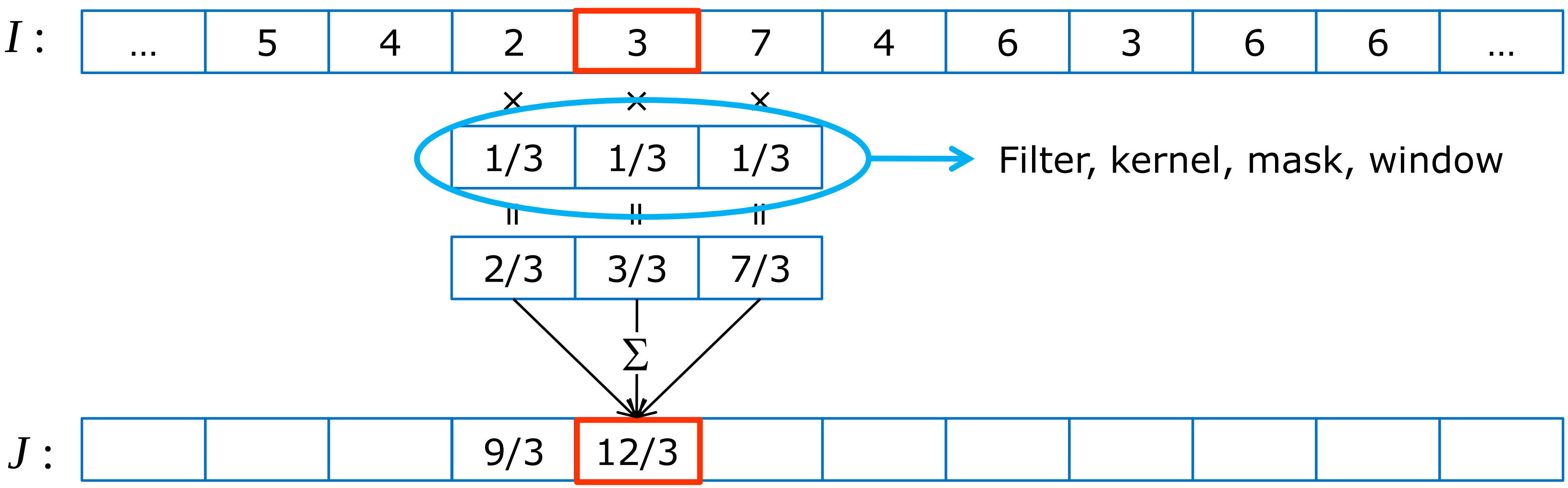
Correlation

- Averaging in a slightly different way



Correlation

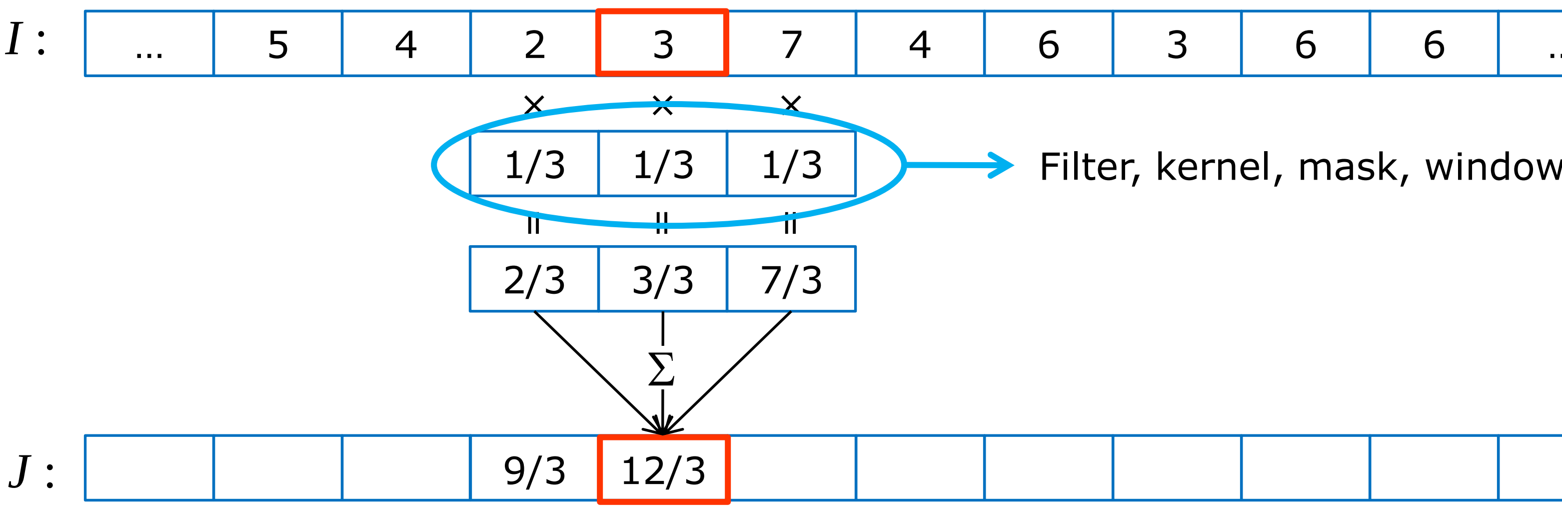
- Averaging in a slightly different way



- How to handle boundaries?
  - Ignore filtered values at boundaries
  - Pad with zeros
  - Pad with first/last image values

Correlation

- Averaging in a slightly different way

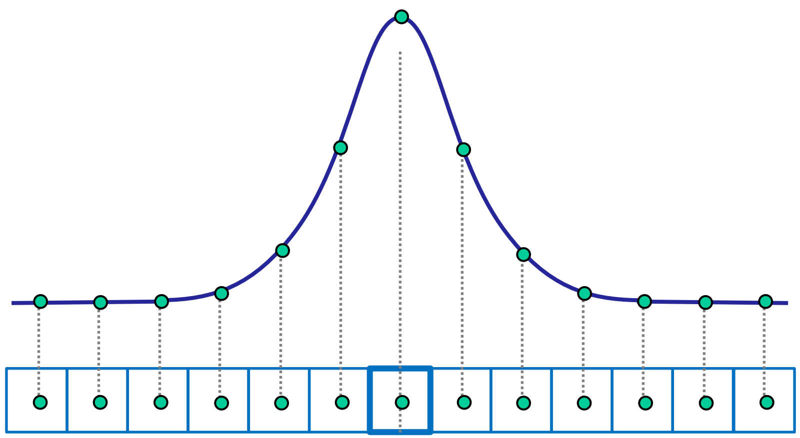


- Formally, Correlation is $J(x) = F \circ I(x) = \sum_{i=-N}^{N} F(i) I(x+i)$

- In this smoothing example $F(i) = \begin{cases} 1/3, i \in [-1,1] \\ 0 \ \ , i \notin [-1,1] \end{cases}$

Constructing Filter from a Continuous Fn

- Common practice for image smoothing: use a Gaussian $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$



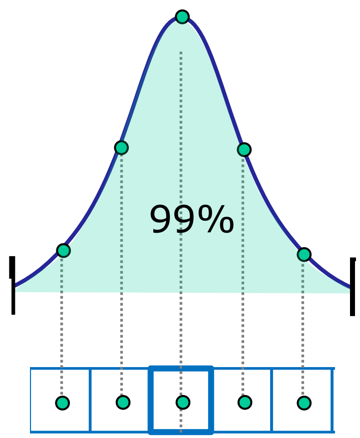
$\mu = 0$

σ : controls the amount of smoothing

- Near-by pixels have a bigger influence on the averaged value rather than more distant ones

Constructing Filter from a Continuous Fn

- Common practice for image smoothing: use a Gaussian $G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$



$\mu = 0$

σ : controls the amount of smoothing

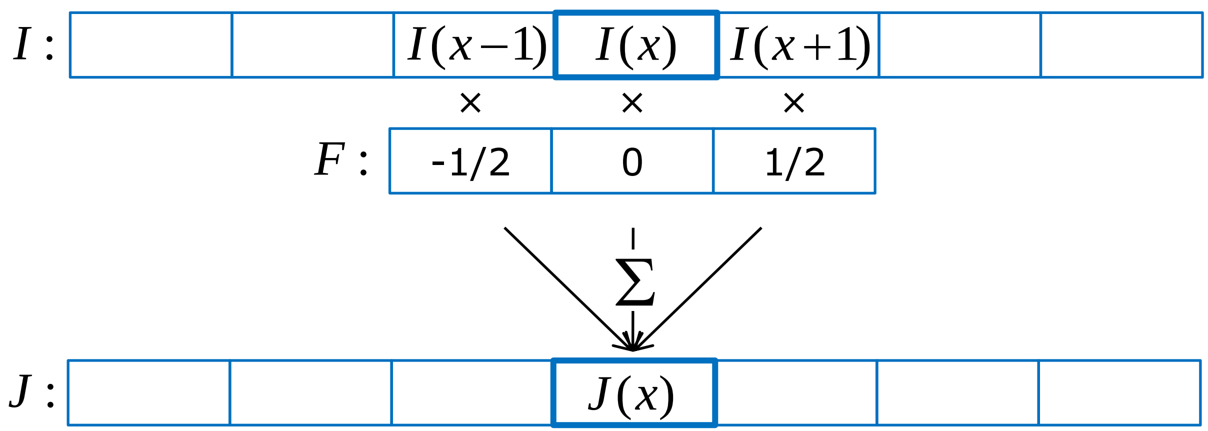
Normalize filter so that values always add up to 1

- Near-by pixels have a bigger influence on the averaged value rather than more distant ones

Taking derivatives with Correlation

- Derivative of an image: quantifies how quickly intensities change
  (along the direction of the derivative)

- Approximate a derivative operator:



$$J(x) = \frac{I(x+1) - I(x-1)}{2}$$

Matching using Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template

3	8	3

⇨ test it against all image locations

I :

3	2	4	1	3	8	4	0	3	8	7	7

- Similarity measure: Sum of Squared Differences (SSD)

$$\sum_{i=-N}^{N} \left(F(i) - I(x+i)\right)^2$$

J :

26	37	21	50	54	1	50	65	59	16	42	17

Matching using Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template

3	8	3

⇨ test it against all image locations

I :

3	2	4	1	3	8	4	0	3	8	7	7

- Similarity measure: Sum of Squared Differences (SSD)

$$\sum_{i=-N}^{N}(F(i)-I(x+i))^2 = \sum_{i=-N}^{i=N}(F(i))^2 + \sum_{i=-N}^{i=N}(I(x+i))^2 - 2\boxed{\sum_{i=-N}^{i=N}(F(i)I(x+i))}$$

Correlation

J :

26	37	21	50	54	1	50	65	59	16	42	17

- Similarity measure: Correlation?

Matching using Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template

3	8	3

⇨ test it against all image locations

$I:$

3	2	4	1	3	8	4	0	3	8	7	7

- Similarity measure: Sum of Squared Differences (SSD)

$$\sum_{i=-N}^{N}\left(F(i)-I(x+i)\right)^2 = \sum_{i=-N}^{i=N}\left(F(i)\right)^2 + \sum_{i=-N}^{i=N}\left(I(x+i)\right)^2 - 2\underbrace{\boxed{\sum_{i=-N}^{i=N}\left(F(i)I(x+i)\right)}}_{\textbf{Correlation}}$$

$J:$

26	37	21	50	54	1	50	65	59	16	42	17

- Similarity measure: Correlation?

$J:$

30	37	41	29	51	85	56	21	48	86	101	77

NCC: Normalized Cross Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template

3	8	3

⇨ test it against all image locations

I :

3	2	4	1	3	8	4	0	3	8	7	7

- Correlation value is affected by the magnitude of intensities
- Similarity measure: Normalized Cross Correlation (NCC)

$$\frac{\sum_{i=-N}^{i=N}\left(F(i)I(x+i)\right)}{\sqrt{\sum_{i=-N}^{i=N}\left(F(i)\right)^2}\sqrt{\sum_{i=-N}^{i=N}\left(I(x+i)\right)^2}}$$

J :

0.919	0.759	0.988	0.628	0.655	0.994	0.691	0.464	0.620	0.860	0.876	0.859

ZNCC: Zero-mean Normalized Cross Correlation

- Find locations in an image that are similar to a ***template***
- Filter = template

3	8	3

⇨ test it against all image locations

$I:$

3	2	4	1	3	8	4	0	3	8	7	7

- For better invariance to intensity changes

Similarity measure: Zero-mean Normalized Cross Correlation (ZNCC)

$$\frac{\sum_{i=-N}^{i=N}\left(F(i)-\mu_F\right)\left(I(x+i)-\mu_{I_x}\right)}{\sqrt{\sum_{i=-N}^{i=N}\left(F(i)-\mu_F\right)^2}\sqrt{\sum_{i=-N}^{i=N}\left(I(x+i)-\mu_{I_x}\right)^2}}$$

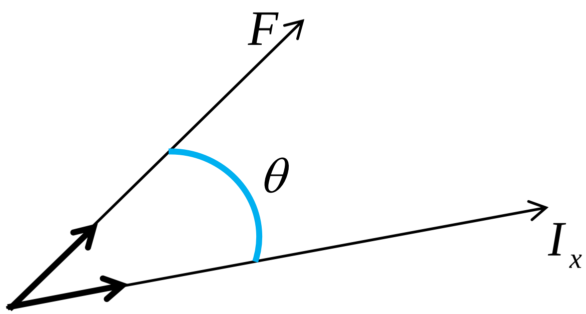
where

$$\begin{cases} \mu_F = \dfrac{\sum_{i=-N}^{N} F(i)}{2N+1} \\ \mu_{I_x} = \dfrac{\sum_{i=-N}^{N} I(x+i)}{2N+1} \end{cases}$$

Correlation as an inner product

- Considering the filter F and the portion of the image I_x as vectors then their correlation is:

$$\langle F, I_x \rangle = \|F\| \|I_x\| \cos\theta$$



- In NCC and ZNCC we considering the unit vectors of F and I_x , hence we measure their similarity based on the angle θ

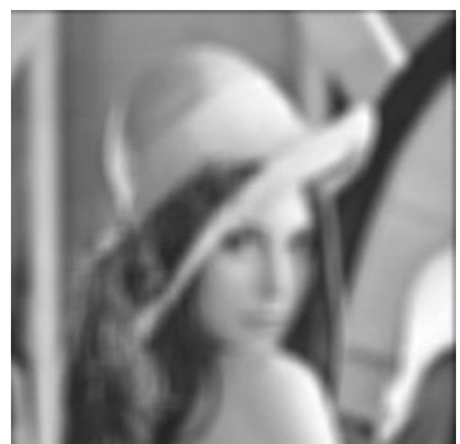
Correlation in 2D

$$F \circ I(x,y) = \sum_{j=-M}^{M} \sum_{i=-N}^{N} F(i,j) I(x+i, y+j)$$

- Example: Constant averaging filter $F = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$

- If $size(F) = (2N+1)^2$ i.e. this is a square filter

- 2D Correlation ⇨ no. multiplications per pixel = $(2N+1)^2$
  no. additions per pixel = $(2N+1)^2 - 1$

This example was generated with a 21x21 mask

Correlation in 2D

$$F \circ I(x,y) = \sum_{j=-M}^{M} \sum_{i=-N}^{N} F(i,j) I(x+i, y+j)$$

"separable" filter

- Example: Constant averaging filter $F = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$

- If $size(F) = (2N+1)^2$ i.e. this is a square filter

- 2D Correlation ⇨ no. multiplications per pixel = $(2N+1)^2$
  no. additions per pixel = $(2N+1)^2 - 1$
- 2 × 1D Correlation ⇨ no. multiplications per pixel = $2 \times (2N+1)$
  no. additions per pixel = $2 \times 2N$

Correlation in 2D

$$F \circ I(x,y) = \sum_{j=-M}^{M} \sum_{i=-N}^{N} F(i,j) I(x+i, y+j)$$

"separable" filter

- Example: Constant averaging filter $F = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$

- If $size(F) = (2N+1)^2$ i.e. this is a square filter

- 2D Correlation ⇨ no. multiplications per pixel = $(2N+1)^2$
  no. additions per pixel = $(2N+1)^2 - 1$
- 2 × 1D Correlation ⇨ no. multiplications per pixel = $2 \times (2N+1)$
  no. additions per pixel = $2 \times 2N$
- 2 × 1D Correlation (with const. factor) ⇨ no. multiplications per pixel = 1
  no. additions per pixel = $2 \times 2N$

2D Gaussian Smoothing

- A general, 2D Gaussian

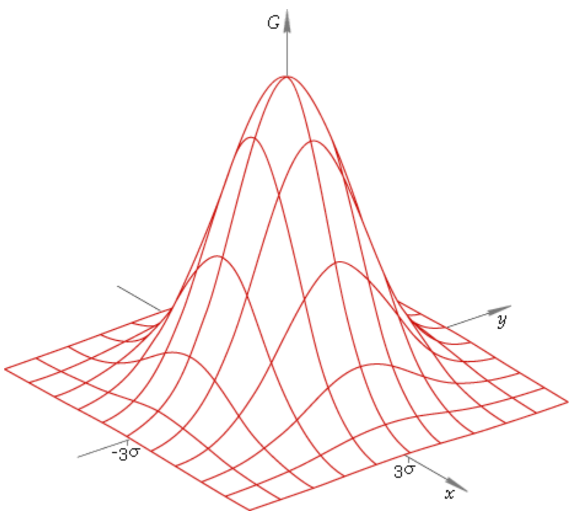
$$G(x,y)=\frac{1}{2\pi|S|^{1/2}}e^{-\frac{1}{2}\begin{pmatrix}x\\ y\end{pmatrix}S^{-1}\begin{pmatrix}x & y\end{pmatrix}}$$

