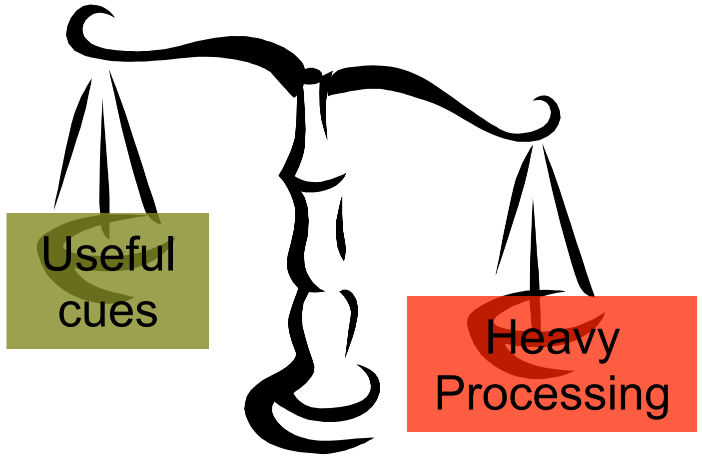
Perception III:
Fundamentals of Image Processing (incl. image features)

Autonomous Mobile Robots

Margarita Chli
Martin Rufli, Roland Siegwart

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



What is **useful**, what is **redundant**?

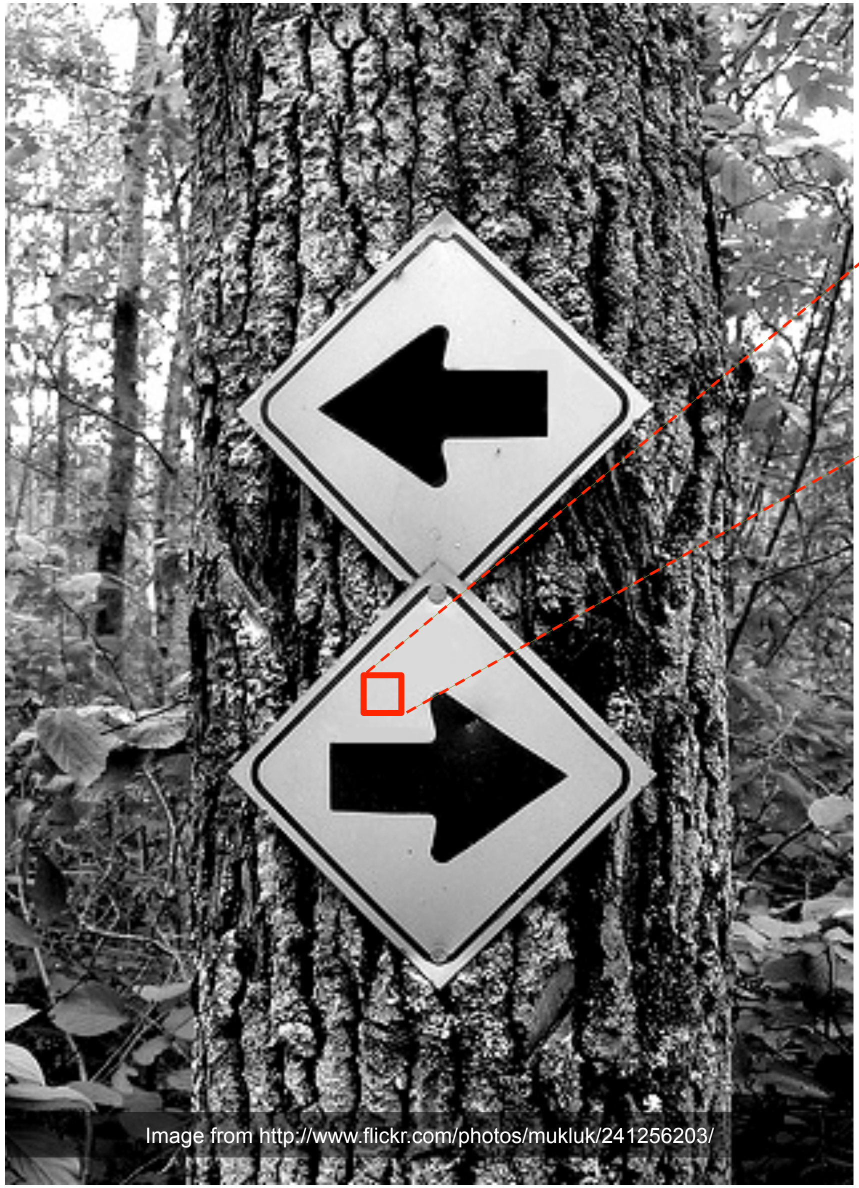
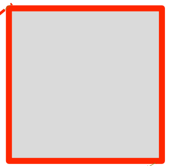
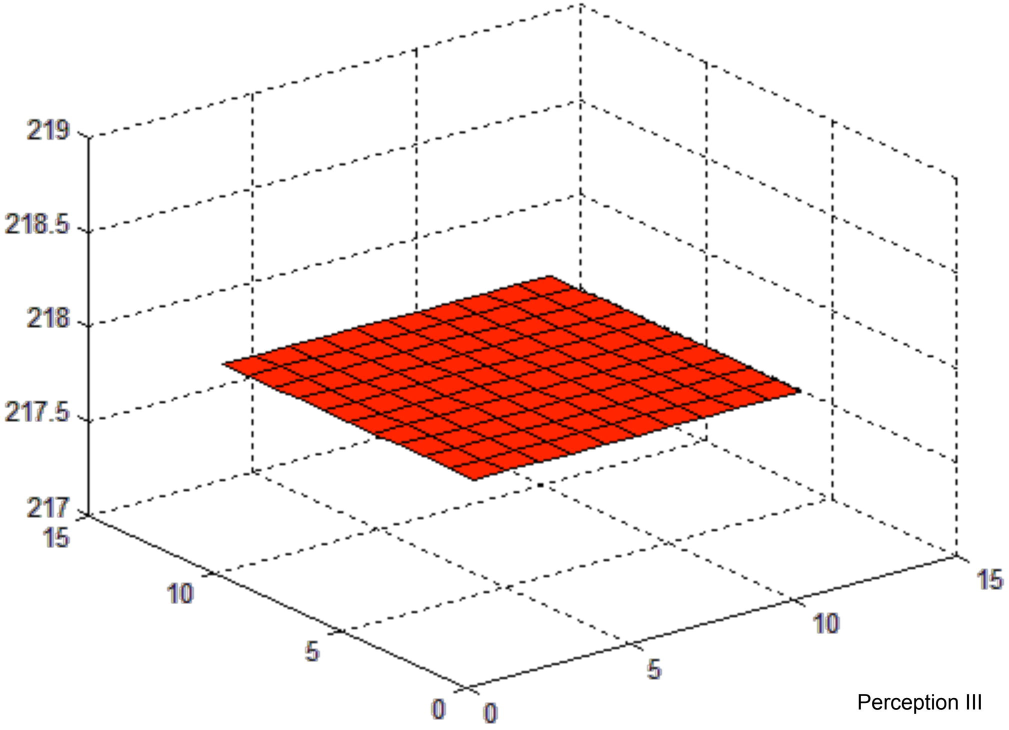


Image from http://www.flickr.com/photos/mukluk/241256203/



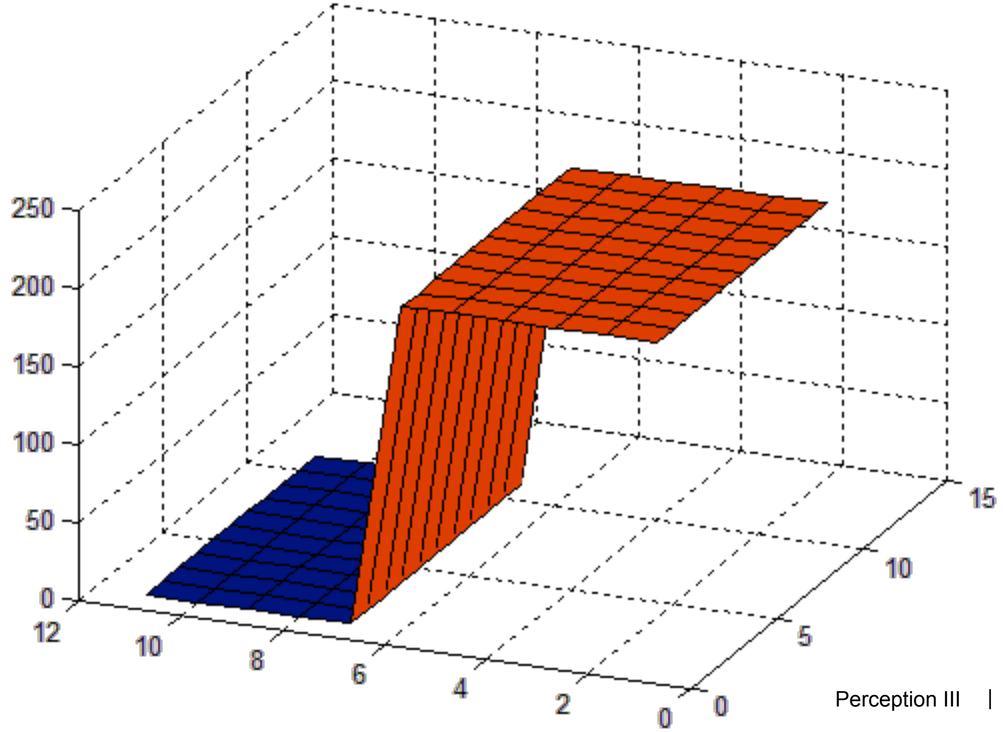
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	218
218	218	218	218	218	218	218	218	218	218	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**?