- We usually want to smooth by the same amount in both x and y directions $S=\begin{bmatrix}\sigma^2 & 0\\ 0 & \sigma^2\end{bmatrix}$

- So this simplifies to:

$$G_\sigma(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}=\underbrace{\boxed{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}}}}_{G_\sigma(x)}\cdot\underbrace{\boxed{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{y^2}{2\sigma^2}}}}_{G_\sigma(y)}$$

- Another separable filter



Convolution

- Convolution is equivalent to: Correlation with a flipped filter before correlating
- CONVOLUTION $J(x) = F * I(x) = \sum_{i=-N}^{N} F(i) I(x-i)$
- CORRELATION $J(x) = F \circ I(x) = \sum_{i=-N}^{N} F(i) I(x+i)$

So if $F = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

$F' = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$

Then, $J(x) = J'(x)$

- Likewise, in 2D we flip the filter both horizontally & vertically

$$J(x,y) = F * I(x,y) = \sum_{j=-M}^{M} \sum_{i=-N}^{N} F(i,j) I(x-i, y-j)$$

- Key difference between correlation and convolution is that **convolution is associative**:

$$F * (G * I) = (F * G) * I$$

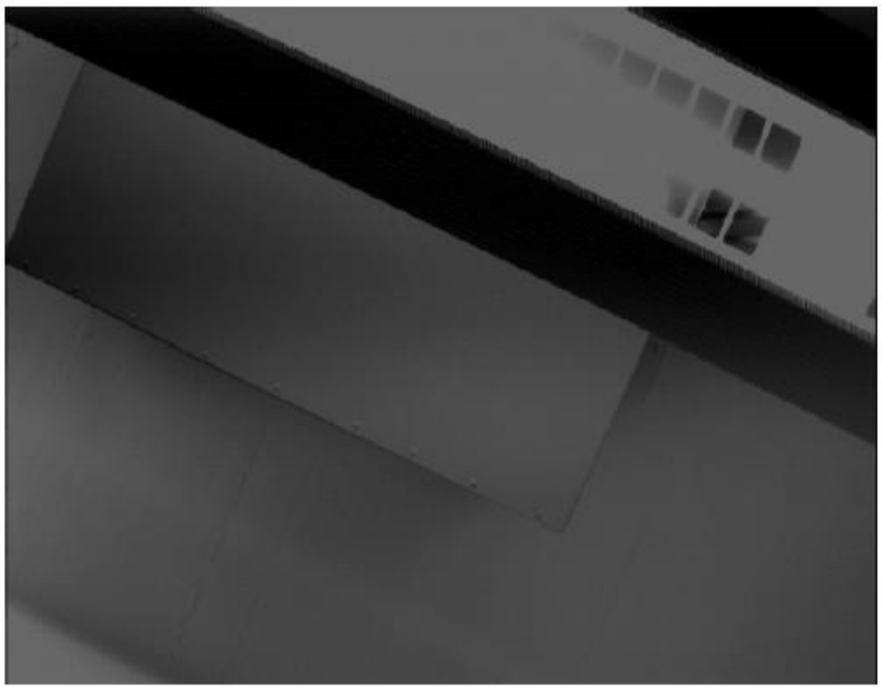
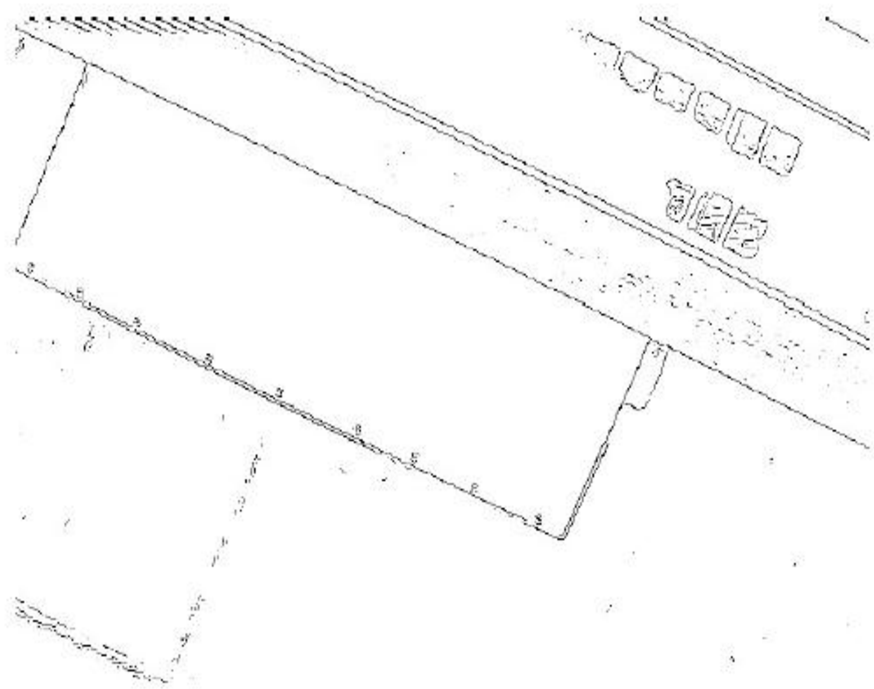
- Very useful!
- Example: smooth an image & take its derivative ⇨ convolve the Derivative filter with the Gaussian Filter & convolve the resulting filter with the image

Convolution Demo

- Interactive demo of (continuous) convolution here: 
  (suggested by one of the students)

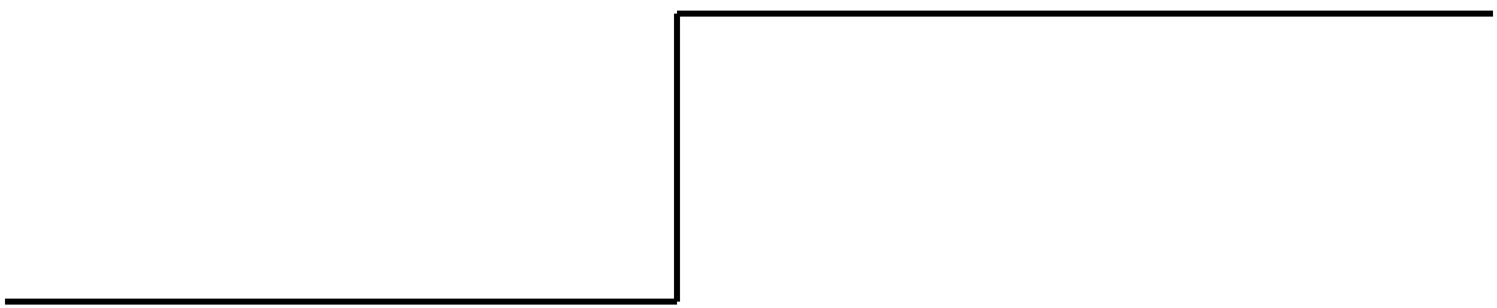
Edge Detection

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

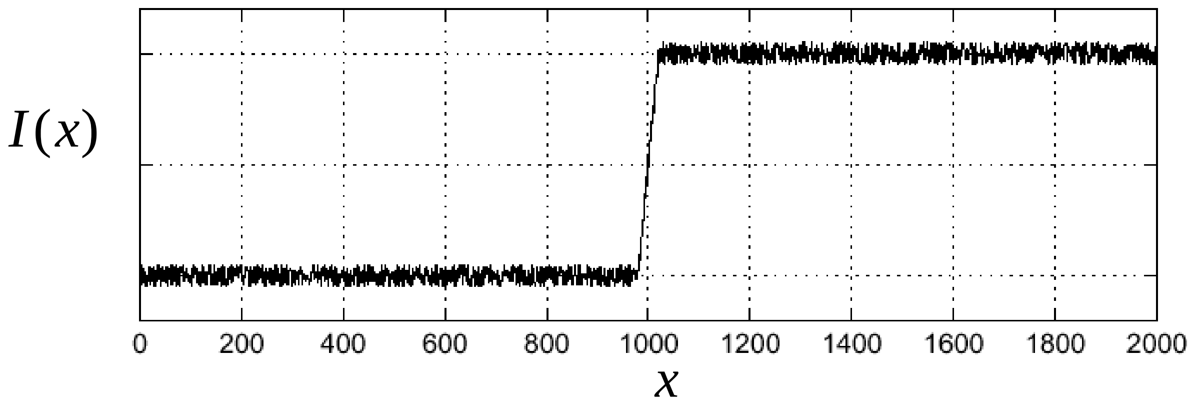
Edge = intensity discontinuity in one direction

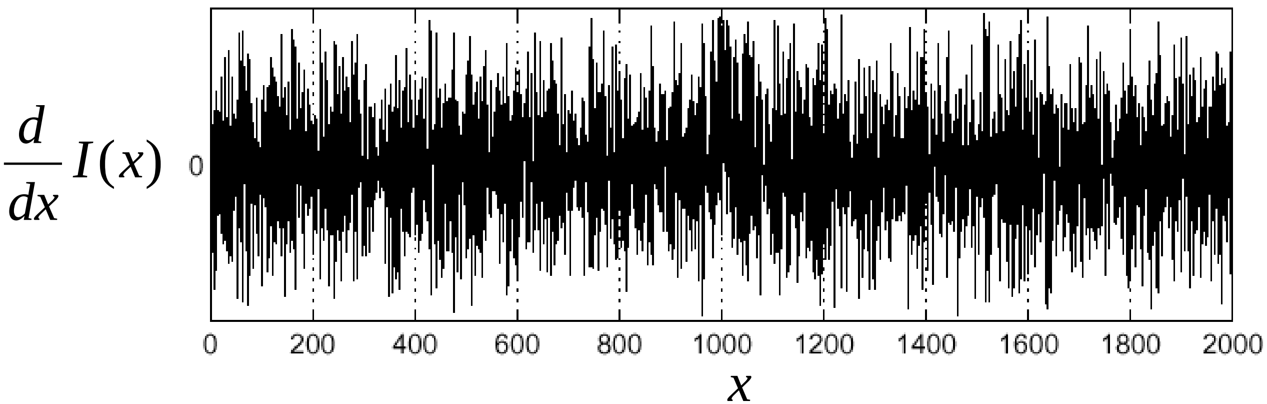
- Edges correspond to sharp changes of intensity
- Change is measured by 1st order derivative in 1D
- Big intensity change ⇨ magnitude of derivative is large
- Or 2nd order derivative is zero.



1D Edge detection

- Image intensity shows an obvious change





- Where is the edge? ⇨ image noise cannot be ignored

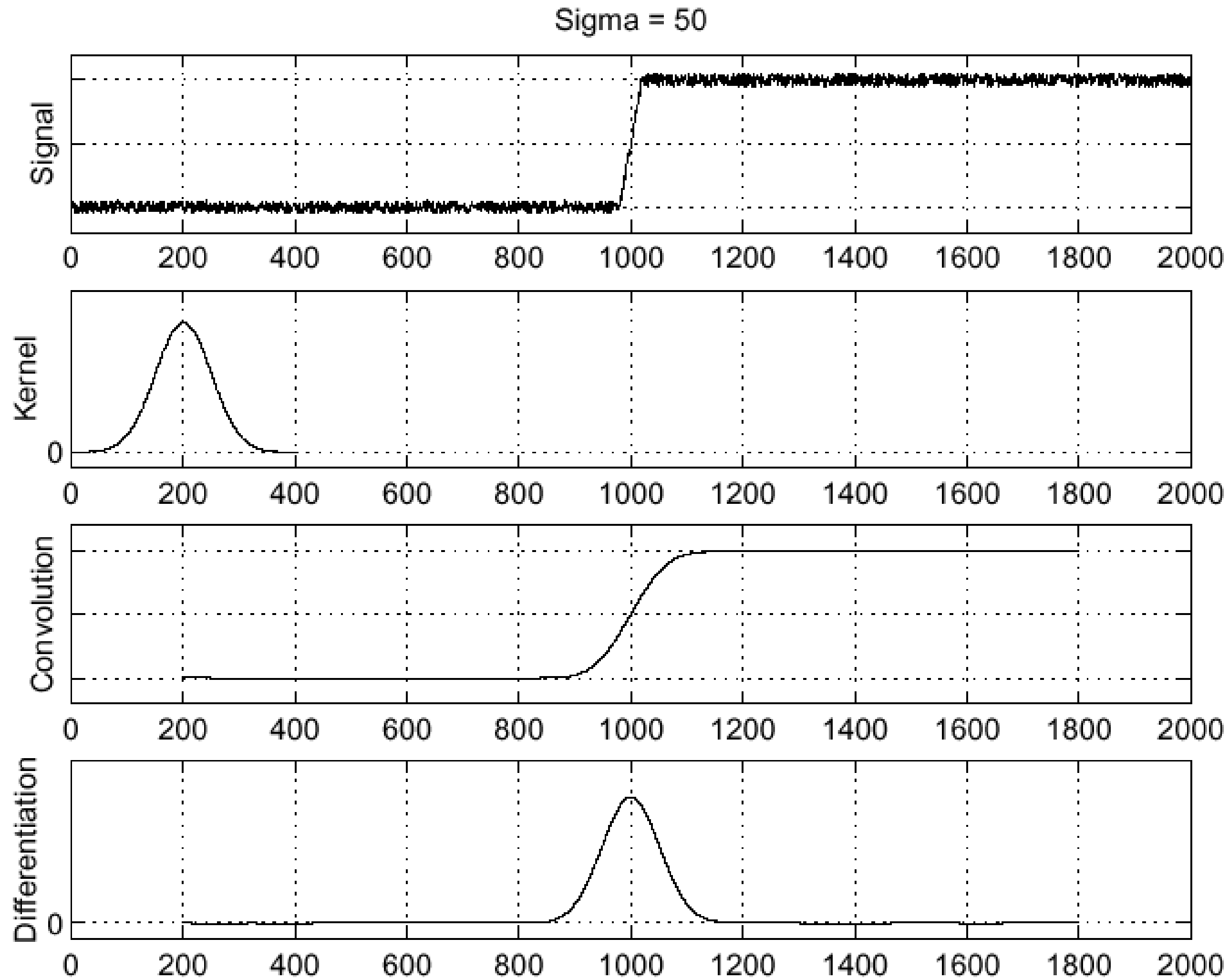
Solution: smooth first

$$I(x)$$

$$G_\sigma(x)$$

$$s(x) = I(x) * G_\sigma(x)$$

$$s'(x) = \frac{d}{dx}\big(s(x)\big)$$



- Where is the edge? At the extrema of $s'(x)$

Derivative theorem of convolution

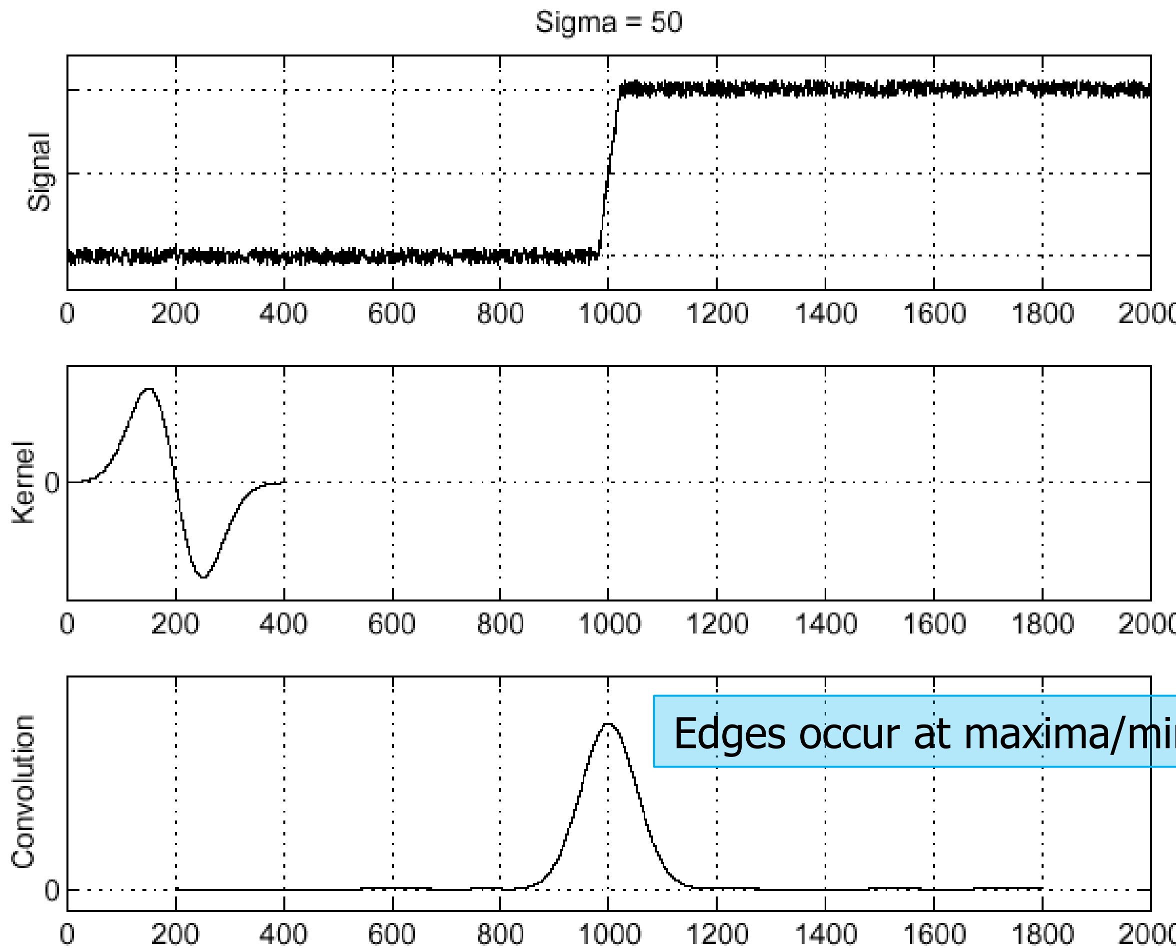
- $$s'(x) = \frac{d}{dx}\left(G_\sigma(x) * I(x)\right) = G'_\sigma(x) * I(x)$$

- This saves us one operation:

$I(x)$

$$G'_\sigma(x) = \frac{d}{dx} G_\sigma(x)$$

$$s'(x) = G'_\sigma(x) * I(x)$$



Edges occur at maxima/minima of $s'(x)$

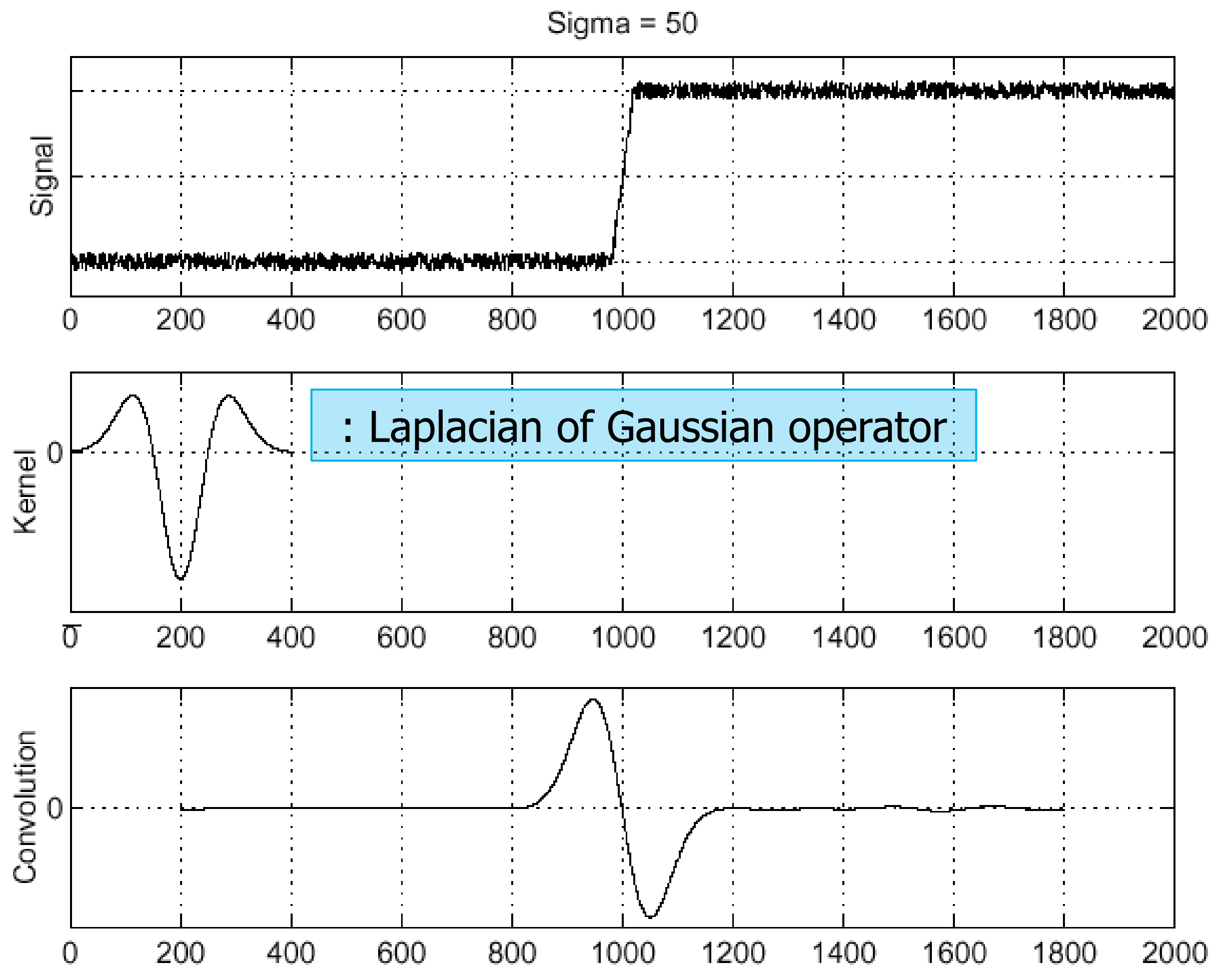
Zero-crossings

- Locations of Maxima/minima in $s'(x)$ are equivalent to zero-crossings in $s''(x)$

$$I(x)$$

$$G''_\sigma(x) = \frac{d^2}{dx^2} G_\sigma(x)$$

$$s''(x) = G''_\sigma(x) * I(x)$$



2D Edge detection

- Find gradient of smoothed image in both directions

Usually use a separable filter such that:
$G_\sigma(x,y) = G_\sigma(x) G_\sigma(y)$

$$\nabla S = \nabla(G_\sigma * I) = \begin{bmatrix} \frac{\partial(G_\sigma * I)}{\partial x} \\ \frac{\partial(G_\sigma * I)}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial G_\sigma}{\partial x} * I \\ \frac{\partial G_\sigma}{\partial y} * I \end{bmatrix} = \begin{bmatrix} G'_\sigma(x) G_\sigma(y) * I \\ G_\sigma(x) G'_\sigma(y) * I \end{bmatrix}$$

- Discard pixels with $|\nabla S|$ below a certain below a certain threshold

- **Non-maximal suppression**: identify local maxima of $|\nabla S|$ along the directions $\pm|\nabla S|$

2D Edge detection: Example

I : original image (Lena image)

2D Edge detection: Example

$$\nabla S = \nabla\left(G_\sigma * I\right)$$

$|\nabla S|$: Edge strength

2D Edge detection: Example

Thresholding $|\nabla S|$

2D Edge detection: Example

Thinning: non-maximal suppression

Image Feature Extraction

- Edges (seen before)
- **Points:**
  - **Harris corners**
  - **SIFT features**

Example: Build a Panorama

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

Example: Build a Panorama

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

Example: Build a Panorama

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

Example: Build a Panorama

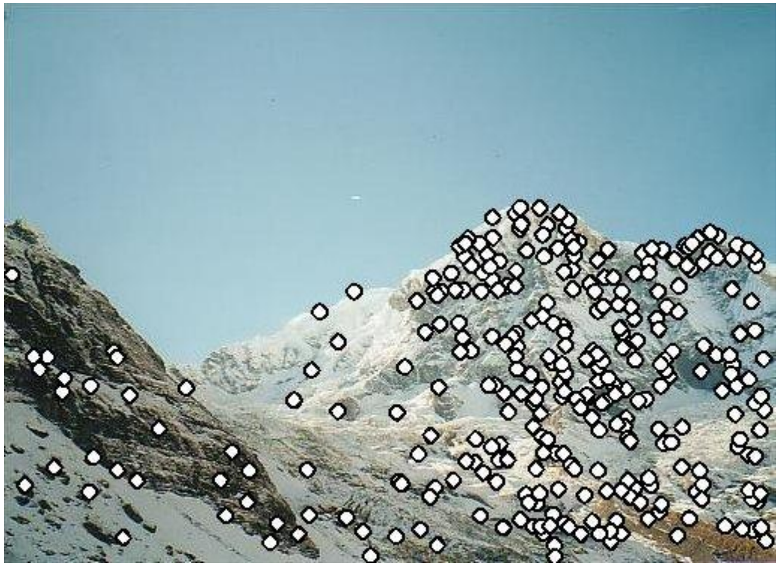
This panorama was generated using **AUTOSTITCH** (freeware), available at
http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html

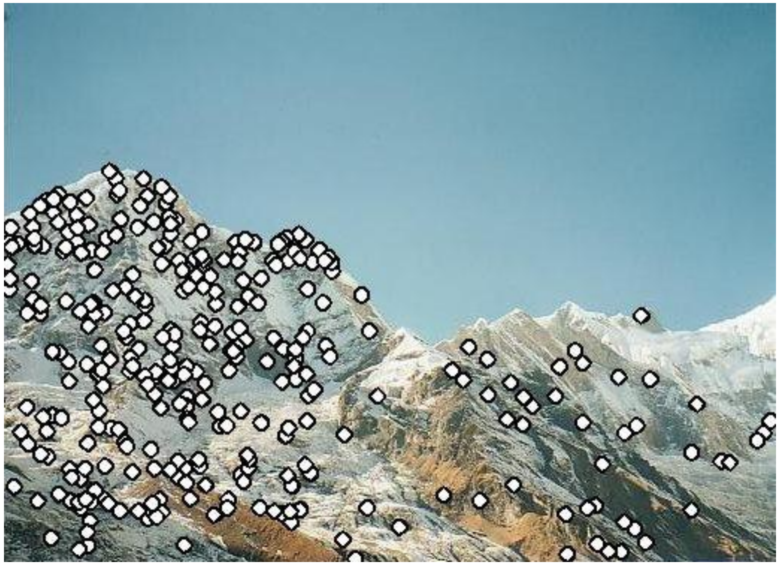
How do we build a panorama?

- We need to match (align) images

How do we build a panorama?

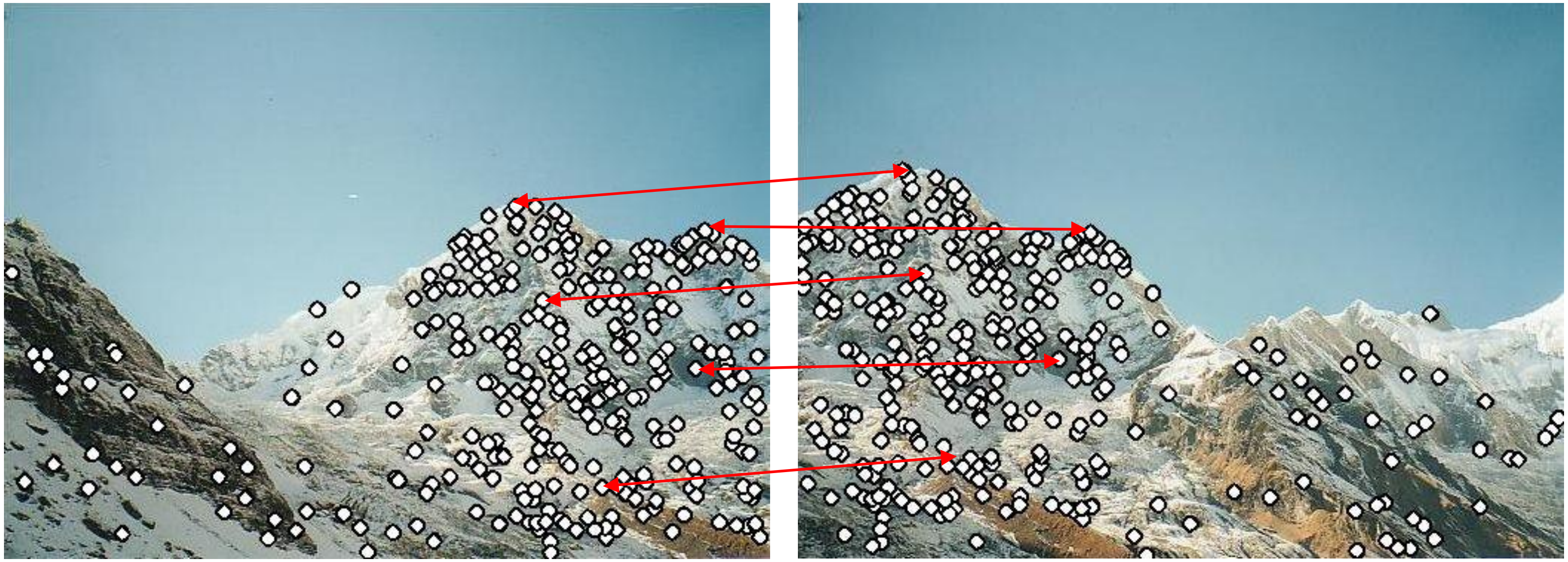
- Detect feature points in both images





How do we build a panorama?

- Detect feature points in both images
- Find corresponding pairs



How do we build a panorama?

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

Matching with Features

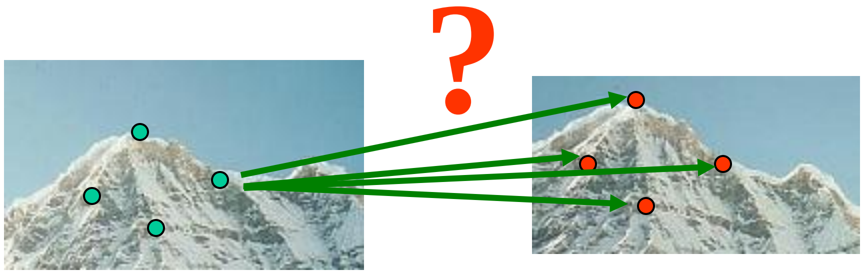
- Problem 1:
  - Detect the **same** point **independently** in both images


no chance to match!

We need a repeatable detector

Matching with Features

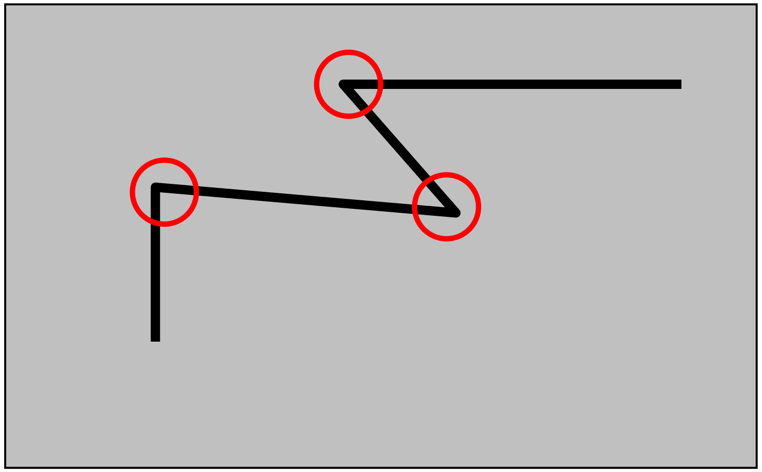
- Problem 2:
  - For each point correctly recognize the corresponding one



We need a reliable and distinctive descriptor

More motivation…

- Feature points are used also for:
  - Robot navigation (we will see in the next exercise)
  - Object recognition (live demo)
  - Image alignment (panoramas)
  - 3D reconstruction
  - Motion tracking
  - Indexing and database retrieval -> Google Images or http://tineye.com
  - … other



Harris Corner Detector

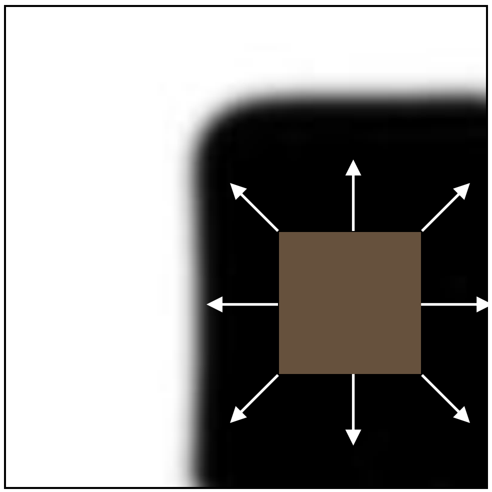
C.Harris, M.Stephens. "A Combined Corner and Edge Detector". 1988

Identifying Corners

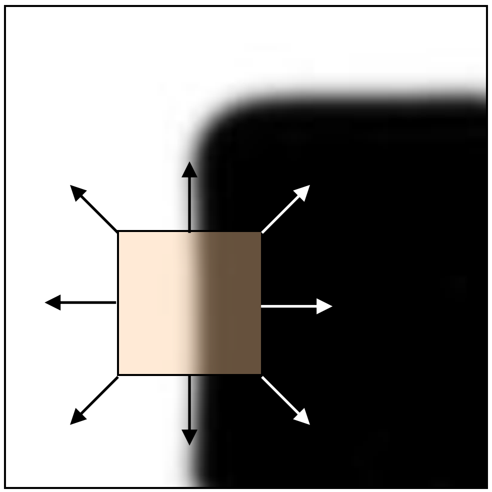
- Key property: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and distinctive

Identifying Corners

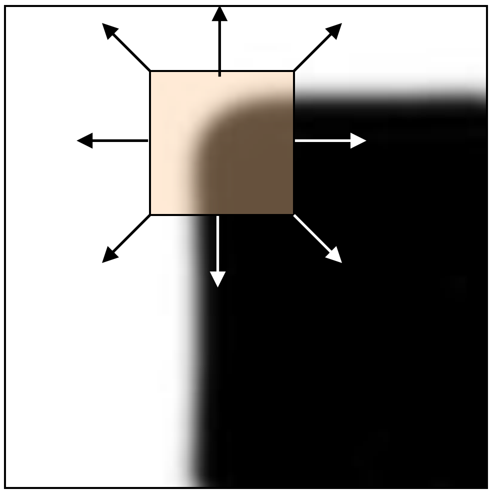
- Shifting a window in **any direction** should give *a* **large change** in intensity in at least 2 directions



"flat" region:
no intensity change



"edge":
no change along the edge direction



"corner":
significant change in at least 2 directions

How do we implement this?