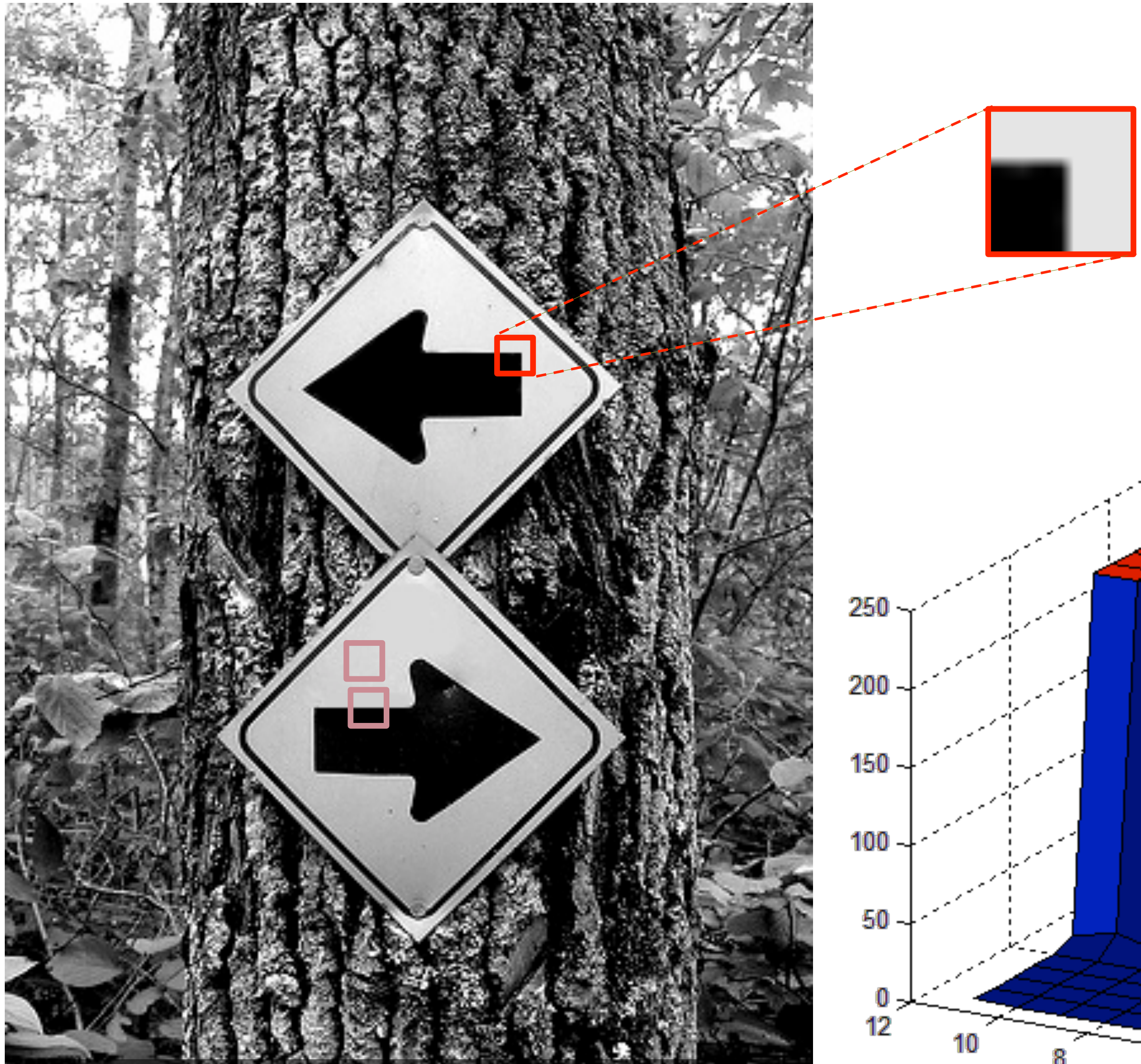
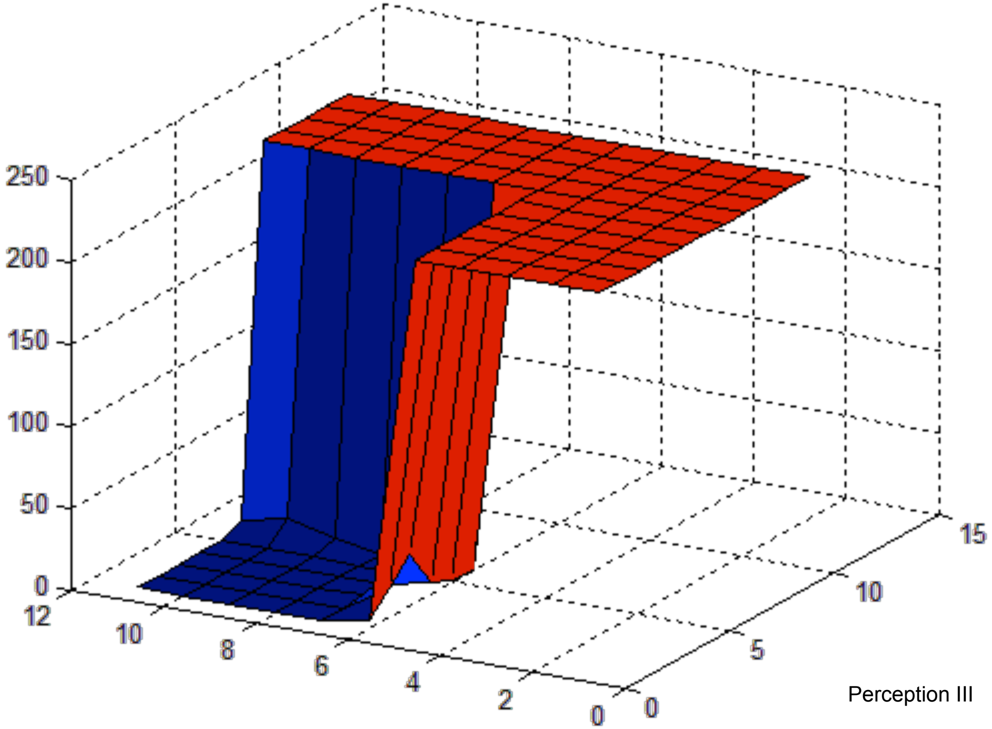


Image from http://www.flickr.com/photos/mukluk/241256203/

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 Outline

- Sections **4.3 – 4.5** of the book
- Image filtering
  - Correlation
  - Convolution
- Edge / Corner extraction
- Point Features
  - Harris corners
  - SIFT features
  - + some more recent image features from the state of the art

Optional Reading:

- Harris Corner Detector: *C. Harris and M. Stephens. "A combined corner and edge detector." Alvey vision conference, 1988.* [paper]
- Shi-Tomasi features*: J. Shi and C. Tomasi. "Good features to track." IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 1994.* [paper]
- SIFT features: *D. G. Lowe. "Distinctive image features from scale-invariant keypoints."International Journal of Computer Vision (IJCV), 2004.* [paper][demo code]
- FAST corner detector: *E. Rosten, R. Porter, and T. Drummond. "Faster and better: A machine learning approach to corner detection."IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2010.* [paper]
- BRIEF descriptor: *M. Calonder, V. Lepetit, C. Strecha, P. Fua. "Brief: Binary robust independent elementary features." European Conference on Computer Vision (ECCV), 2010.* [paper]
- BRISK features: *S. Leutenegger, M. Chli, and R. Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints." International Conference on Computer Vision (ICCV), 2011.*[paper]
- Open source implementation of some of these methods (and others) in OpenCV.

Image filtering

- **filtering:** accept / reject certain components
- <u>example:</u> a low-pass filter allows low frequencies ⇨ blurring (smoothing) effect on an image – used to reduce image noise
- Smoothing can be achieved not only with **frequency filters**, but also with **spatial filters**.

Low-pass filtering:
retains low-frequency components (smoothing)

High-pass filtering:
retains high-frequency components (edge detection)

Image filtering | 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

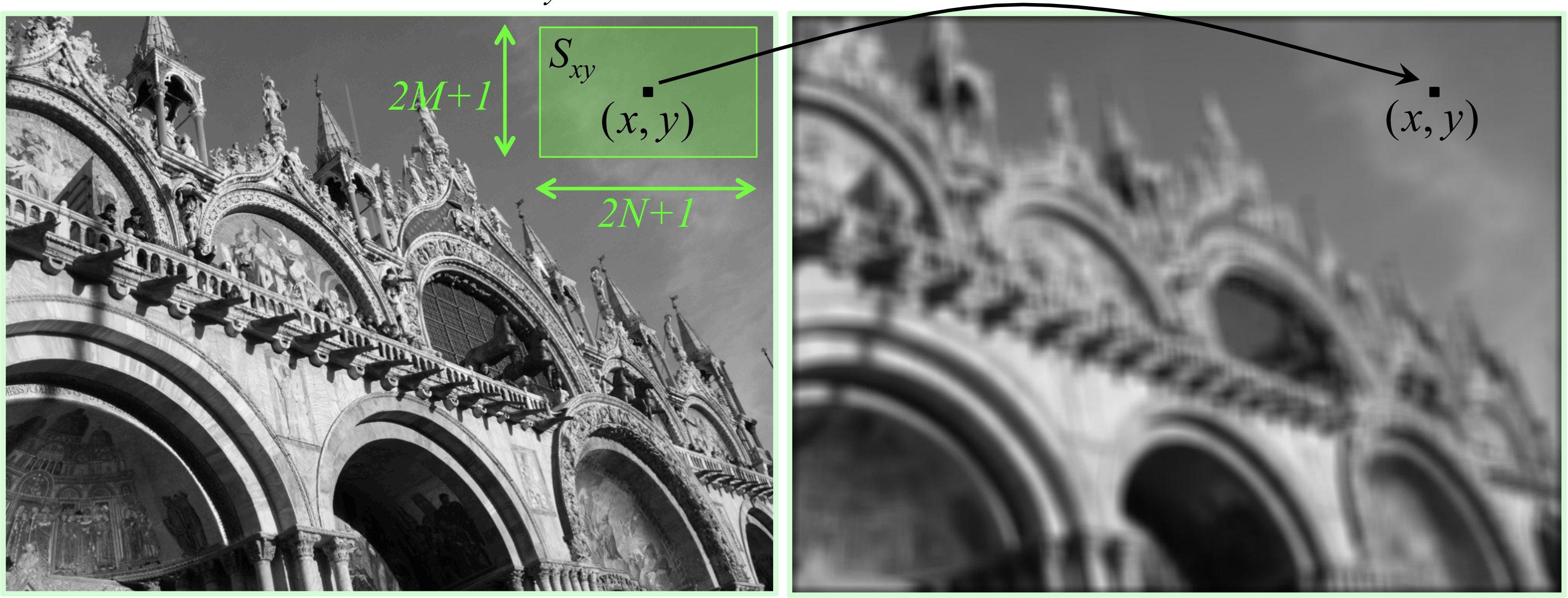


Image I Filtered Image $J = F(I)$

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

Image filtering | 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
- Why filter?
  - Noise reduction, image enhancement, feature extraction, …
- Basic & very useful filtering operations:
  - Correlation
  - Convolution
- Brief study of these filters in the simplest case of 1D images (i.e. a row of pixels) & their extension to 2D

Image filtering | correlation

- An averaging filter

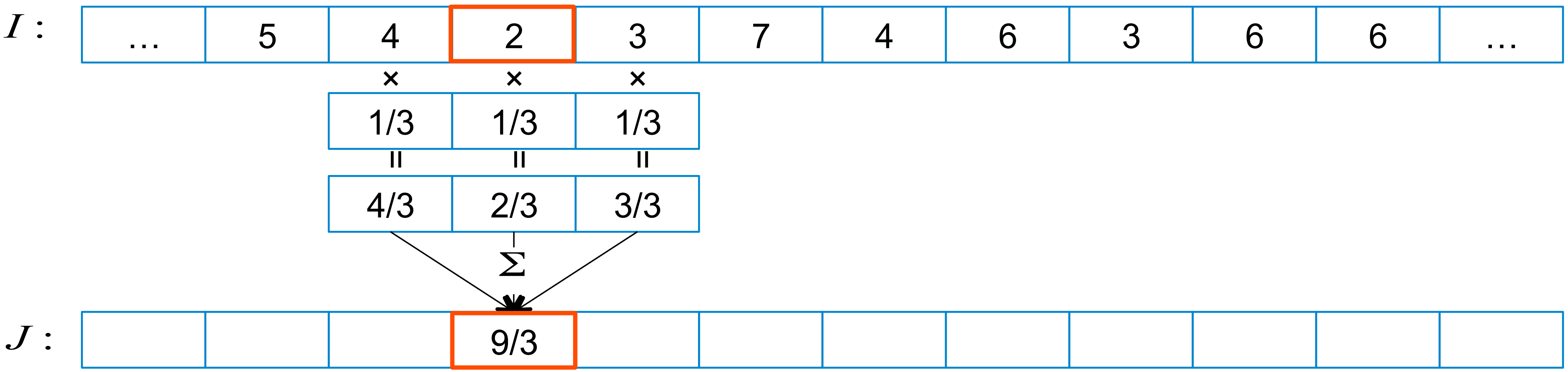
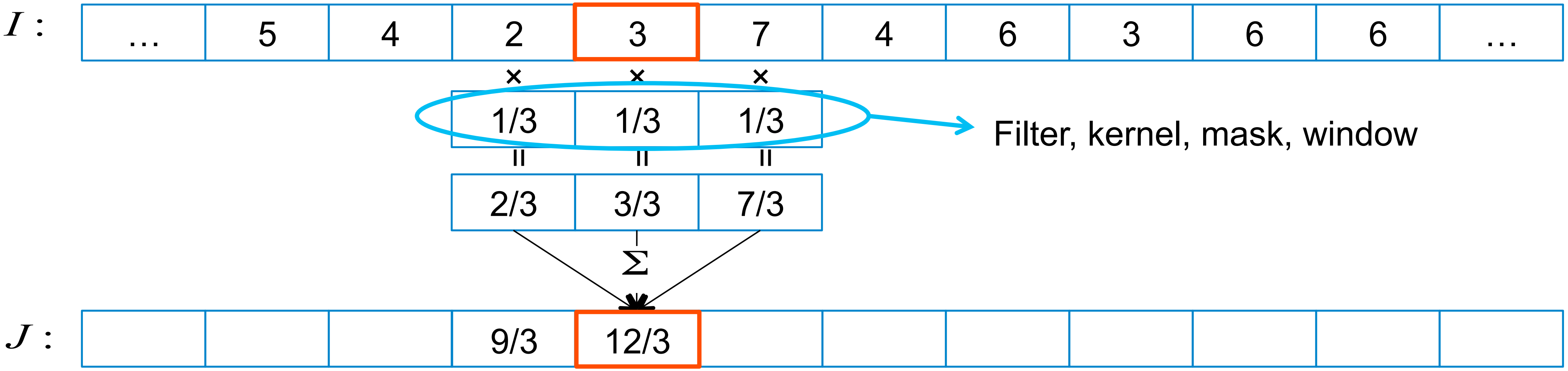


Image filtering | correlation

- An averaging filter

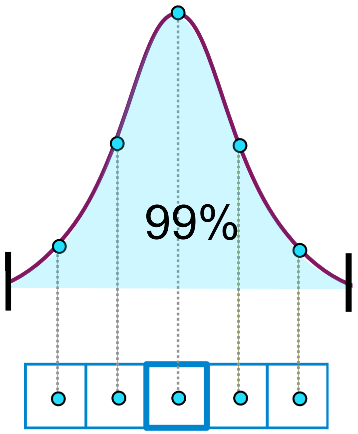


- Formally, Correlation is $J(x) = F \circ I(x) = \sum_{i \in [-N,N]} F(i) I(x+i)$

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

Image filtering | constructing filter from a continuous fn

- Common practice for image smoothing:
  use a Gaussian



Normalize filter so that values always add up to 1

$$g(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$\mu = 0$

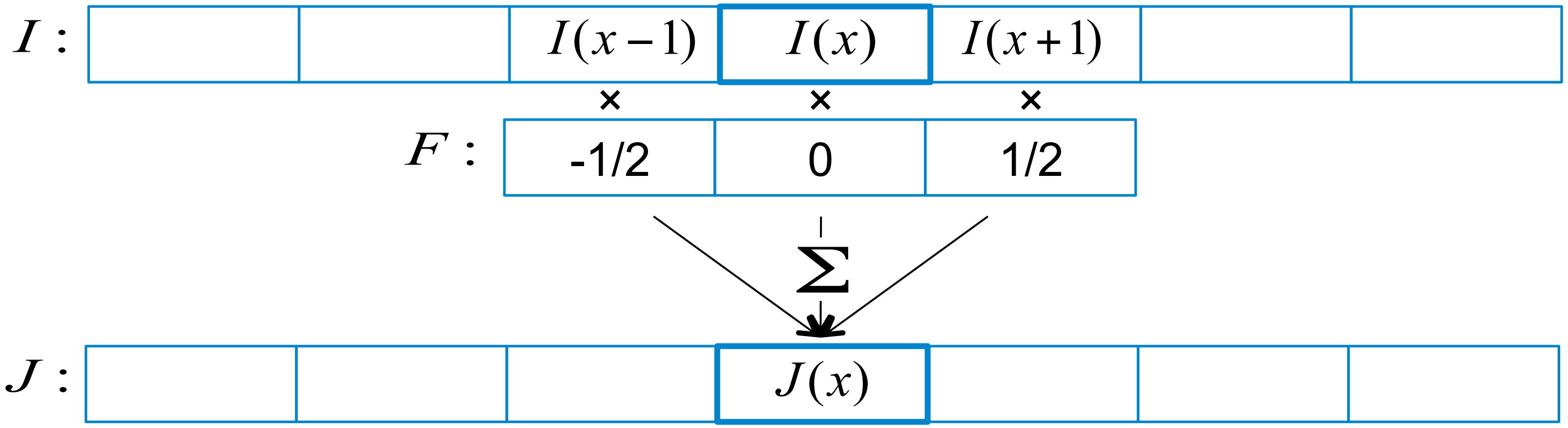
σ : controls the amount of smoothing

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

Image filtering | taking derivatives with correlation

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

- Approximate a derivative operator:



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

Image filtering | matching using correlation

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

Correspondence Search | the problem

- **goal**: identify image regions / patches in the left & right images, corresponding to the same scene structure
  - Typical **similarity measures**: Normalized Cross-Correlation (NCC) , Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), …
  - **Exhaustive** image search can be computationally very expensive! *Can we search for correspondences more efficiently?*

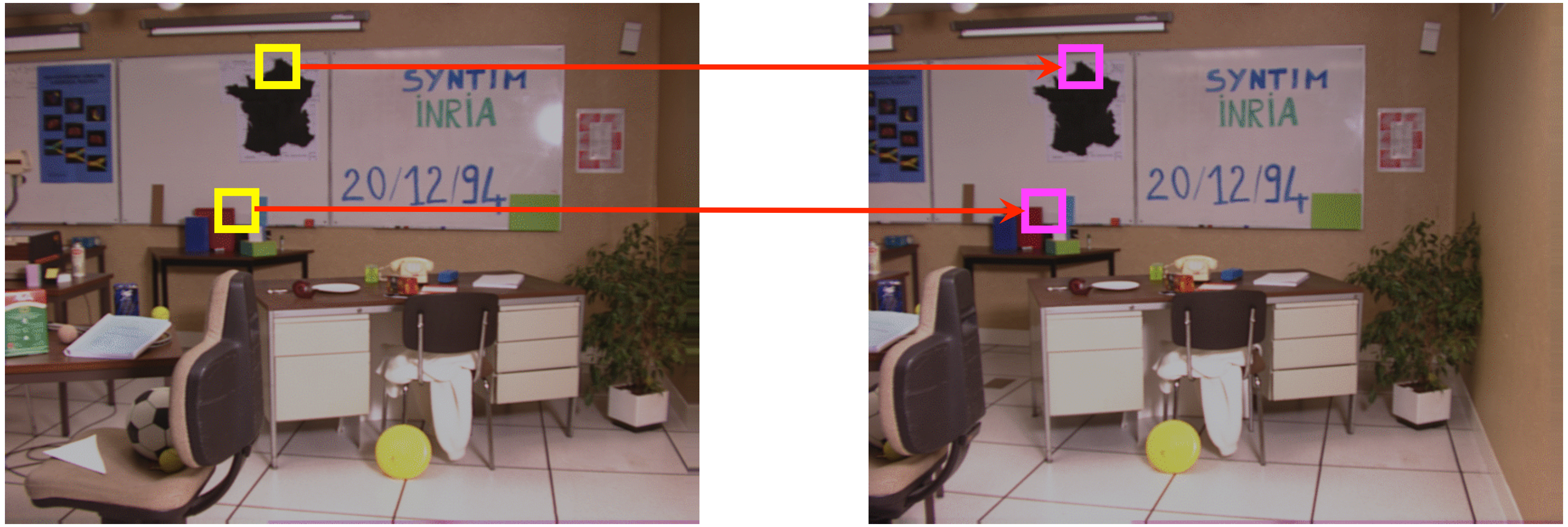


Image filtering | matching using correlation

- 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**) – minimize

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

$J:$

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

Image filtering | 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**) – minimize

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

Correlation

J :

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

- Similarity measure: Correlation? – maximize

Image filtering | 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**) – minimize

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

Correlation

J :

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

- Similarity measure: Correlation? – maximize

J :

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

Image filtering | 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**) – maximize

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

$J:$

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

Image filtering | ZNCC: Zero-mean NCC

- 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: Zero-mean Normalized Cross Correlation (**ZNCC**) – maximize

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

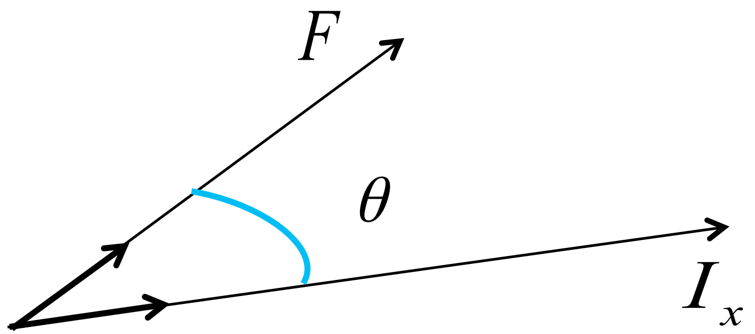
, where

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

Image filtering | correlation as a dot product

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

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



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

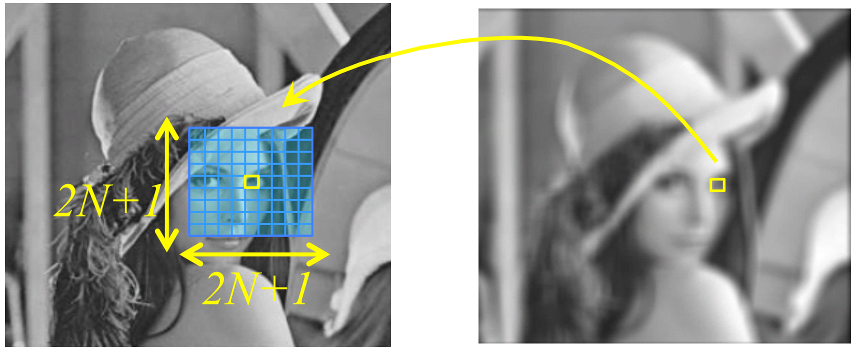
Image filtering | correlation in 2D

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

- Example:
  Constant averaging filter

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

This example was generated with a 21x21 mask



- 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 $= (2N+1)^2 - 1$

Image filtering | correlation in 2D

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

- Example:
  Constant averaging filter

$$F = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

"separable" filter

- 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 $= 2(2N+1)$
    no. additions per pixel $= 4N$