- Consider taking an image patch P centered on (x, y) and shifting it by $(\Delta x, \Delta y)$
  The Sum of Squared Differences between these two patches is given by:

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} \left(I(x, y) - I(x + \Delta x, y + \Delta y) \right)^2$$

- Let $I_x = \dfrac{\partial I(x, y)}{\partial x}$ and $I_y = \dfrac{\partial I(x, y)}{\partial y}$. Approximating with a 1st order Taylor expansion:

$$I(x + \Delta x, y + \Delta y) \approx I(x, y) + I_x(x, y)\Delta x + I_y(x, y)\Delta y$$

- This produces the approximation

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \left(I_x(x, y)\Delta x + I_y(x, y)\Delta y \right)^2$$

- Which can be written in a matrix form as

$$SSD(\Delta x, \Delta y) \approx \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

How do we implement this?

$$SSD(\Delta x, \Delta y) \approx \sum_{x,y \in P} \left(I_x(x,y)\Delta x + I_y(x,y)\Delta y \right)^2 = \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

- M is the "second moment matrix" $M = \sum_{x,y \in P} \begin{bmatrix} I_x^{\ 2} & I_x I_y \\ I_x I_y & I_y^{\ 2} \end{bmatrix}$

- Since M is symmetric, we can rewrite M as

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

where λ_1 and λ_2 are the eingenvalues of M

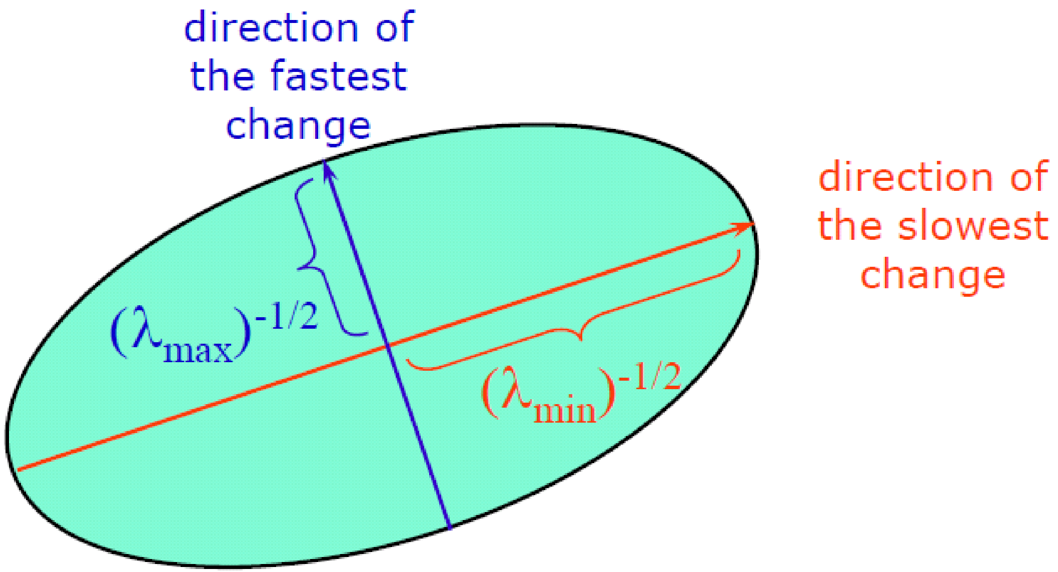
How do we implement this?

- The Harris detector analyses the **eigenvalues** of M to decide if we are in presence of a corner or not
  ⇨ i.e. looks for large intensity changes in at least 2 directions

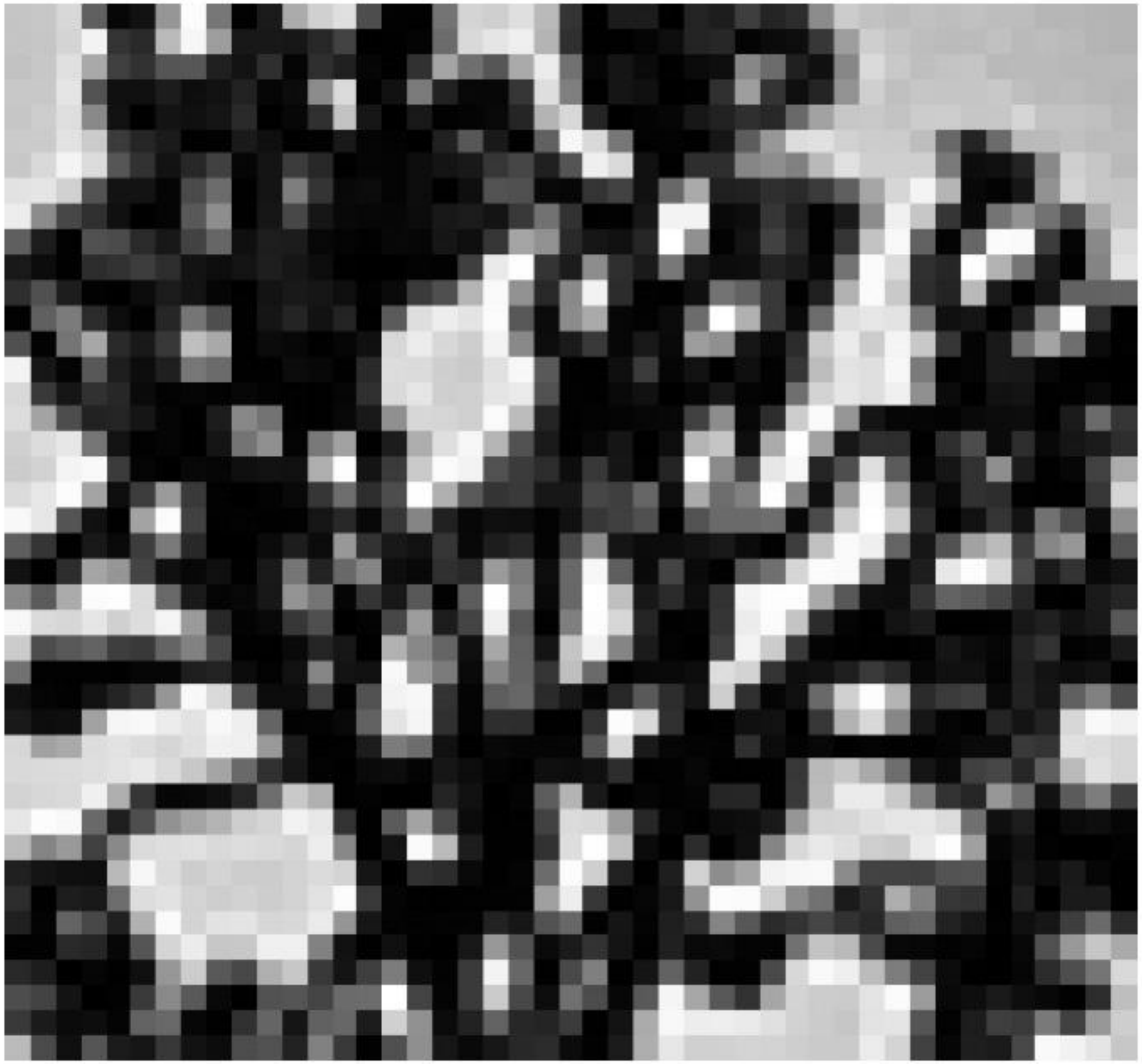
- We can visualize M as an **ellipse** with axis-lengths determined by the eigenvalues and orientation determined by *R*

$$\begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = const.$$

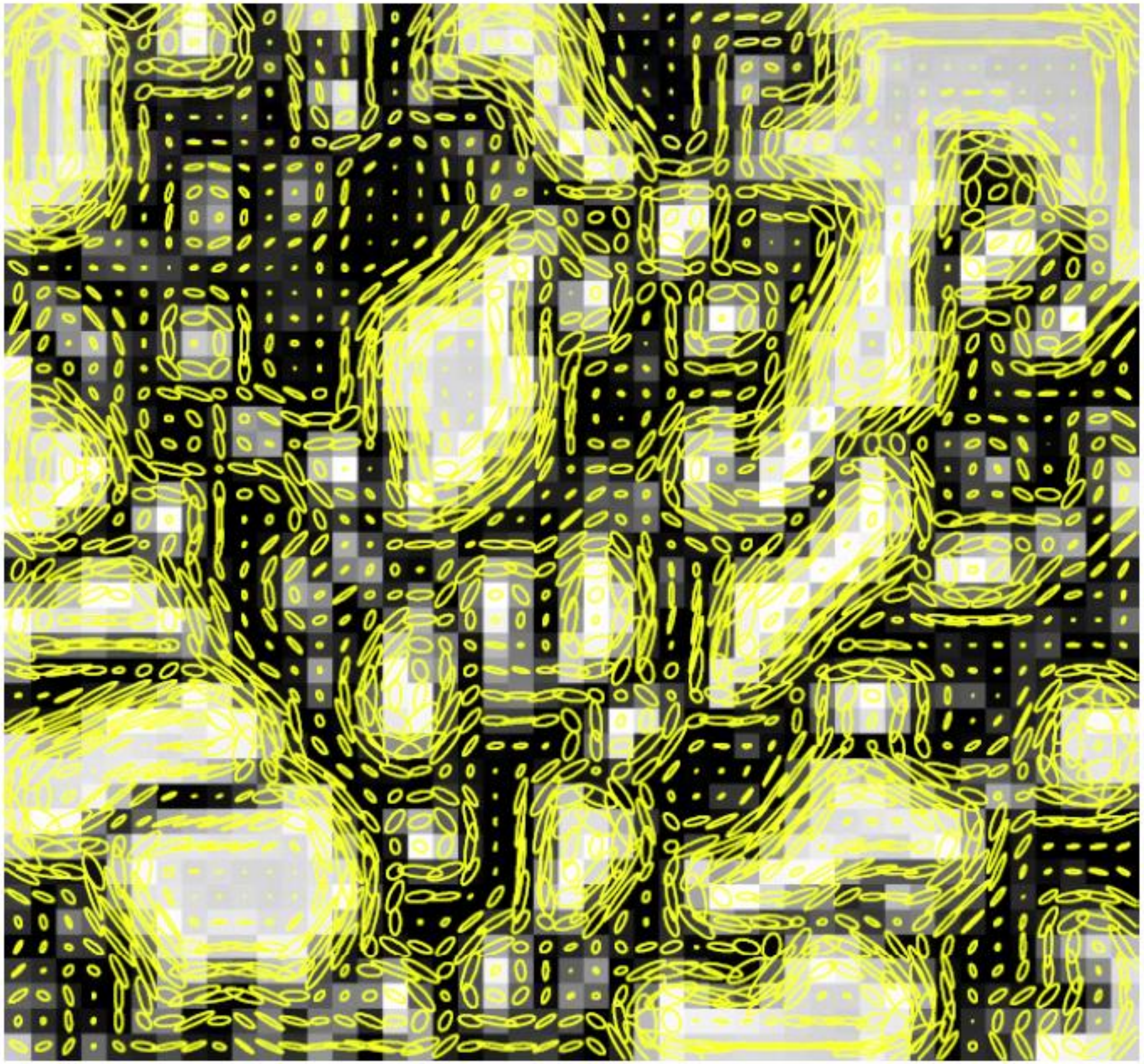
$$M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$



Visualization of second moment matrices



Visualization of second moment matrices



Corner response function

Does the patch P describe a corner or not?

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

- **No structure:** $\lambda_1 \approx \lambda_2 \approx 0$
  SSD is almost constant in all directions, so it's a **flat** region
- **1D structure:** $\lambda_1 \approx 0, \lambda_2$ is large (or vice versa)
  SSD has a large variation only in one direction, which is the one perpendicular to the **edge**.
- **2D structure:** λ_1, λ_2 are both large
  SSD has large variations in all directions and then we are in presence of a **corner**.

Computation of eigenvalues is expensive ⇨ Harris and Stephens suggested using a "**cornerness function**" instead:

$$C = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = det(M) - \kappa \cdot trace^2(M)$$

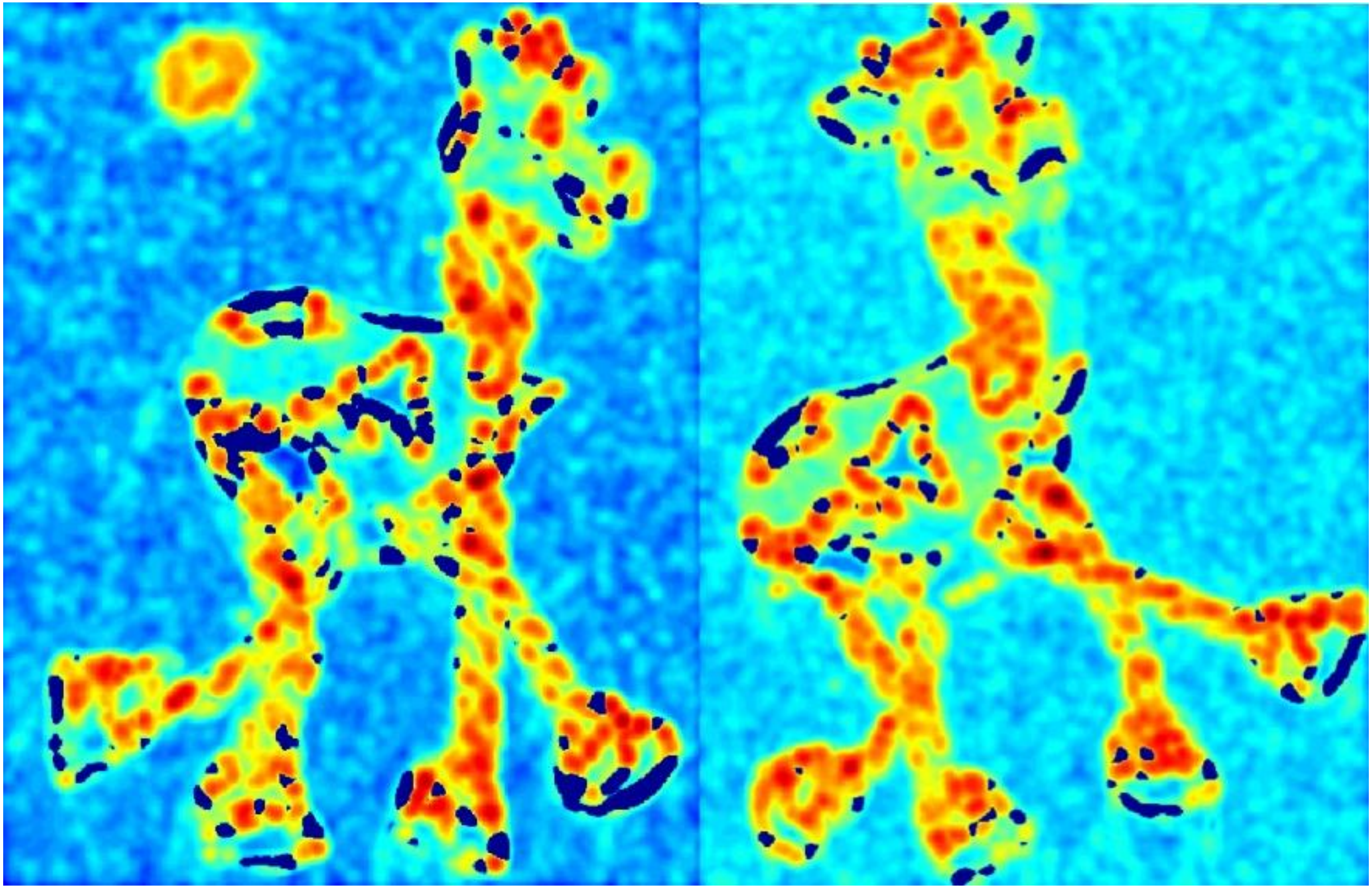
Where k is a between 0.04 and 0.15

- Last step of Harris corner detector: extract local maxima of the cornerness function