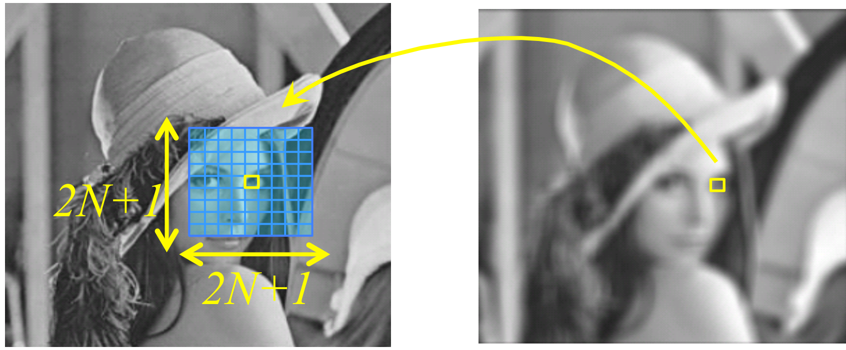


Image filtering | correlation in 2D

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

- Example:
  Constant averaging filter

$$F = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

"separable" filter

- 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 $= 2(2N+1)$
    no. additions per pixel $= 4N$
  - 2 × 1D Correlation ⇨ no. multiplications per pixel $= 1$
    (with const. factor) no. additions per pixel $= 4N$

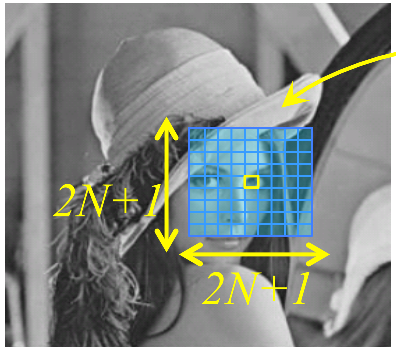


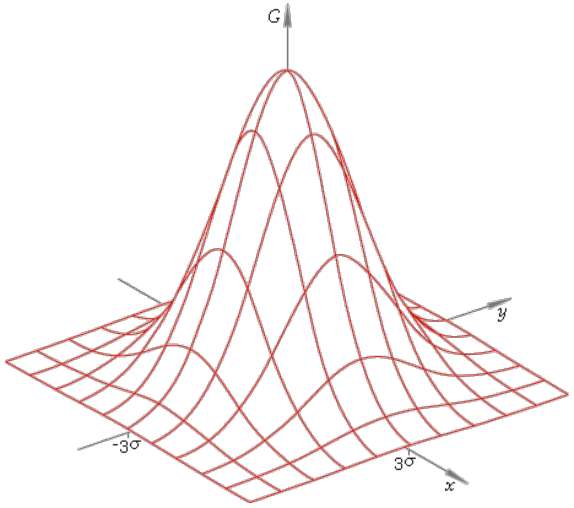
Image filtering | 2D gaussian smoothing

- 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:

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



- Another separable filter

Image filtering | convolution

VISION FOR ROBOTICS LAB

- Convolution is ***equivalent*** to Correlation with a flipped filter before correlating
- CONVOLUTION: $J(x) = F * I(x) = \sum_{i \in [-N,N]} F(i) I(x-i)$
- CORRELATION: $J(x) = F \circ I(x) = \sum_{i \in [-N,N]} F(i) I(x+i)$
- Likewise, in 2D we flip the filter both horizontally & vertically

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

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

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

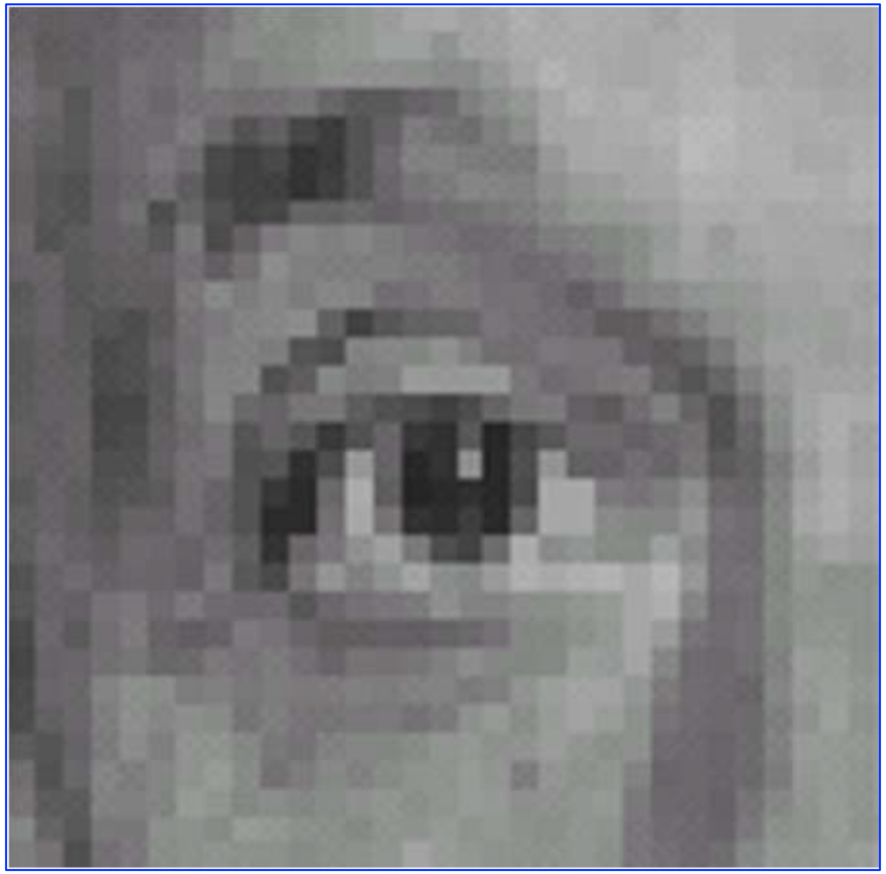
- Very useful!
- Example: smooth an image & take its derivative ⇨ convolve the Derivative filter with the Gaussian Filter & convolve the resulting filter with the Image

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

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

Then, $F * I(x) = F' \circ I(x)$

Image filtering | examples



original image

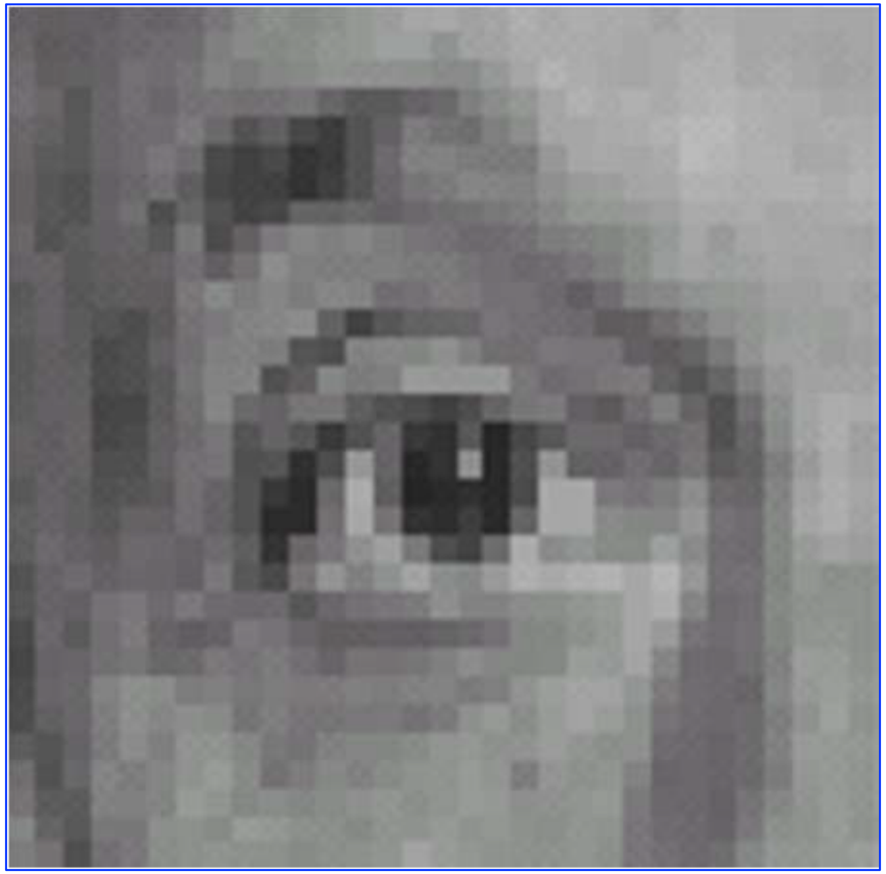
*

0	0	0
0	1	0
0	0	0

=

filtered (no change)

Image filtering | examples

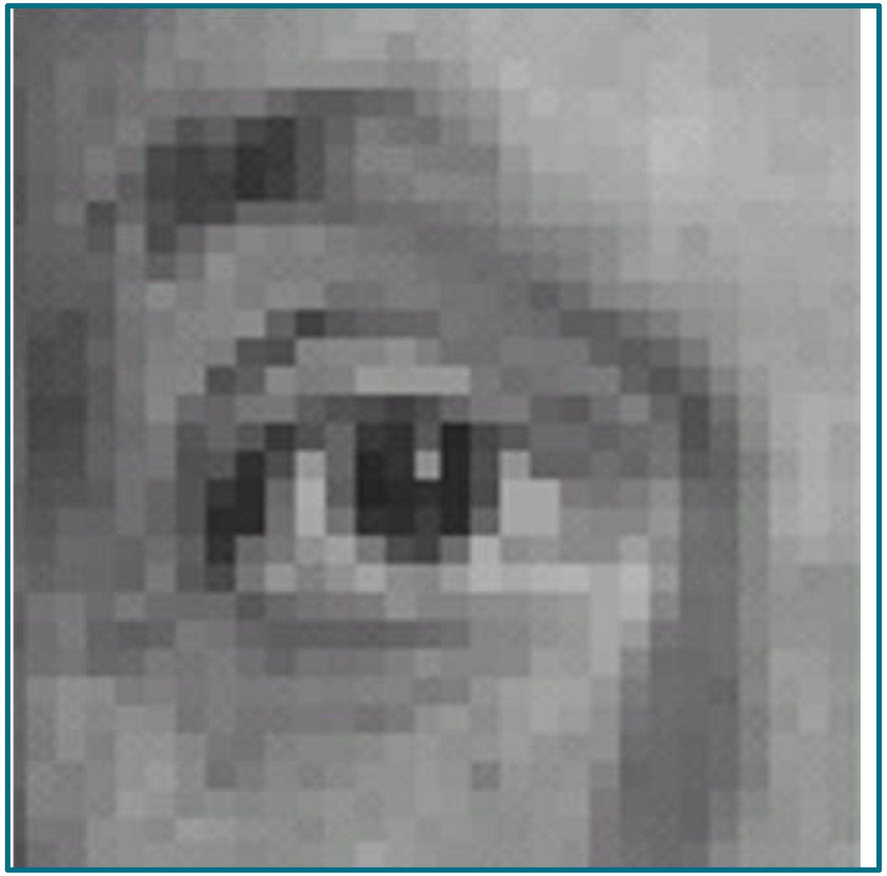


original image

*

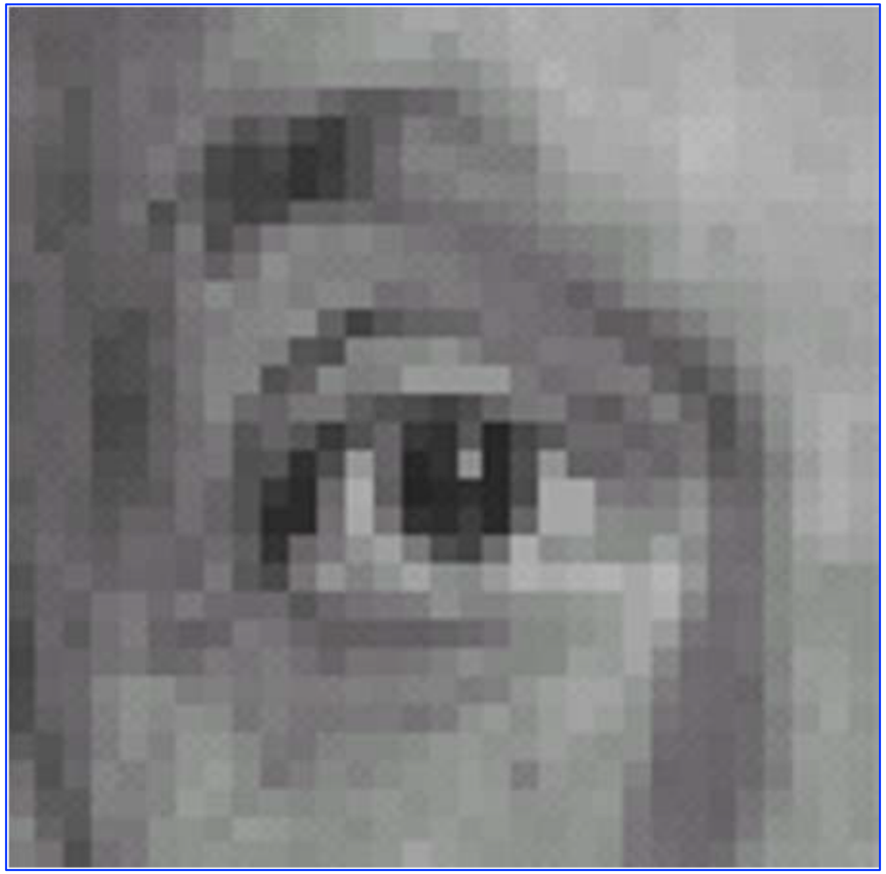
0	0	0
0	0	1
0	0	0

=



filtered (shifted left by 1 pixel)

Image filtering | examples

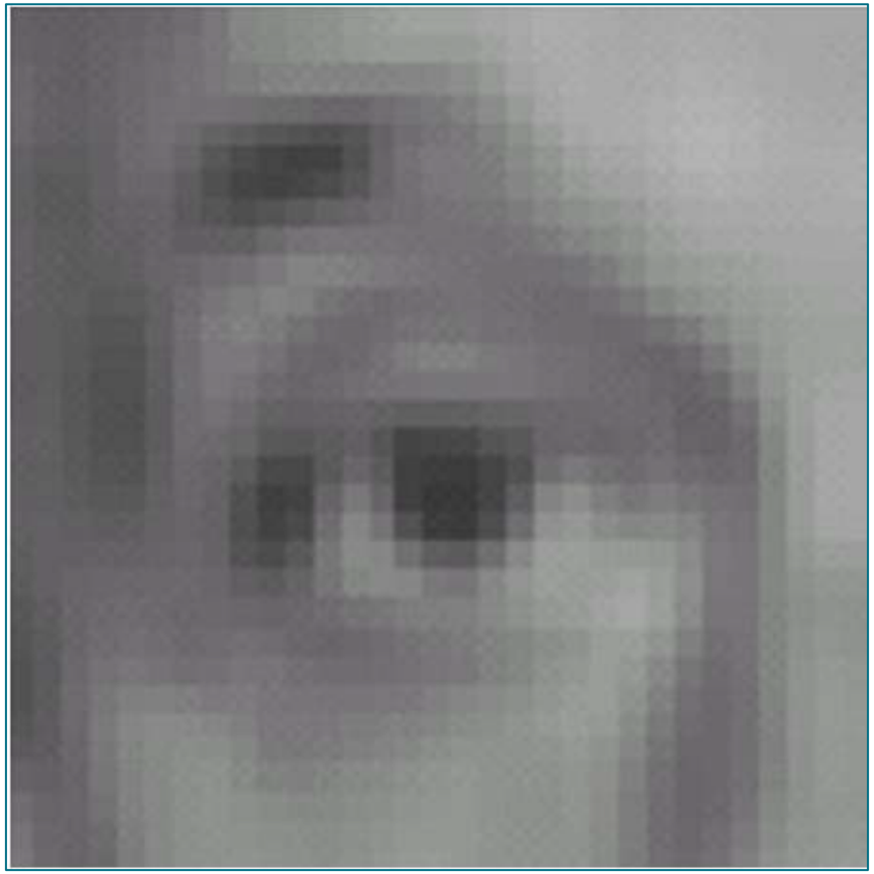


original image

*

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=



filtered (blurred with a box filter)

Image filtering | examples

- What does blurring take away?

original image

\-

smoothed (5x5)

=

detail

Image filtering | examples

- Let's add it back:

original image

+ a

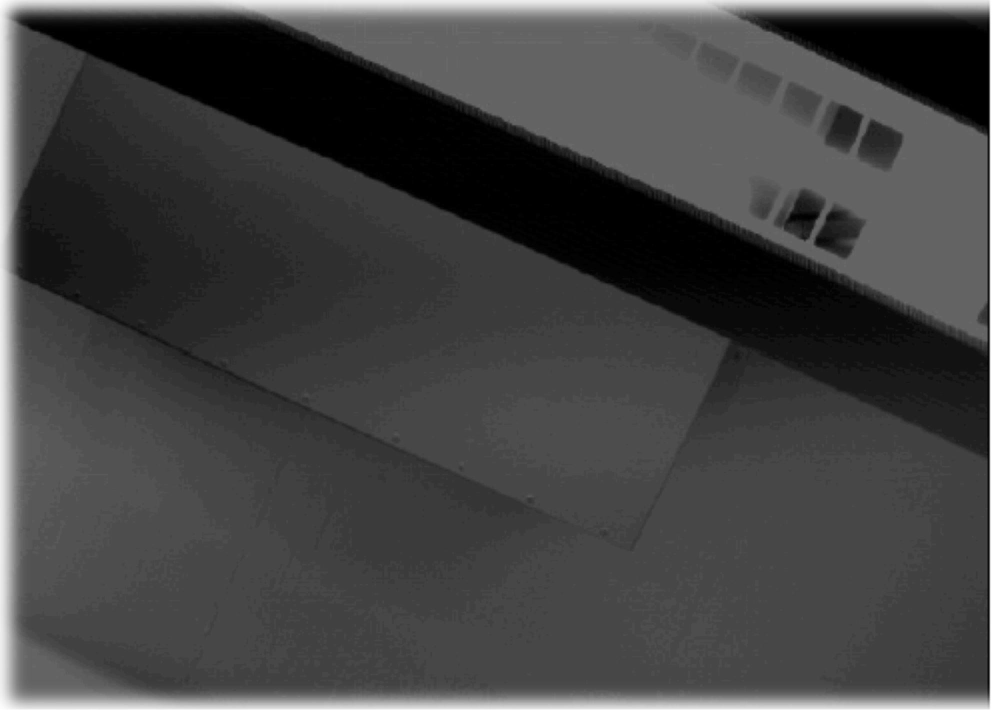
detail

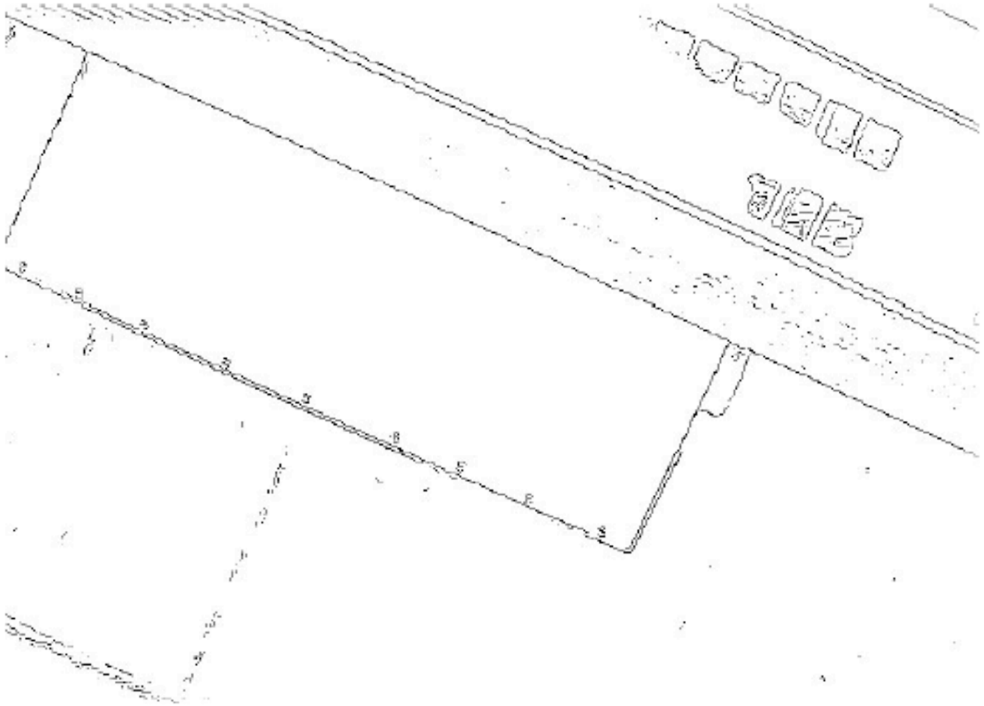
=

sharpened

Edge detection

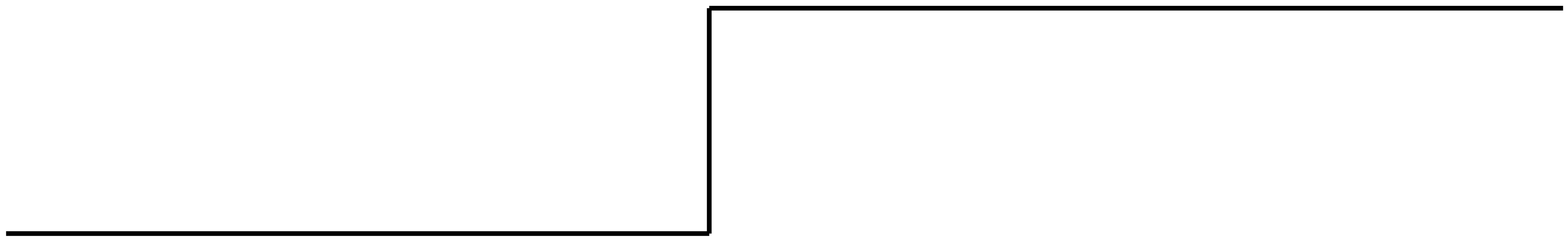
- Ultimate goal of edge detection: an idealized line drawing.
- Edge contours in the image correspond to important scene contours.