Harris Detector: Workflow

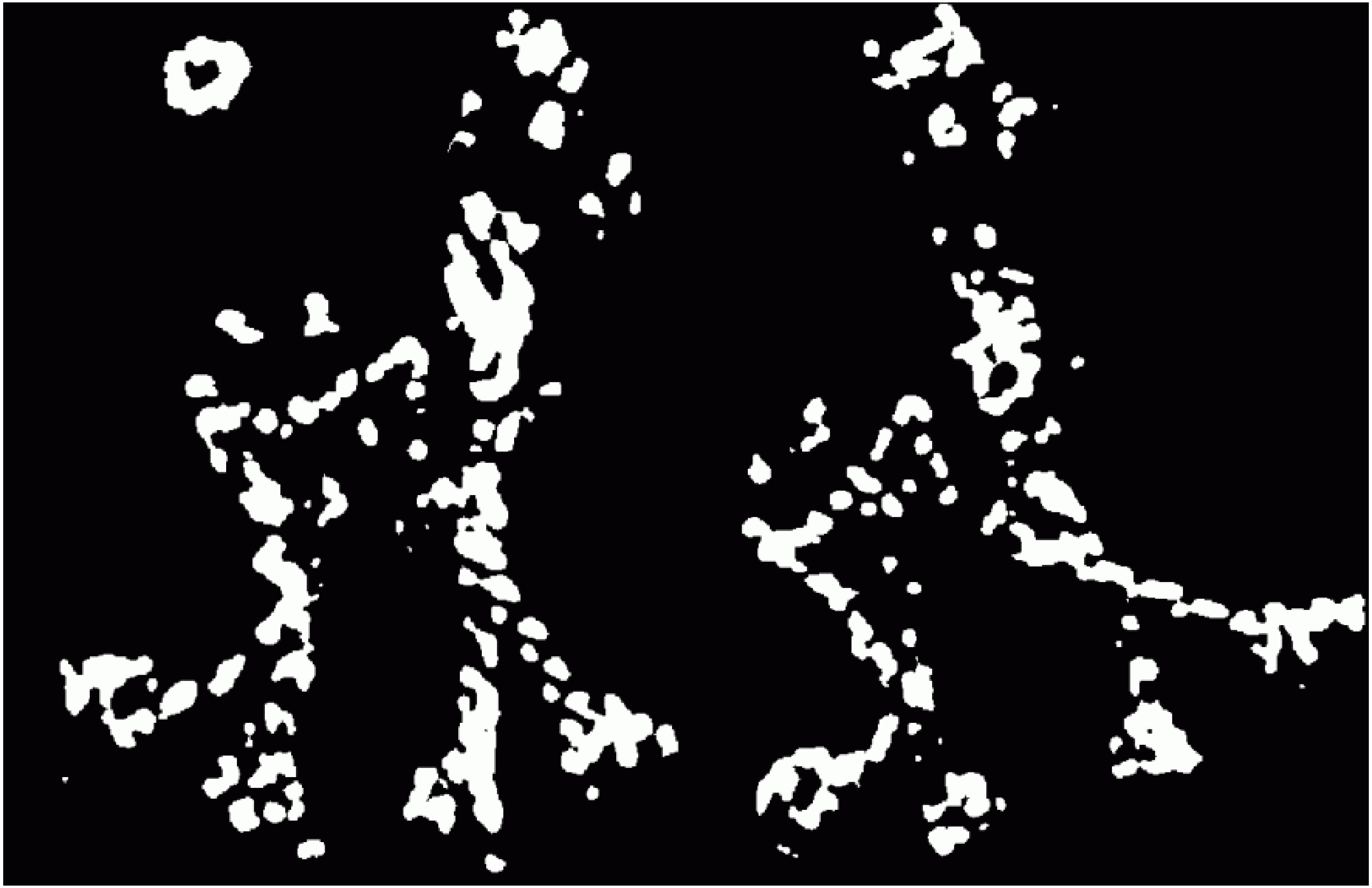
Harris Detector: Workflow

- Compute corner response *C*



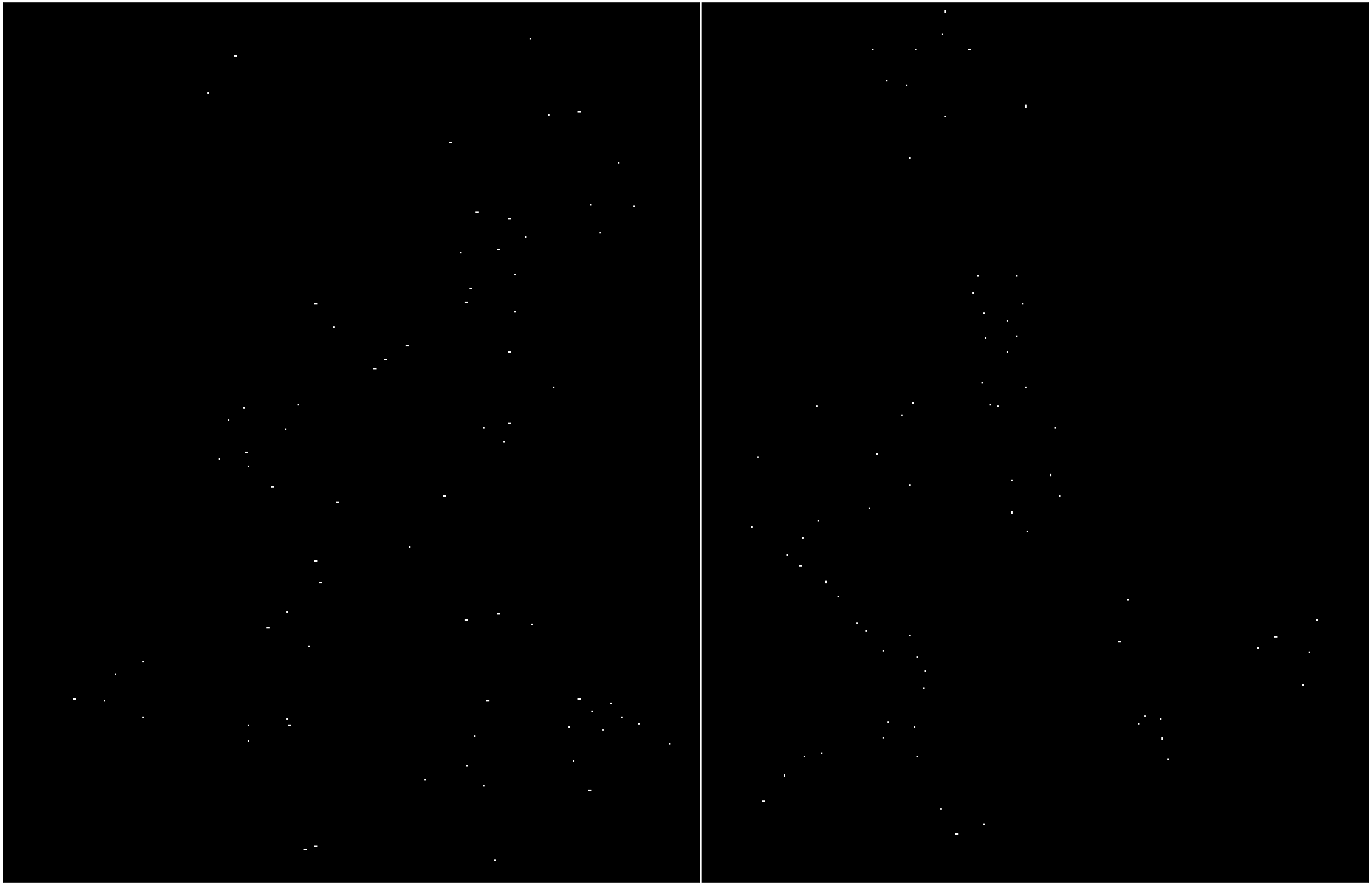
Harris Detector: Workflow

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



Harris Detector: Workflow

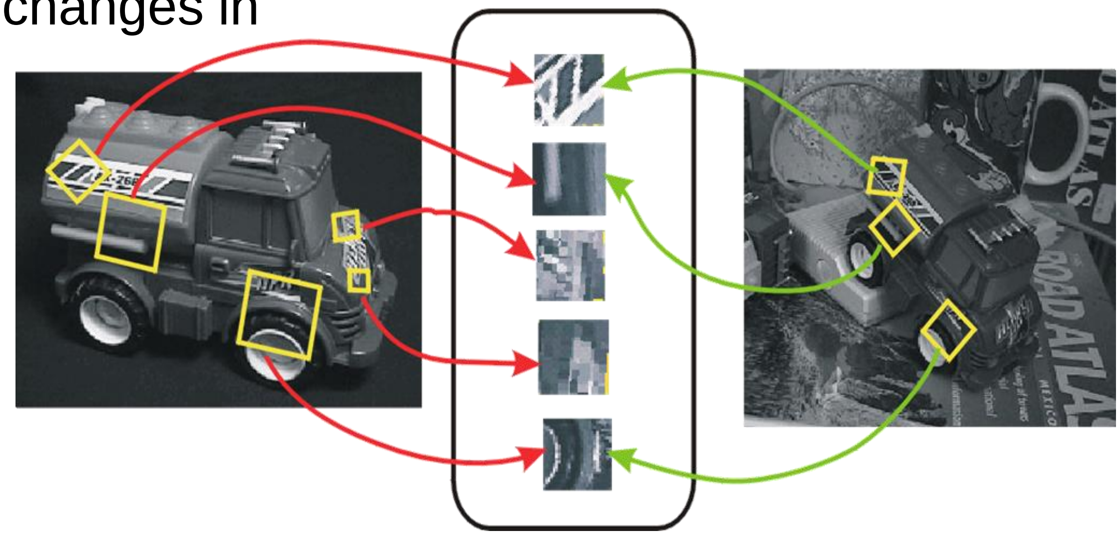
- Take only the points of local maxima of thresholded *C*



Harris Detector: Workflow

Harris detector: properties

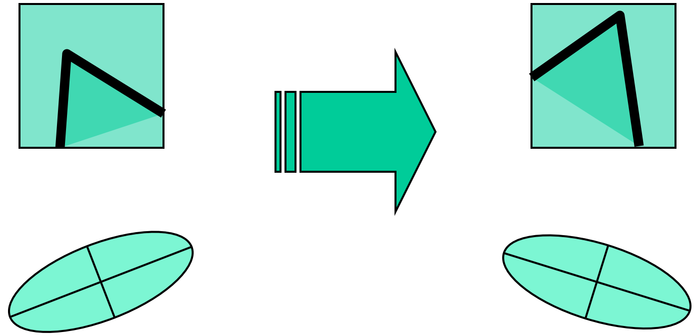
- How does the Harris detector behave to common image transformations?
- Will we be able to re-detect the same image patches (Harris corners) when the image exhibits changes in
  - Rotation,
  - View-point,
  - Zoom,
  - Illumination?



  - Identify properties of detector & adapt accordingly

Harris Detector: Some Properties

- **Rotation invariance**

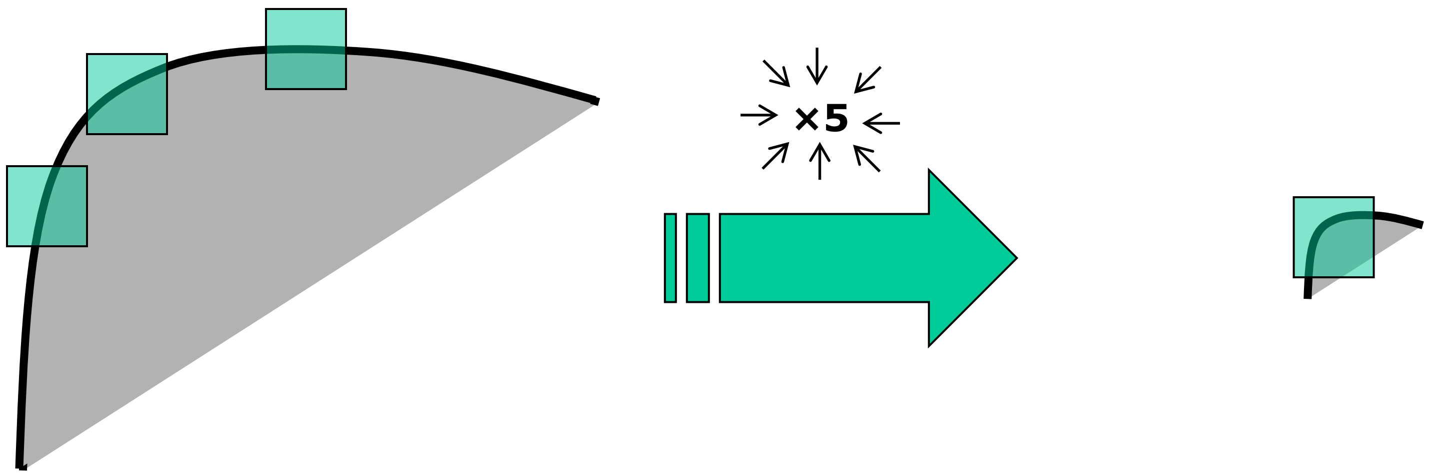


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response C is invariant to image rotation

Harris Detector: Some Properties

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



All points will be classified as edges

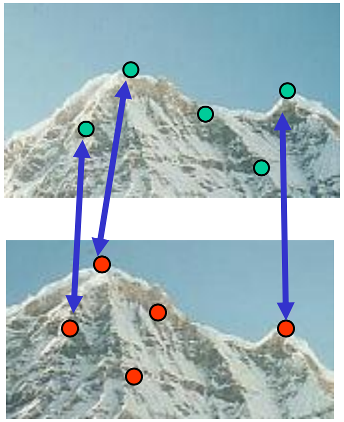
Corner !

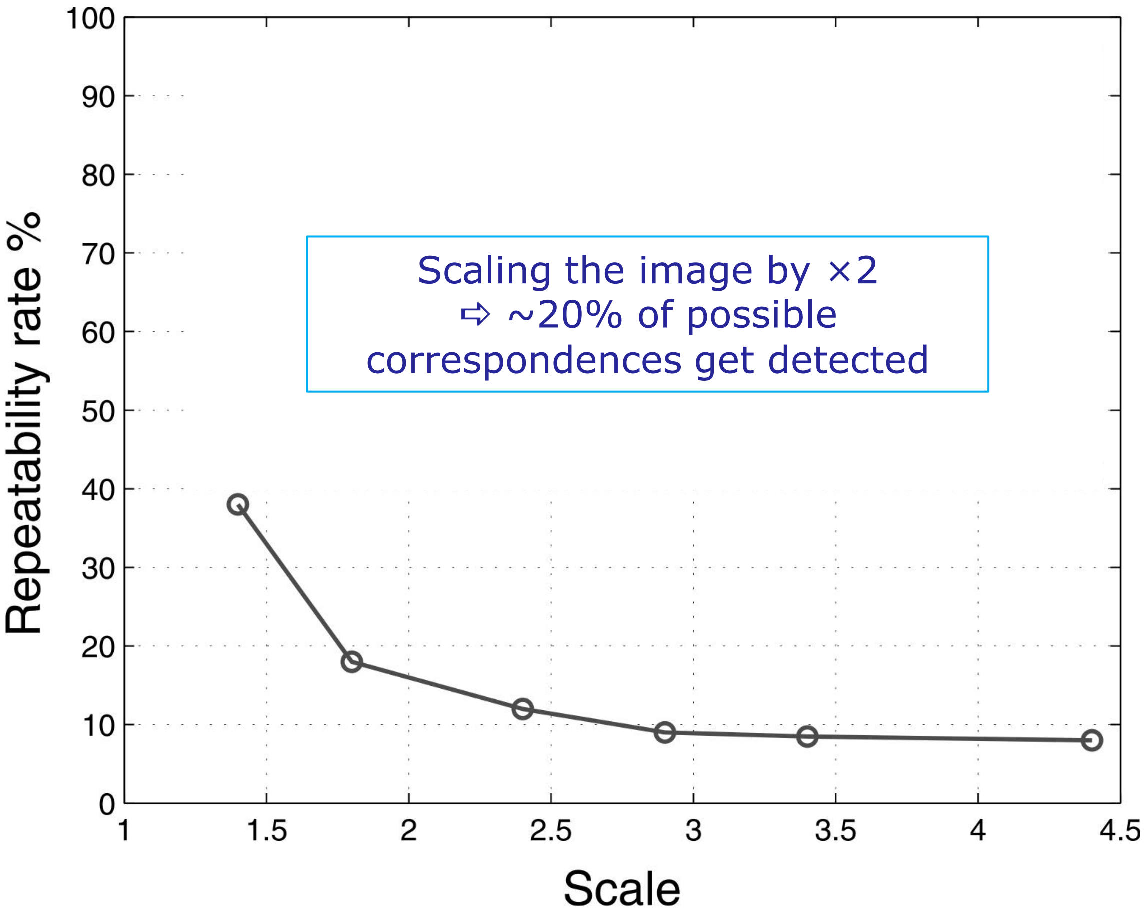
Harris Detector: Some Properties

- Quality of Harris detector for different scale changes

Repeatability rate:

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



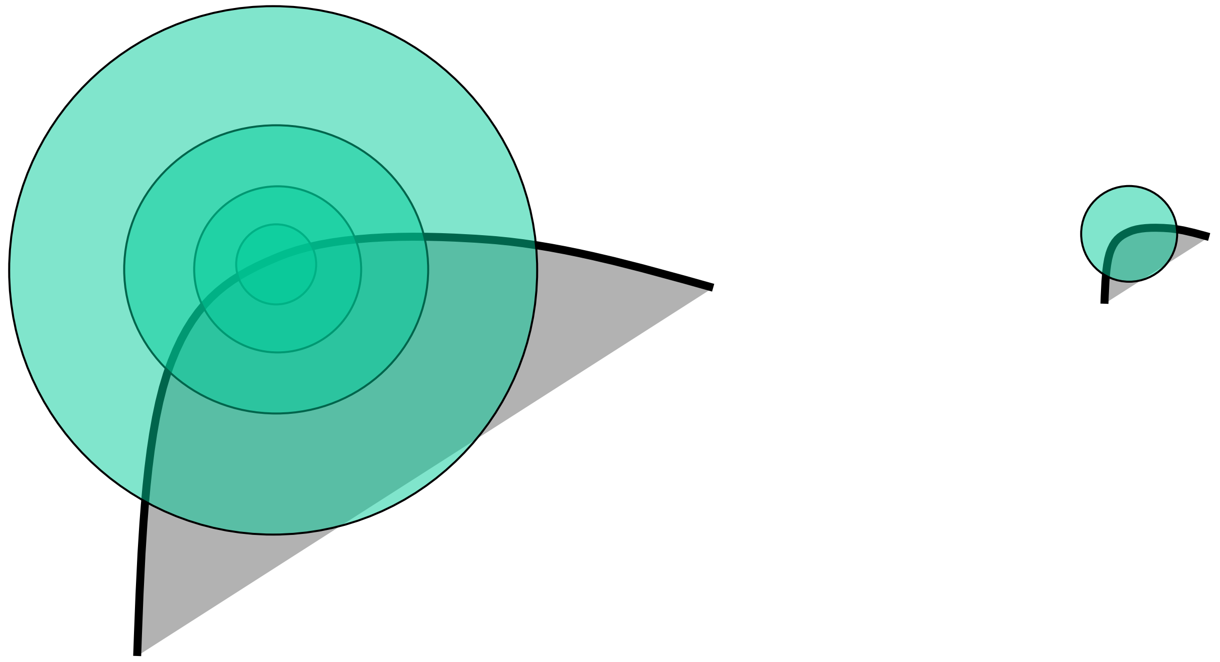


Summary on Harris properties

- Harris detector: probably the most widely used & known corner detector
- The detection is Invariant to
  - Rotation
  - Linear intensity changes
  - **note:** to make the matching invariant to these we need a suitable matching criterion (e.g. SSD in not Rotation / 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)