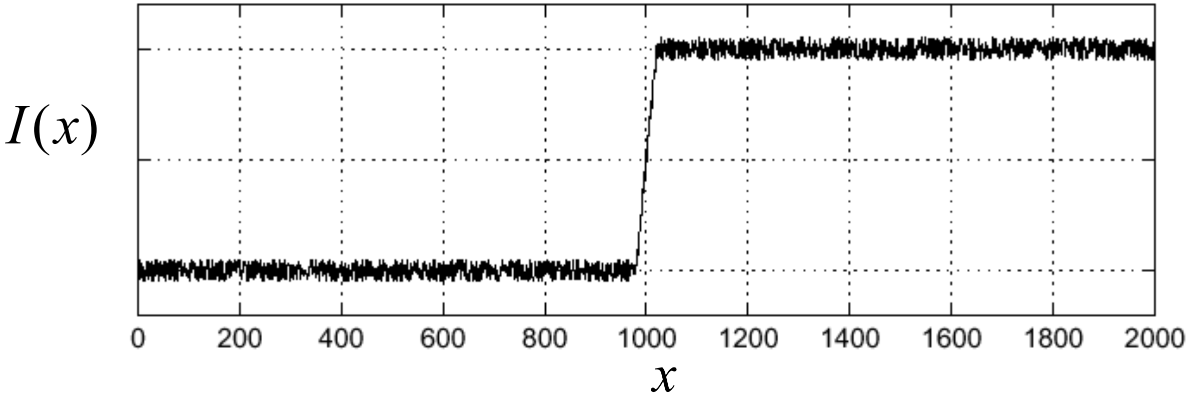
Edge detection | edge = intensity discontinuity in 1 direction

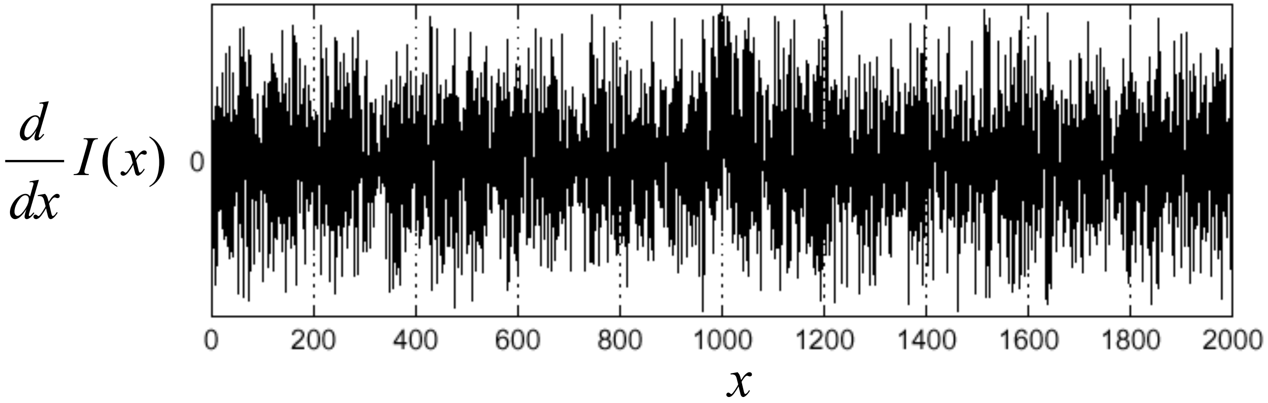
- Edges correspond to sharp changes of intensity
- **How to detect an edge?**
  - Change is measured by 1st order derivative in 1D
  - Big intensity change ⇨ magnitude of derivative is large
  - Or 2nd order derivative is zero.



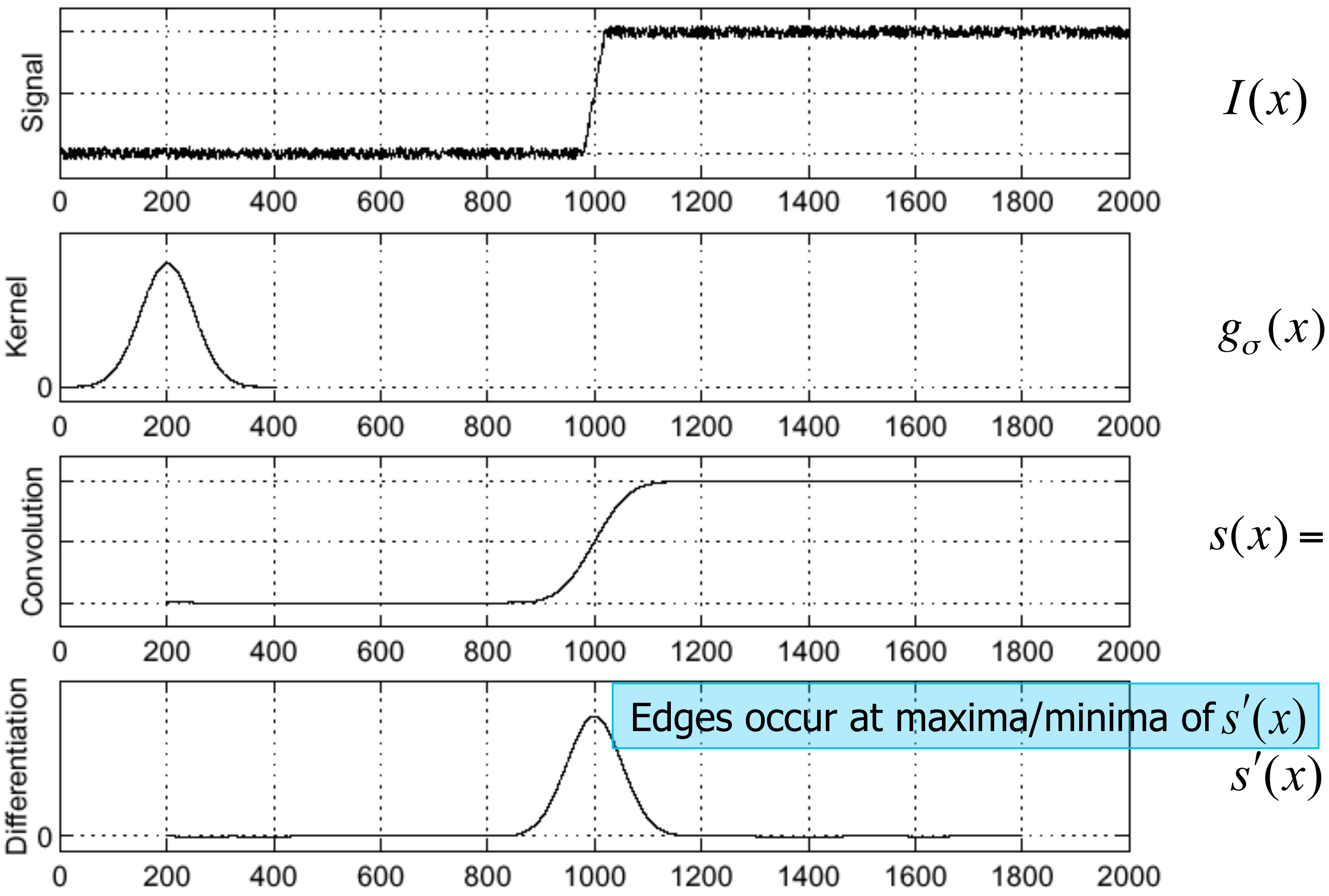
1D edge detection |

- Consider a single row or column of the image, where image intensity shows an obvious change





1D edge detection | solution: smooth first



$$I(x)$$

$$g_\sigma(x)$$

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

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

- Where is the edge?

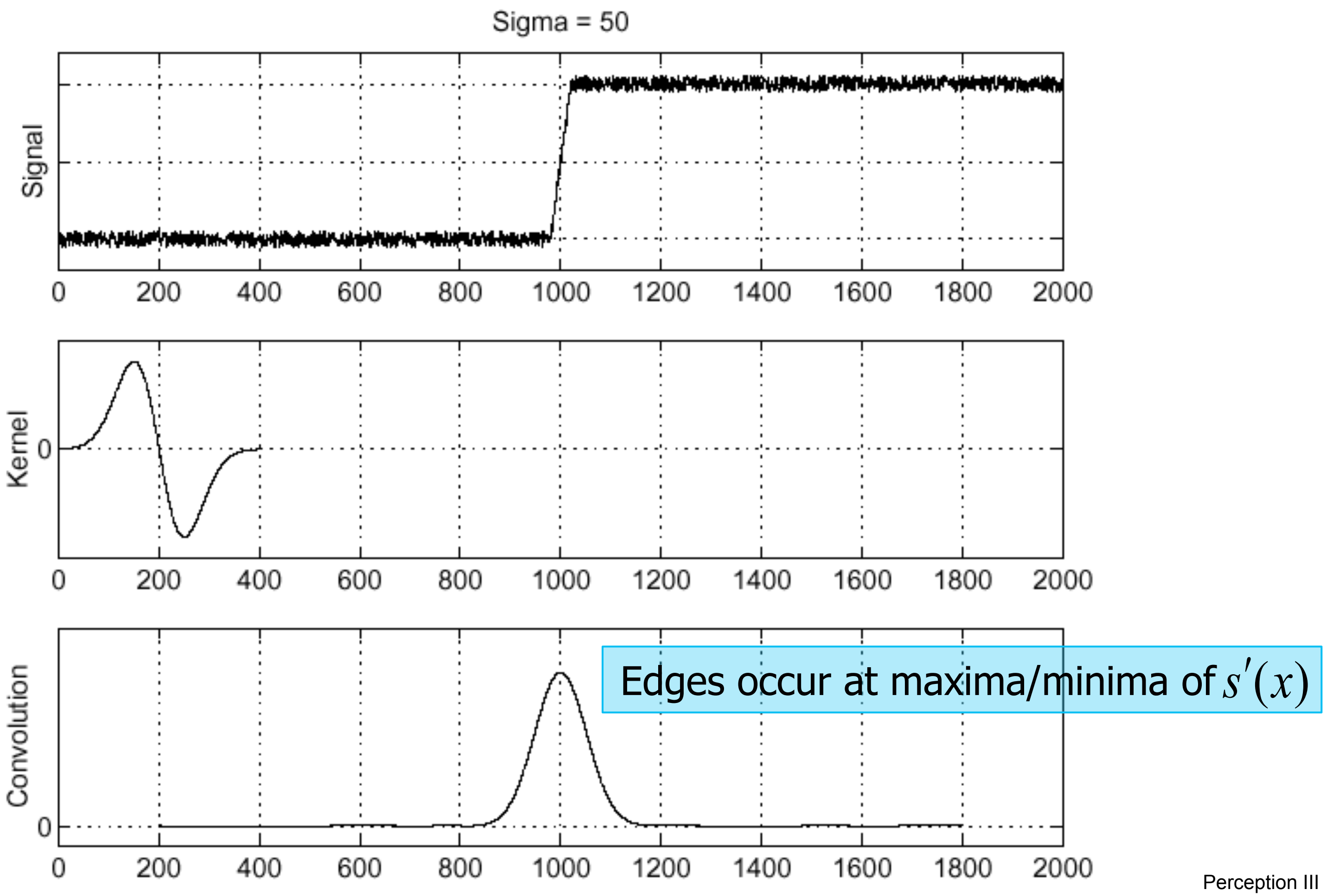
1D edge detection | derivative theorem of convolution