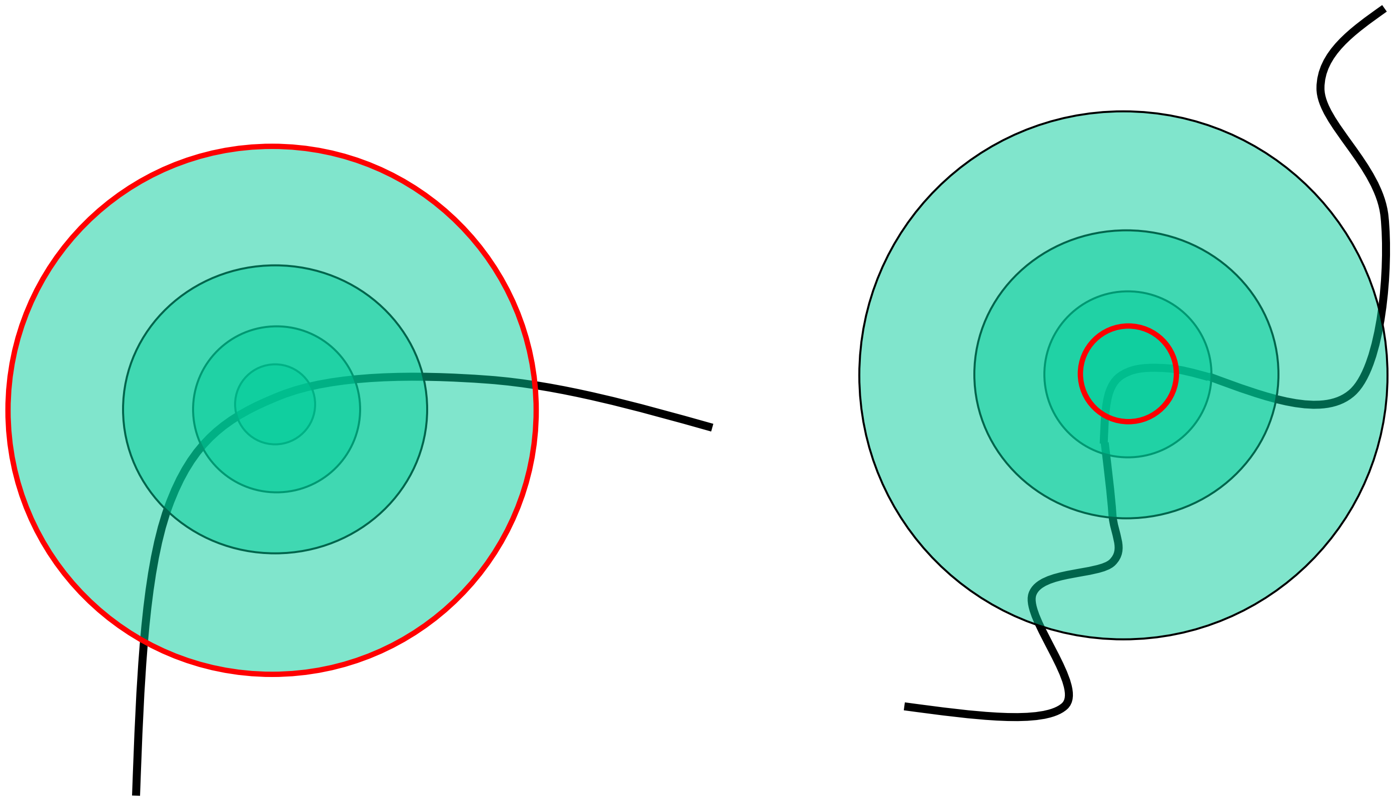
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Corresponding Regions will look the same in image space, when the appropriate scale-change is applied



Scale Invariant Detection

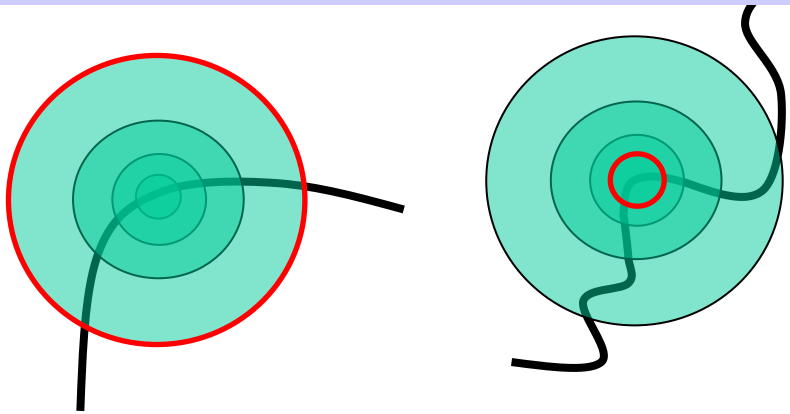
- The problem: how do we choose corresponding regions (circles) **independently** in each image?



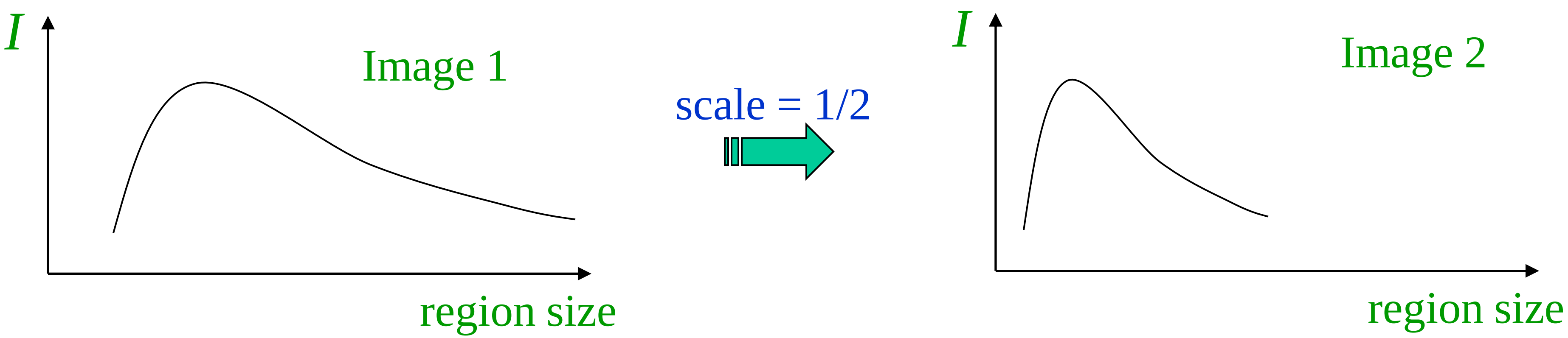
Scale Invariant Detection

- Approach: Design a function on the region (circle), which is "scale invariant" (i.e. remains constant for corresponding regions, even if they are at different scales)

Example: **average image intensity** over corresponding regions (even of different sizes) should remain constant

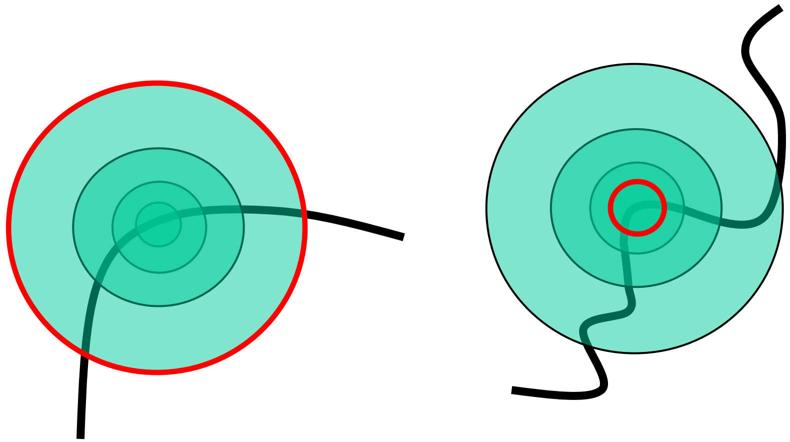


- Average intensity value enclosed in each circle, as a function of the circle-radius:

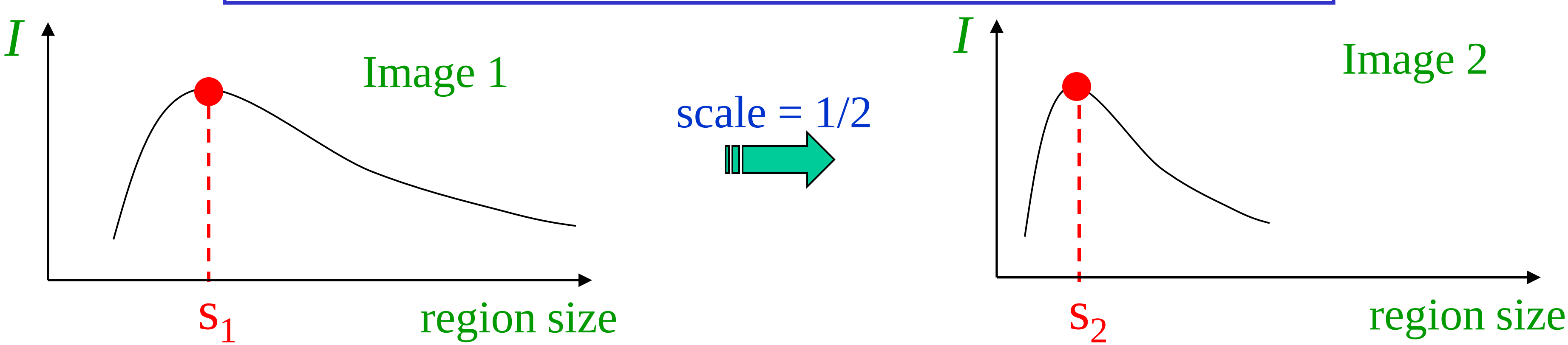


Scale Invariant Detection

Identify the local maximum in each response
⇨ These occur at corresponding region sizes

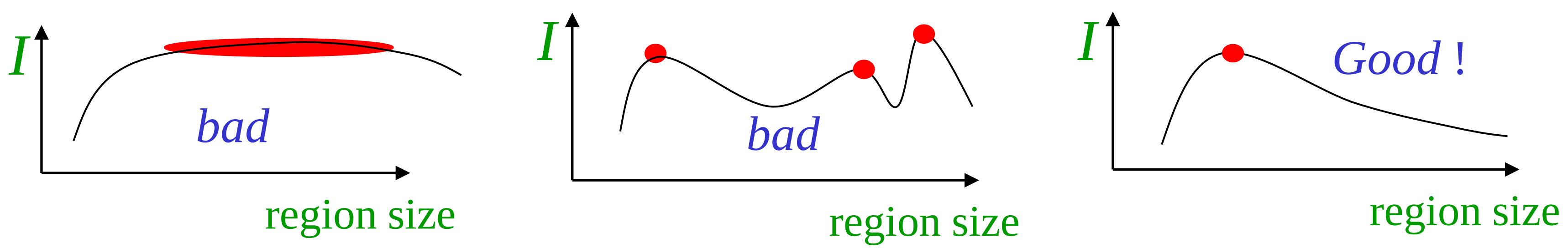


Important: this scale invariant region size is found in each image independently!



Scale Invariant Detection

- A "good" function for scale detection: has one clear, sharp peak



- Sharp, local intensity changes are good functions to monitor for identifying relative scale in usual images.

Scale Invariant Detection

- Functions for determining scale: convolve image with kernel for identifying sharp intensity discontinuities

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

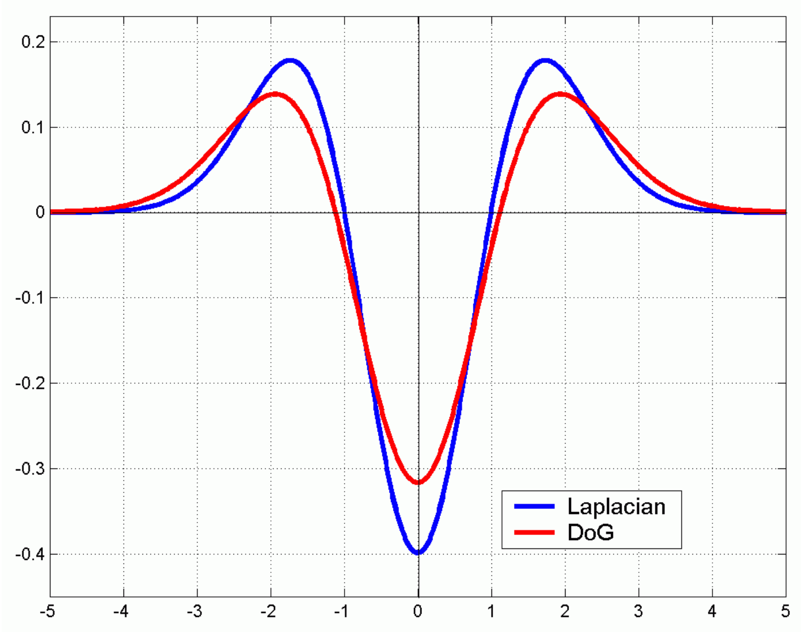
Kernels:

$$LoG = \nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$

(Laplacian of Gaussian)

$$DoG = G_{k\sigma}(x,y) - G_{\sigma}(x,y)$$

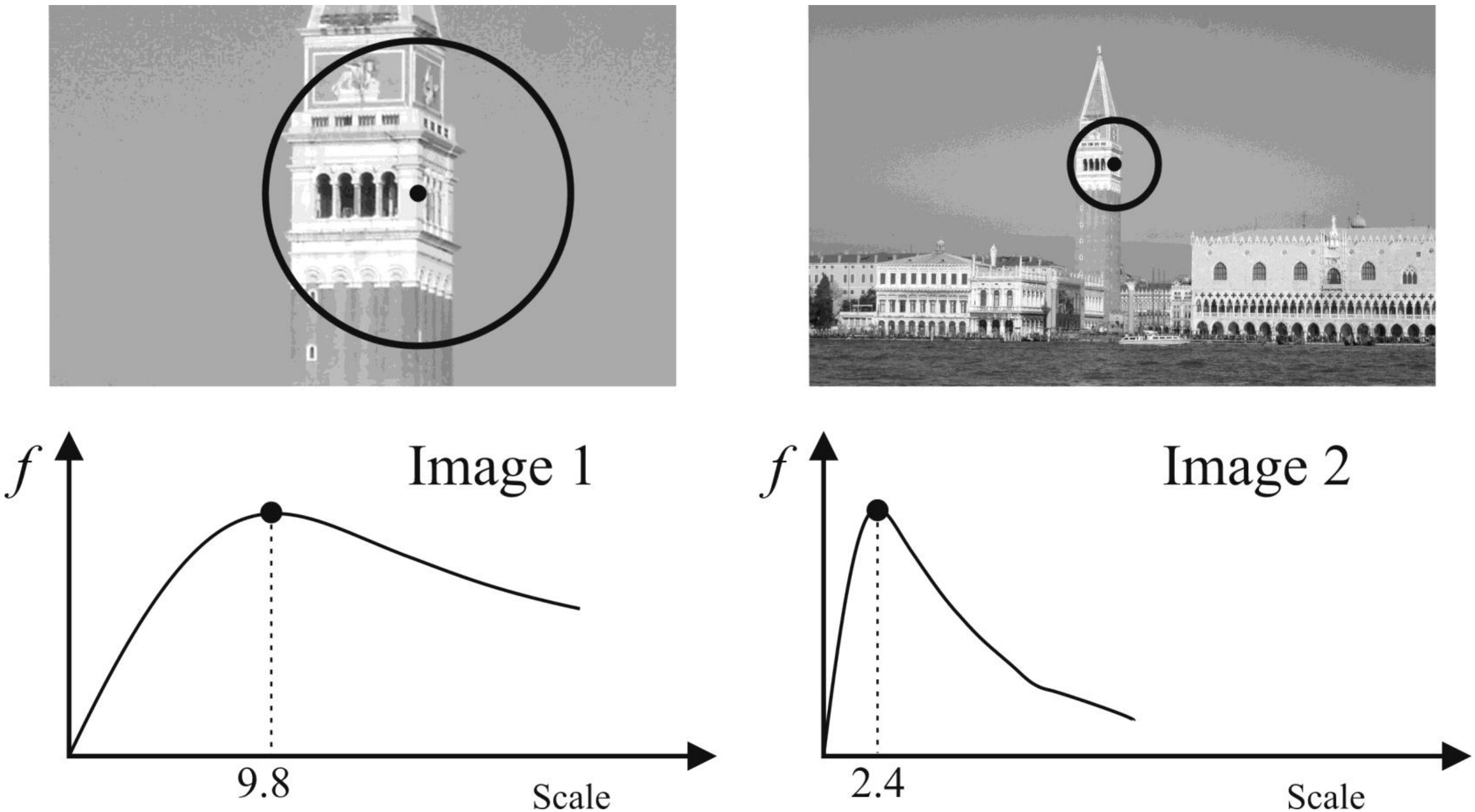
(Difference of Gaussians)



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

LoG for Scale invariant detection

- Response of LoG for corresponding regions



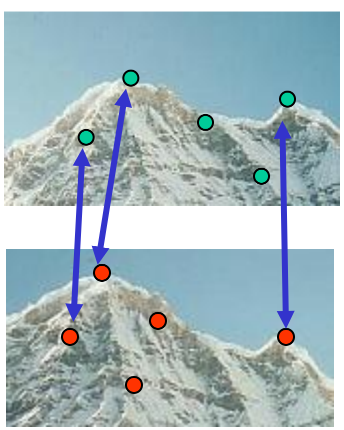
- “Harris-Laplacian” multi-scale detector

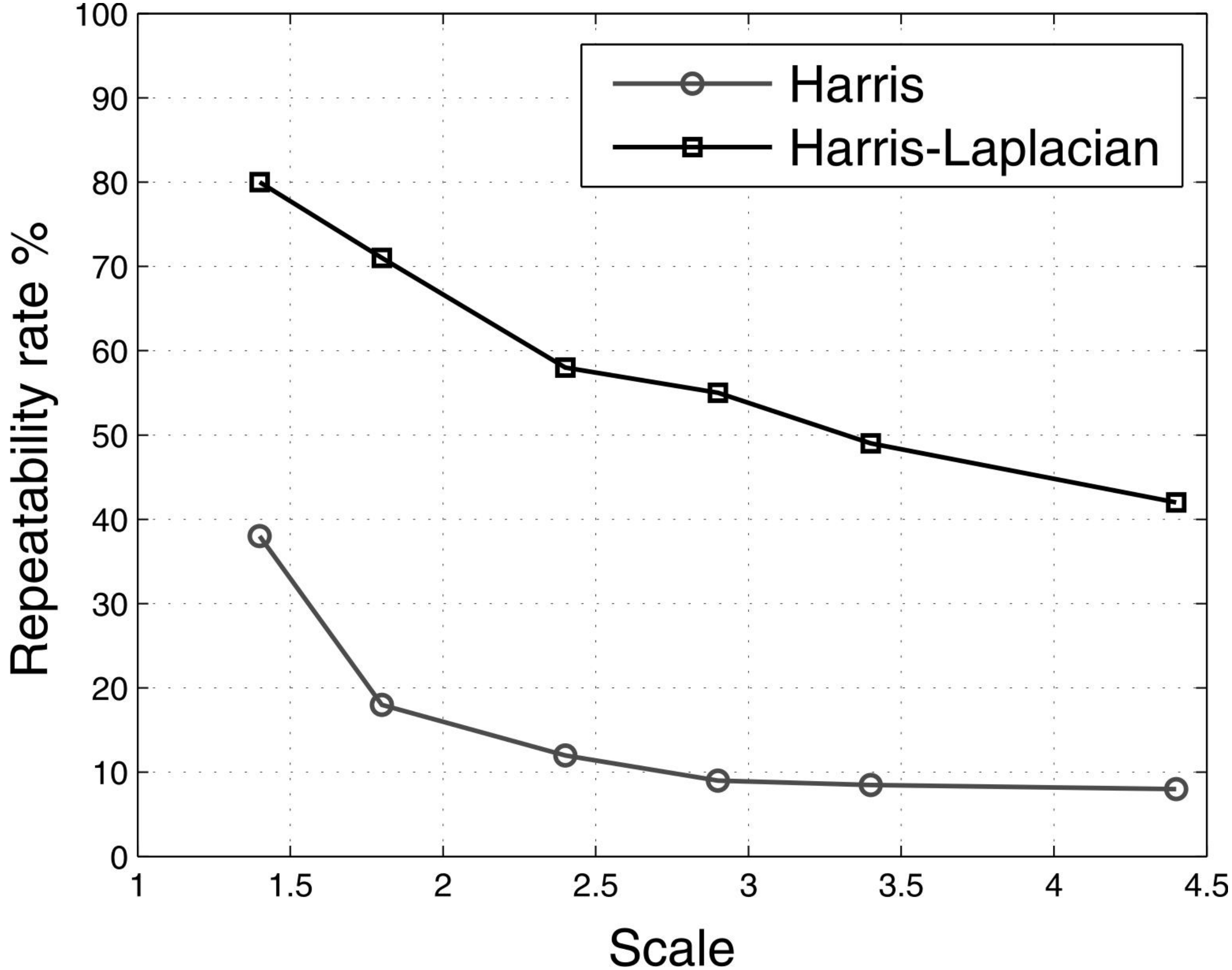
Scale Invariant Detectors

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

Repeatability rate:

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



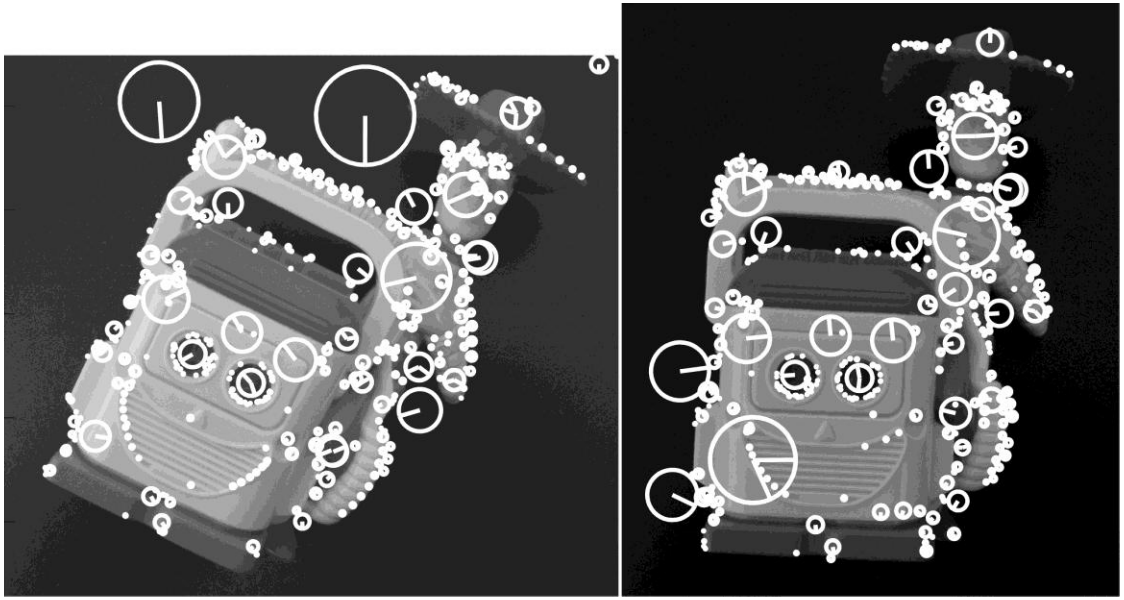


SIFT features

- Scale Invariant Feature Transform (SIFT) is an approach for detecting and describing regions of interest in an image. Developed by D. Lowe.
- SIFT features are reasonably invariant to changes in: Rotation, scaling, small changes in viewpoint, illumination
- Very powerful in capturing + describing distinctive structure, but also computationally demanding

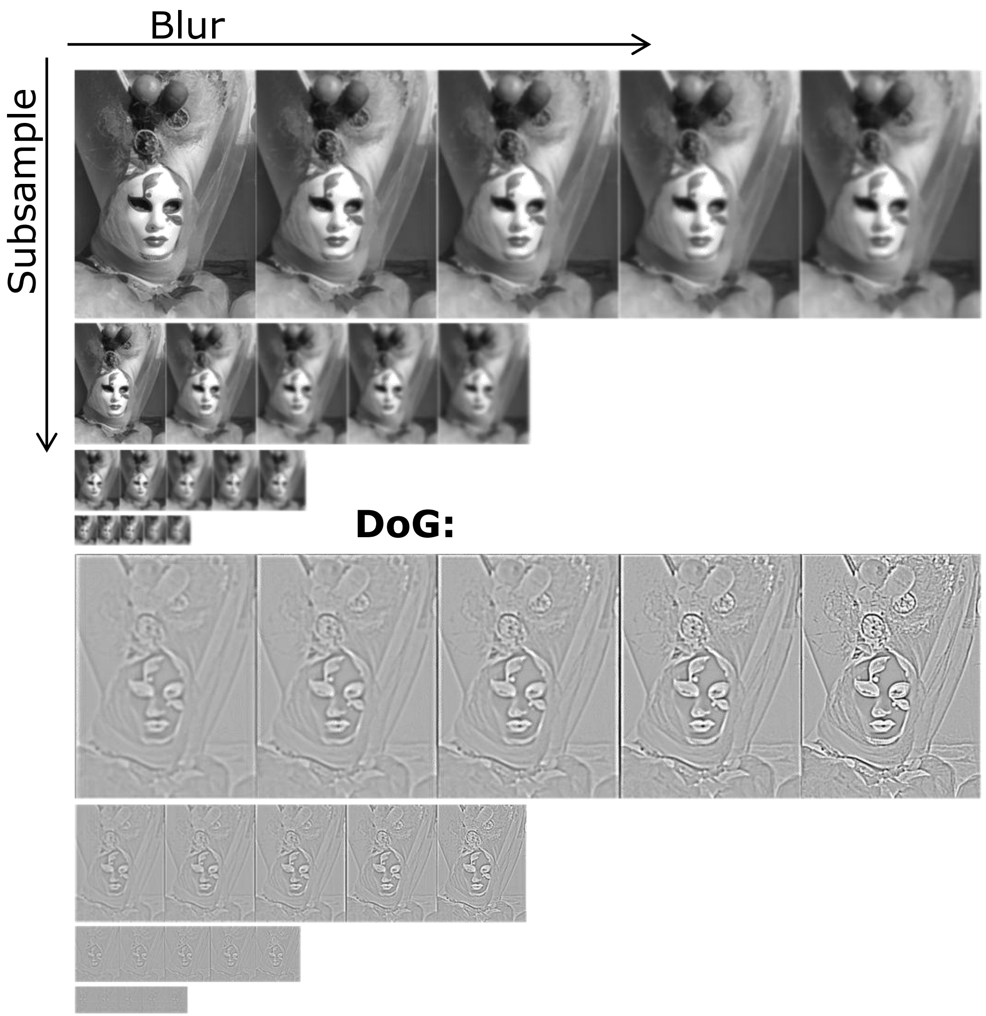
Main SIFT stages:

1. Extract keypoints + scale
2. Orientation assignment
3. Generate keypoint descriptor

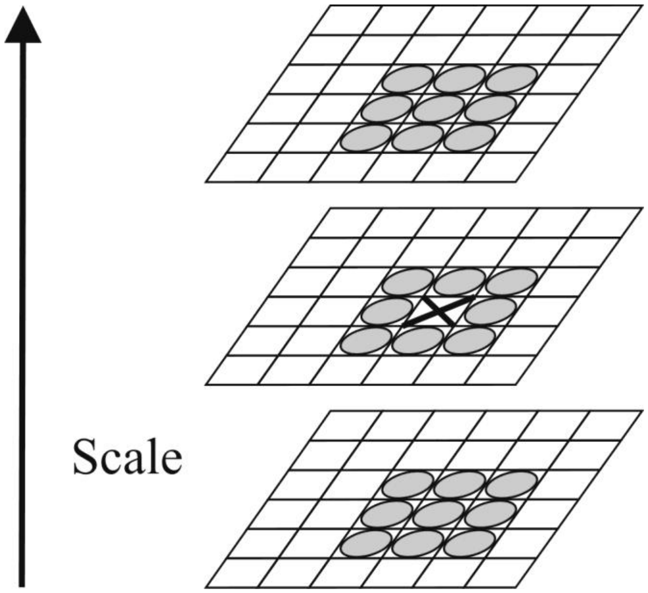


SIFT detector (keypoint location + scale)

1. Scale-space pyramid: subsample and blur original image
2. Difference of Gaussians (DoG) pyramid: subtract successive smoothed images



3. Keypoints: local extrema in the DoG pyramid



SIFT: orientation assignment

Find ‘orientation’ of keypoint to achieve **rotation invariance**

- Sample intensities around the keypoint
- Compute a histogram of orientations of intensity gradients

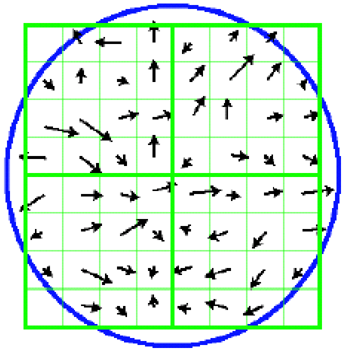
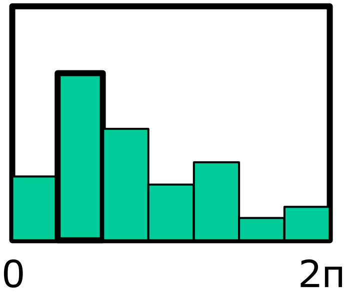
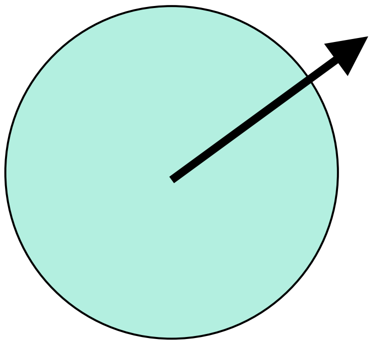


Image gradients



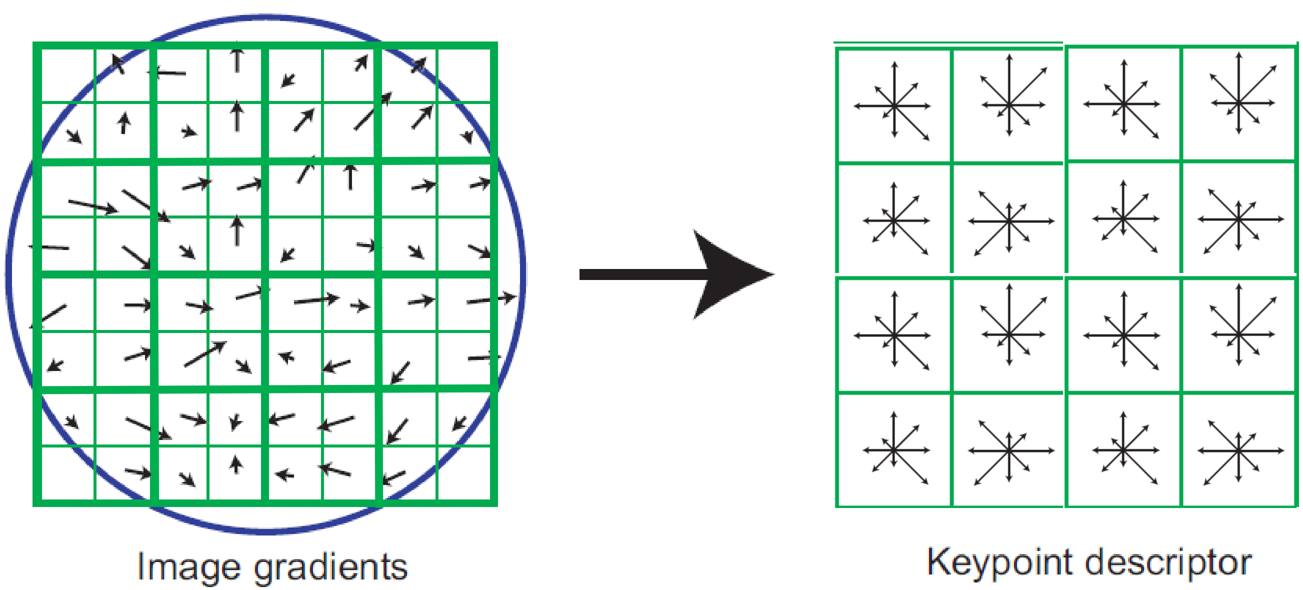
- Peaks in histogram: dominant orientations
- Keypoint orientation = histogram peak
- If there are multiple candidate peaks, construct a different keypoint for each such orientation



SIFT descriptor

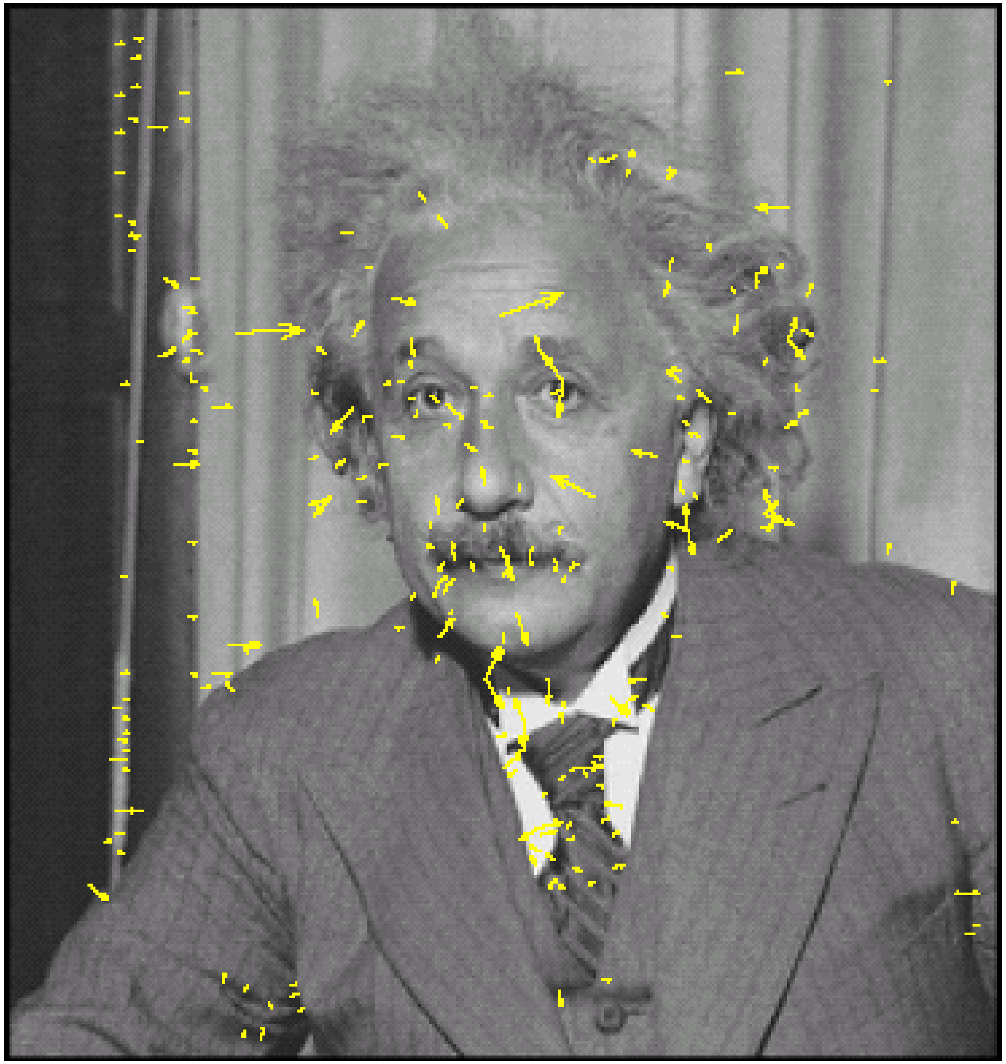
- Descriptor : “identy card” of keypoint
- Simplest descriptor: matrix of intensity values around a keypoint
- Ideally, the descriptor is highly distinctive: allows recognition of a given feature uniquely among many others!