- $s'(x) = \frac{d}{dx}\left(g_\sigma(x) * I(x)\right) = g'_\sigma(x) * I(x)$
- This saves us one operation:

$I(x)$

$g'_\sigma(x) = \frac{d}{dx} g_\sigma(x)$

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



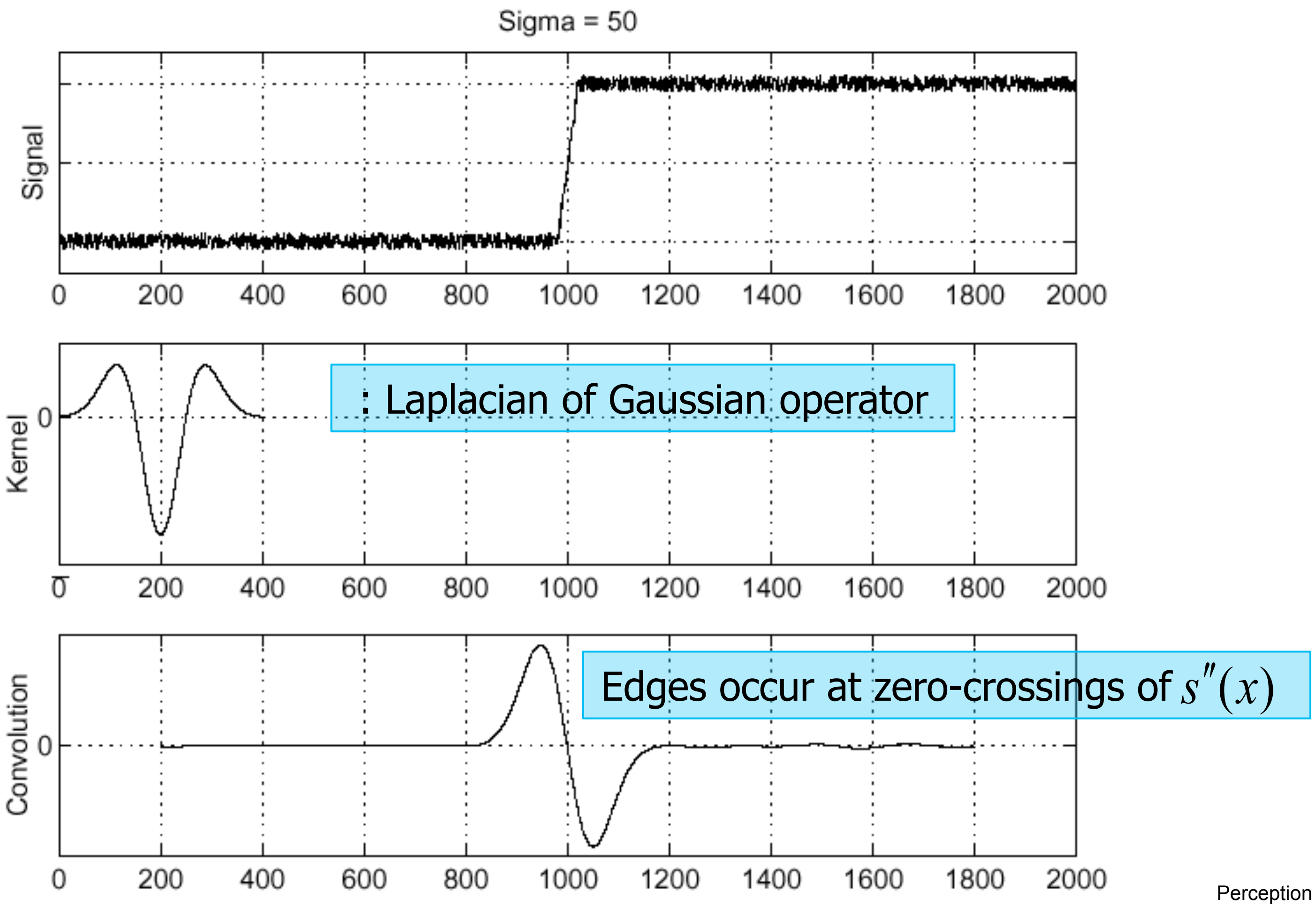
1D edge detection | zero-crossings

- 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

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

- Find gradient of smoothed image in both directions

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

- Discard pixels with $|\nabla S|$ (i.e. edge strength) below a certain threshold
- **Non-maxima suppression**: identify local maxima of $|\nabla S|$
  ⇨ detected edges

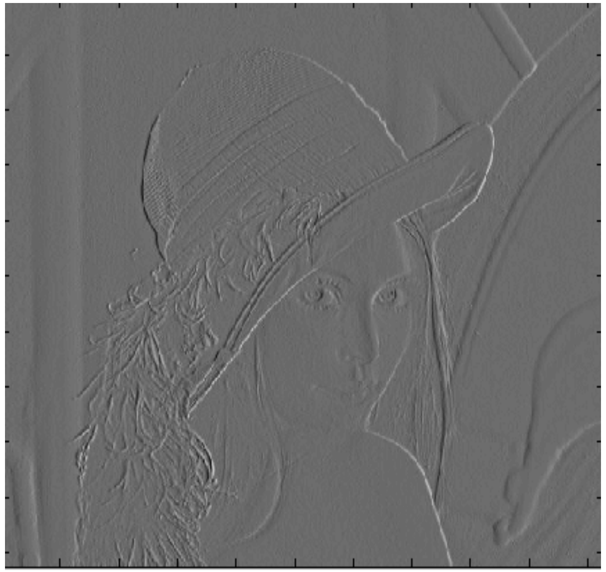
2D edge detection | example

I : original image (Lena)

2D edge detection | example using the Canny edge detector

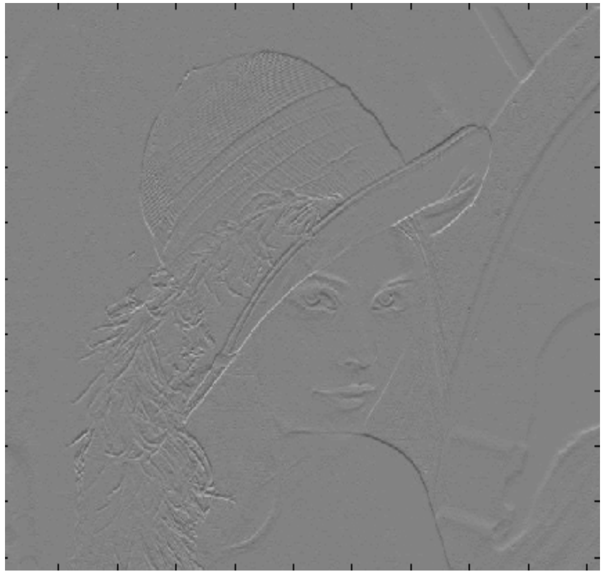
$$|\nabla S| = \sqrt{S_x^2 + S_y^2}$$

$|\nabla S|$: Edge strength



$$S_x = \frac{\partial (G_\sigma * I)}{\partial x}$$

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



$$S_y = \frac{\partial (G_\sigma * I)}{\partial y}$$

2D edge detection | example using the Canny edge detector

Thresholding $|\nabla S|$

2D edge detection | example using the Canny edge detector

V4RL VISION FOR ROBOTICS LAB

Thinning: non-maximal suppression
⇨ **edge image**

2D edge detection | partial derivatives of an image

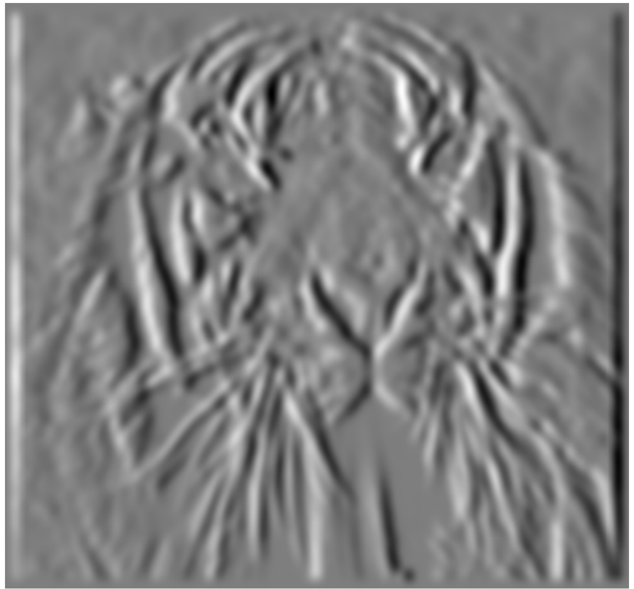
$$S_x = \frac{\partial(G_\sigma * I)}{\partial x}$$

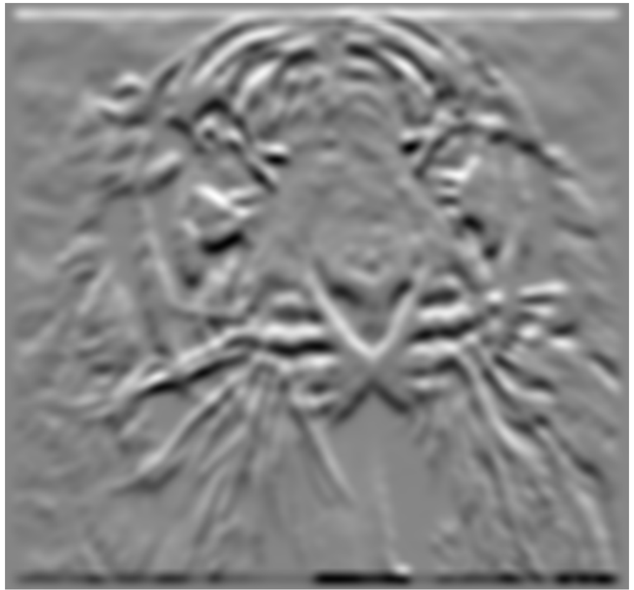
$$F_x = \begin{bmatrix} -1 & 1 \end{bmatrix}$$

$$S_y = \frac{\partial(G_\sigma * I)}{\partial y}$$

$$F_y = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$|\nabla S| = \sqrt{S_x^2 + S_y^2}$$





2D edge detection | other approx. of derivative filters

- Prewitt:

$$F_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad F_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- Sobel:

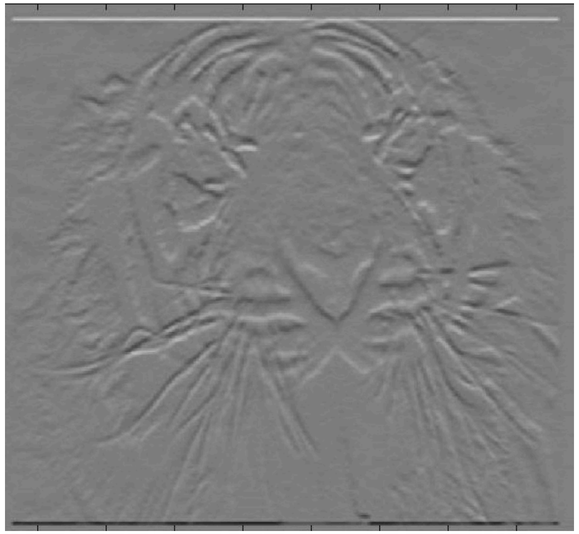
$$F_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad F_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Roberts:

$$F_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad F_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Sample Matlab code