- SIFT descriptor: 128-long vector
- Describe all gradient orientations relative to the Keypoint Orientation
- Divide keypoint neighbourhood 4×4 regions and compute orientation histograms along 8 directions
- SIFT descriptor: concatenation of all 4×4×8 (=128) values

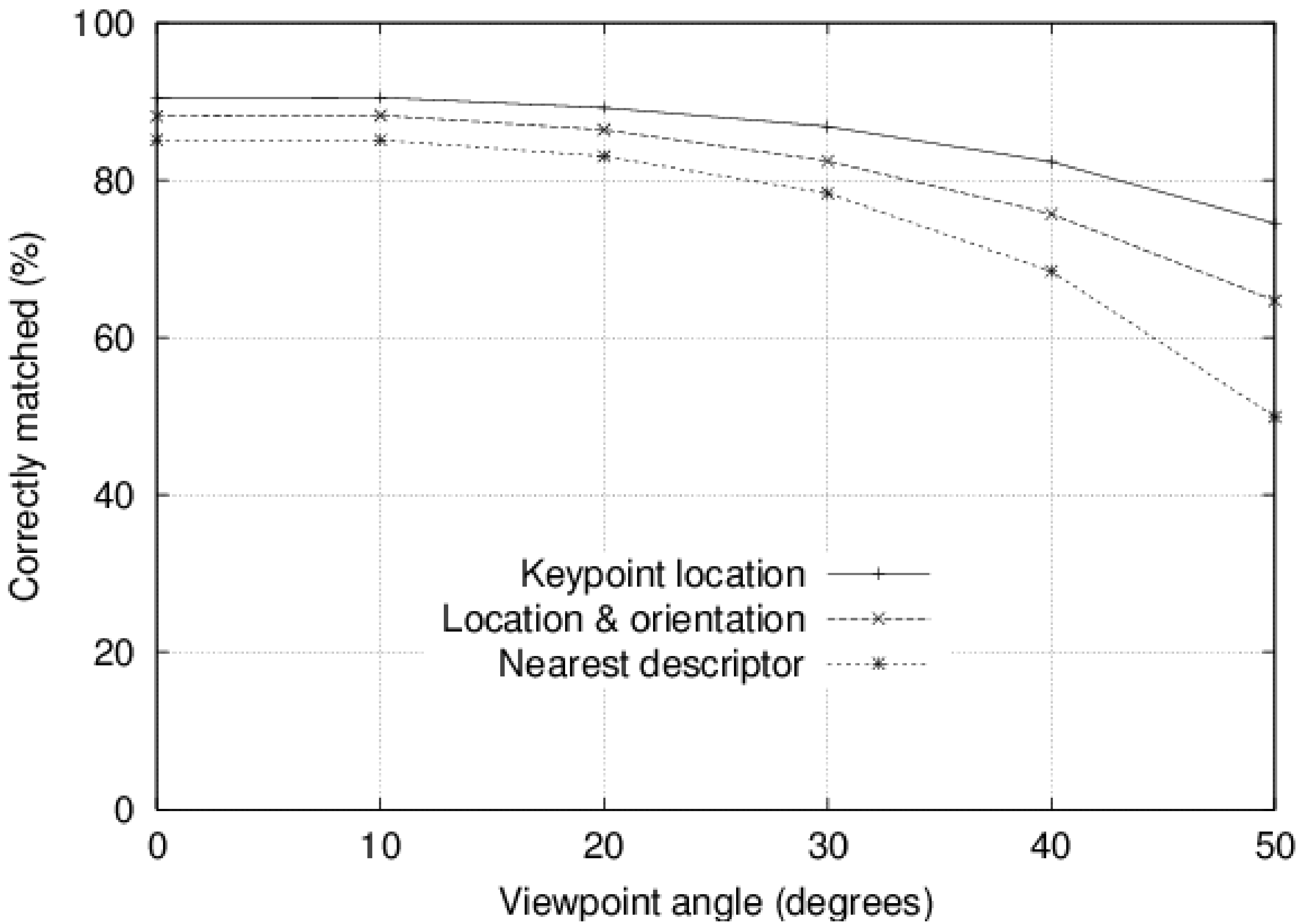


SIFT keypoints

- Final SIFT keypoints with detected orientation and scale



Feature stability to view point change

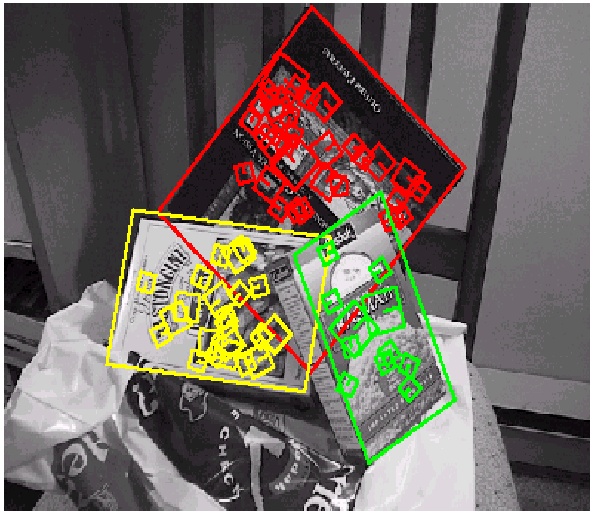


SIFT: Lowe, IJCV 2004

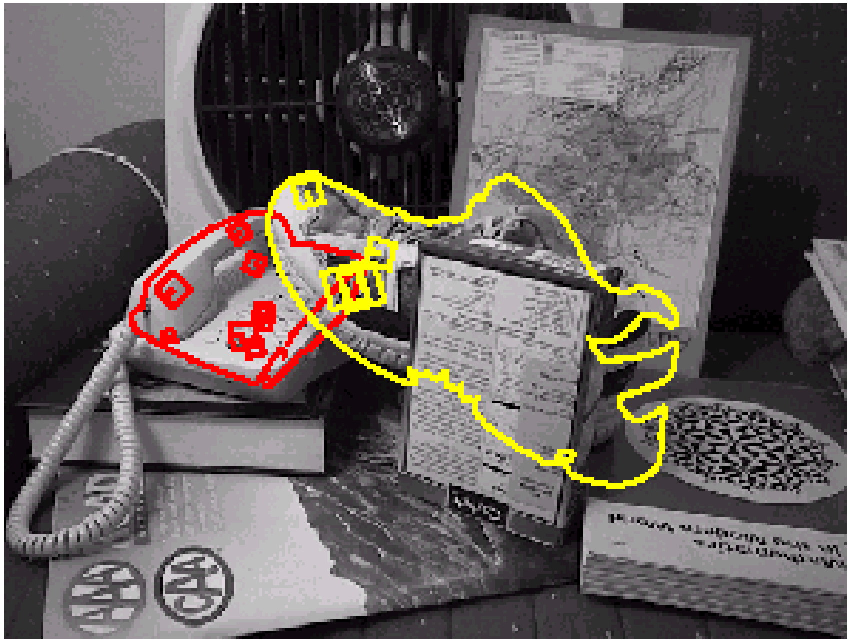
- Original SIFT paper here: 

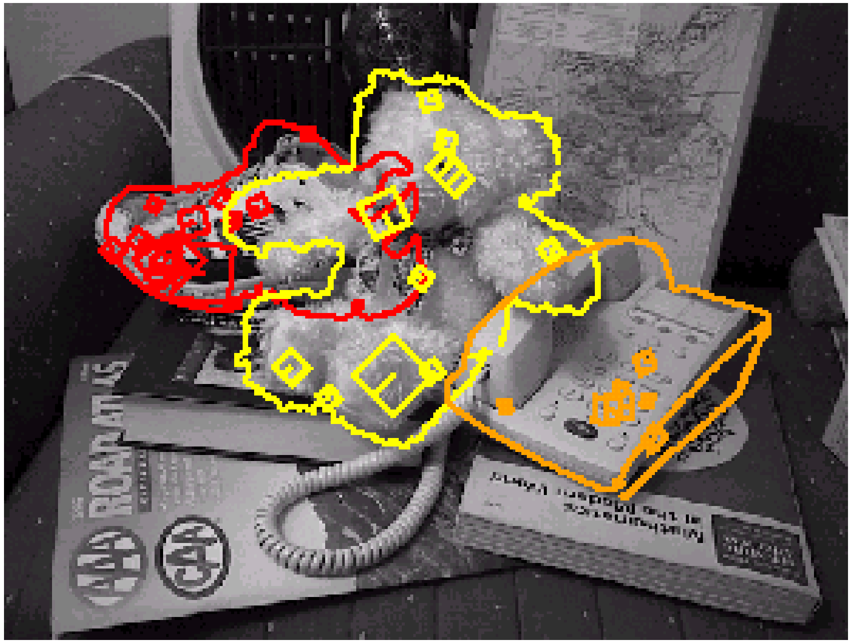
Planar recognition

- Planar surfaces can be reliably recognized at a rotation of 60° away from the camera
- Only 3 points are needed for recognition
- But objects need to possess enough texture

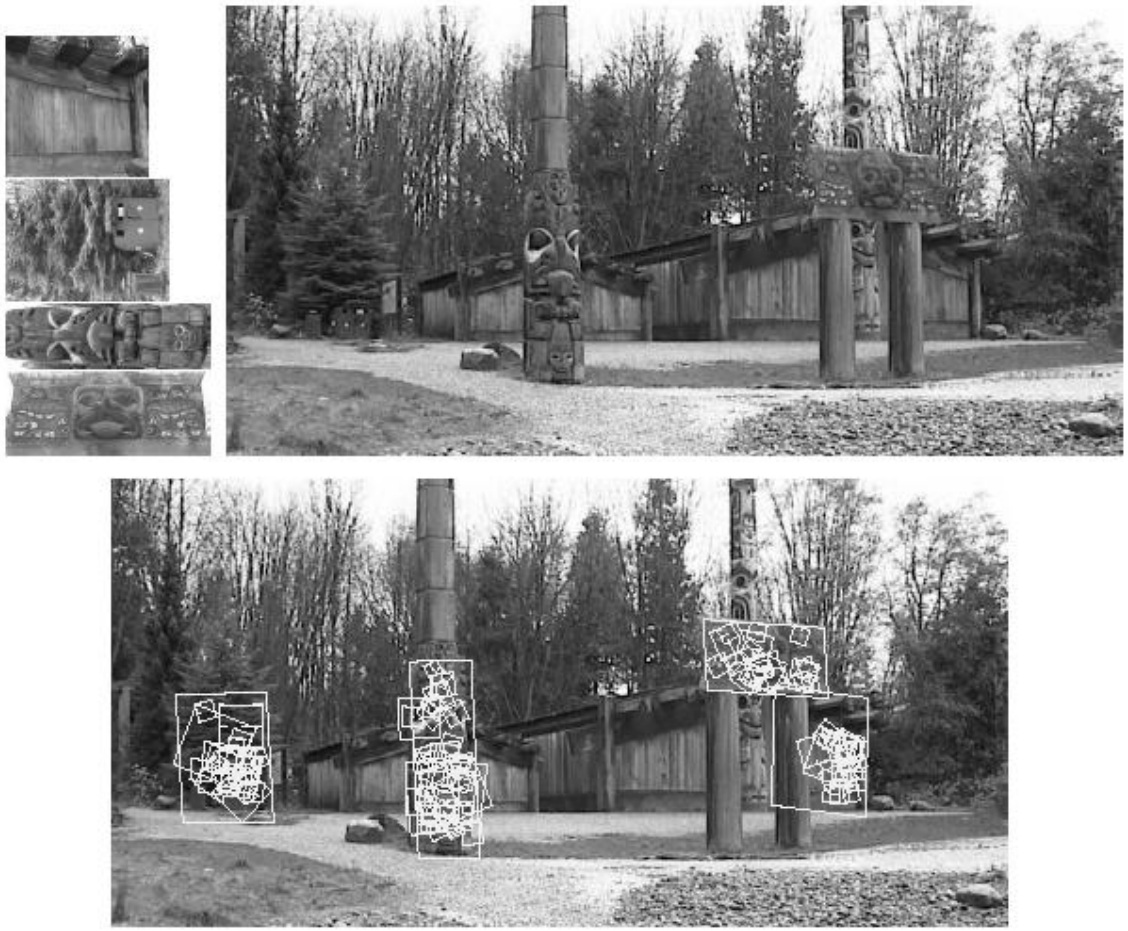


Recognition under occlusion



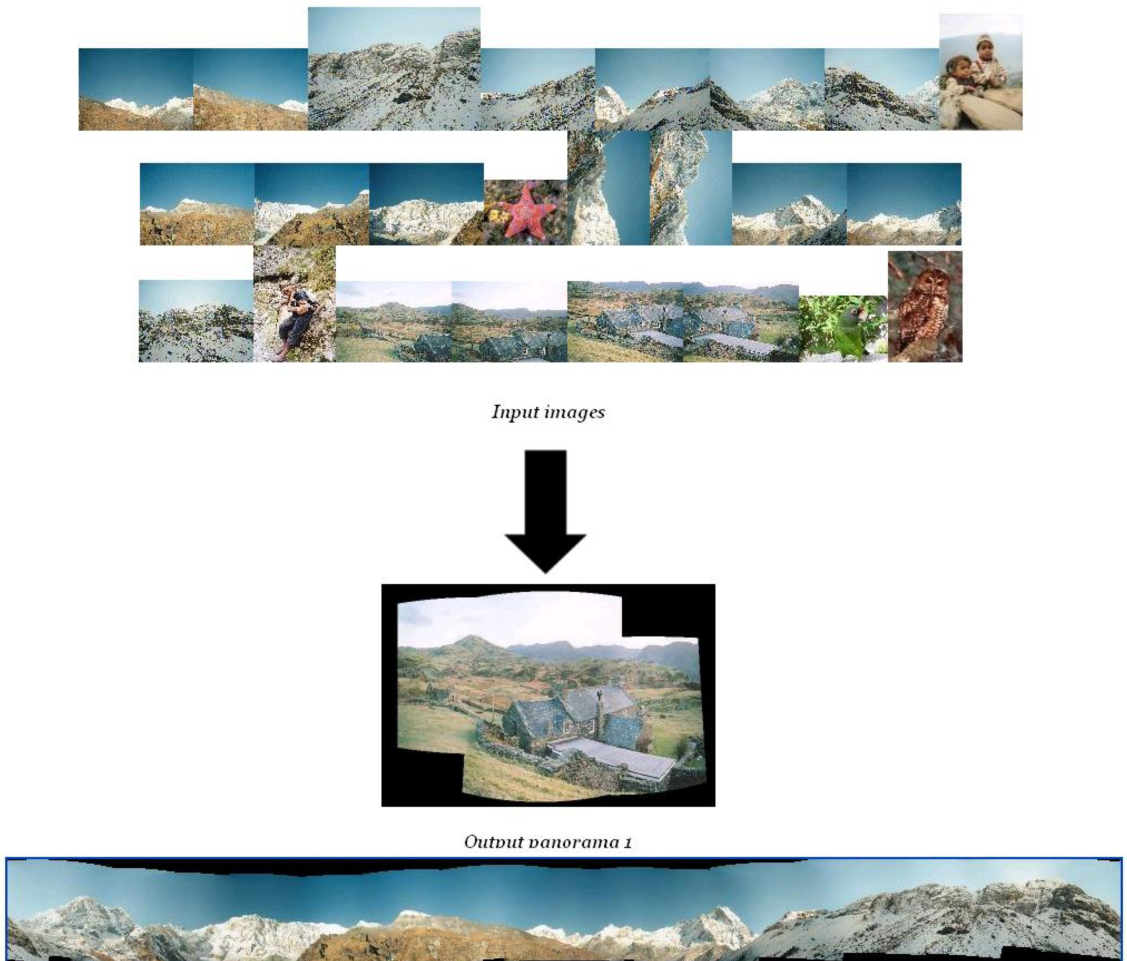


Place recognition



Multiple panoramas from an unordered image set

SIFT is used in current consumer cameras (e.g. Sony, Canon) to build panoramas from multiple shots!



Demos

- **SIFT feature detector Demo: for Matlab, Win, and Linux (freeware)**

http://www.cs.ubc.ca/~lowe/keypoints/ (demo shown in lecture)

http://www.vlfeat.org/~vedaldi/code/sift.html

- Do your own panorama with **AUTOSTITCH (freeware):**

http://www.cs.ubc.ca/~mbrown/autostitch/autostitch.html