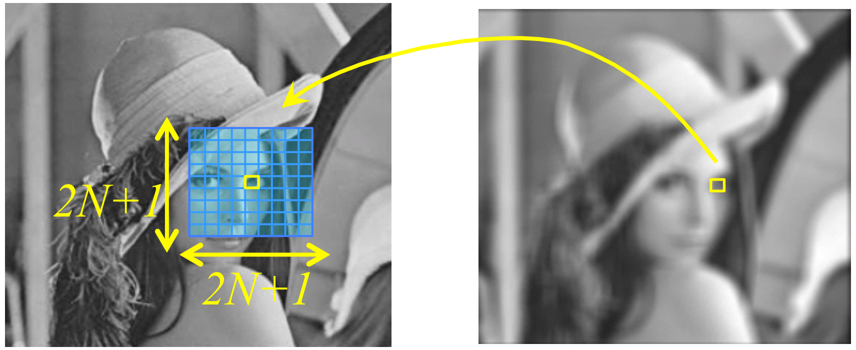
```
>> im = imread('lion.jpg')
>> Fy = fspecial('sobel');
>> outim = imfilter(double(im), Fy);
>> imagesc(outim);
>> colormap gray;
```



Key points on smoothing + derivative masks

Smoothing masks

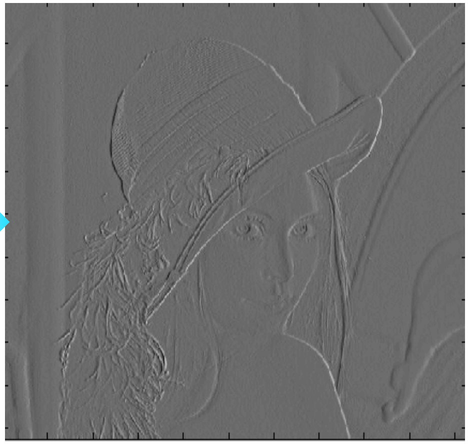
- Values positive
- Always **sum to 1** → constant regions same as input
- Amount of **smoothing proportional to mask size**



Derivative masks

- Opposite signs used to get high response in regions of high contrast
- Always **sum to 0** → no response in constant regions
- High absolute value at points of high contrast

$$F_x = \begin{pmatrix} 1 & -1 \end{pmatrix}$$



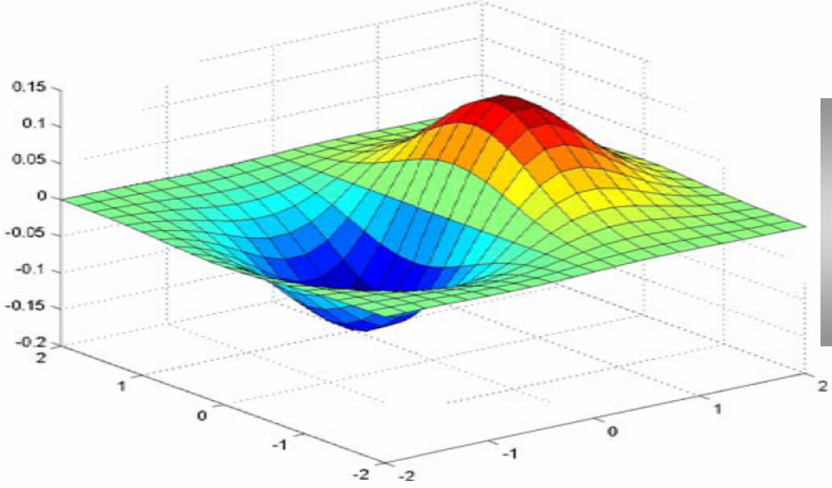
2D edge detection | derivative of gaussian filter

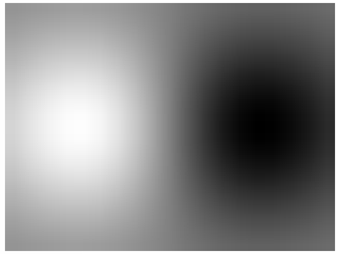
$$\nabla S = \nabla * G * I = \begin{pmatrix} (F_x * G) * I \\ (F_y * G) * I \end{pmatrix}$$

$$F_x = \begin{pmatrix} 1 & -1 \end{pmatrix}$$

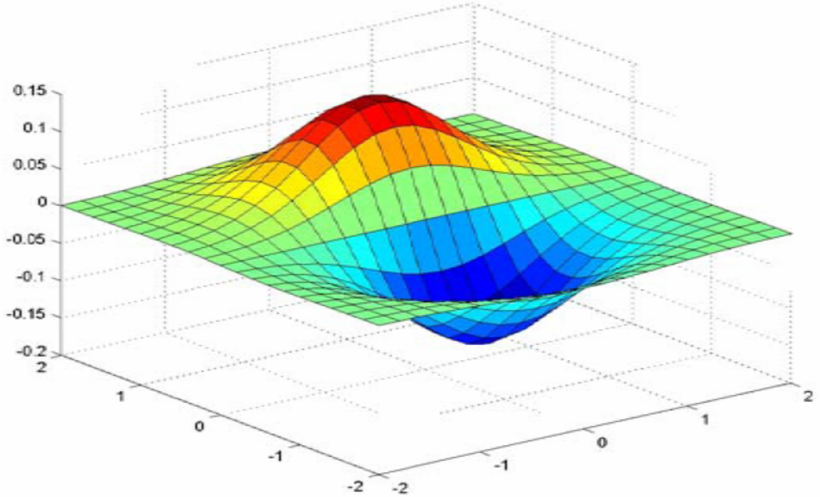
$$F_y = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

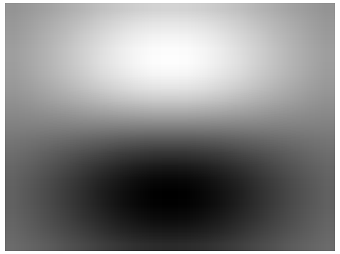
$$G = \begin{pmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{pmatrix}$$





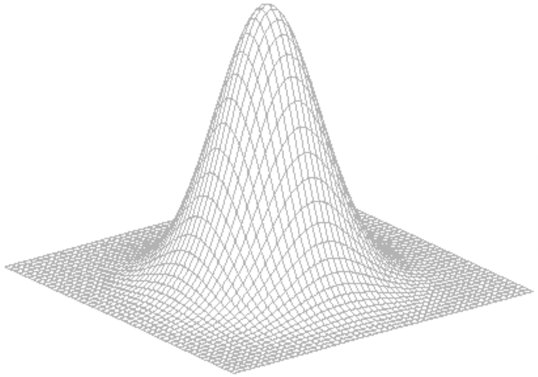
x-direction





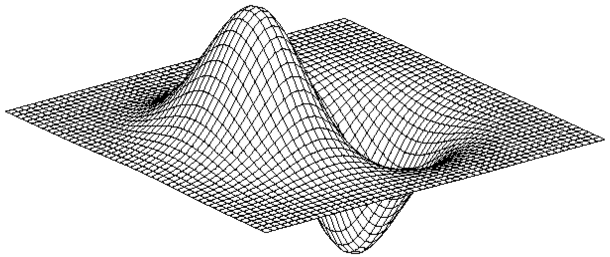
y-direction

2D edge detection | popular edge detection filters



Gaussian

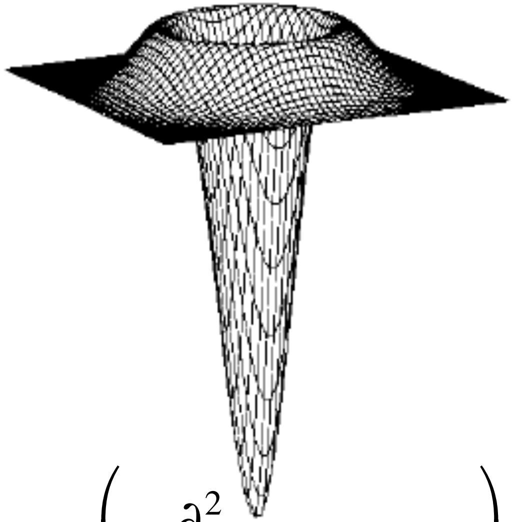
$$G_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



derivative of Gaussian

$$\nabla G_\sigma(u,v) = \begin{pmatrix} \frac{\partial}{\partial u} G_\sigma(u,v) \\ \frac{\partial}{\partial v} G_\sigma(u,v) \end{pmatrix}$$

Laplacian of Gaussian

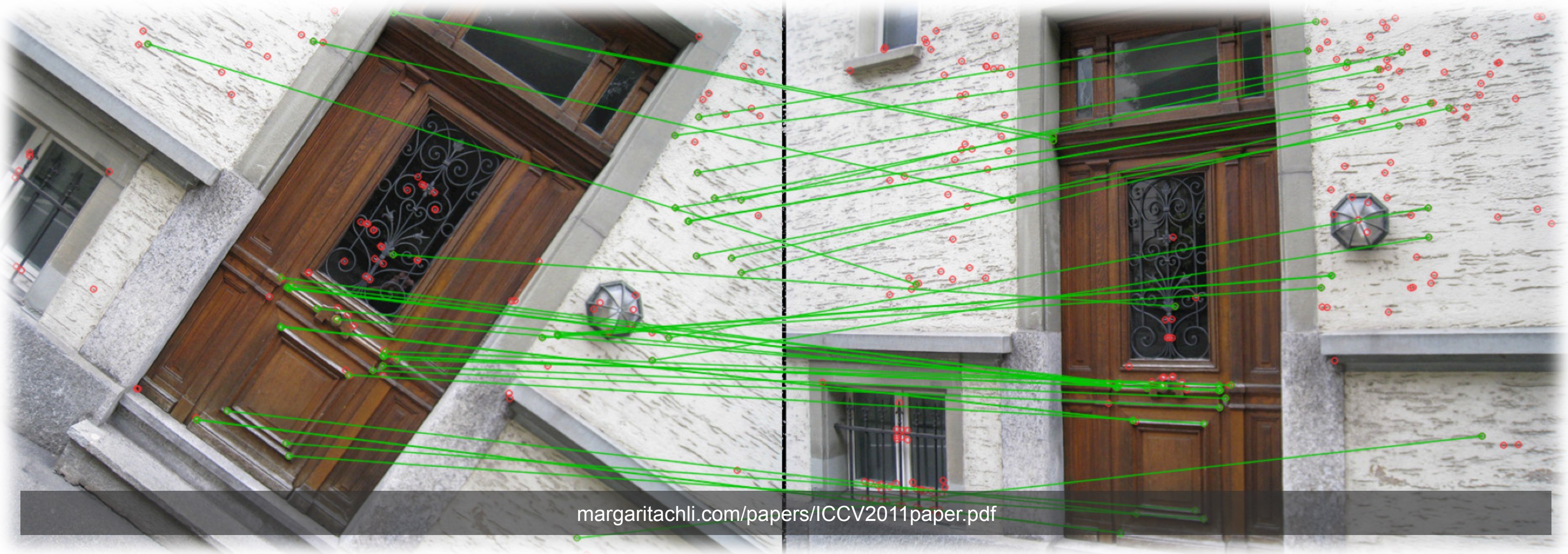


$$\nabla^2 G_\sigma(u,v) = \begin{pmatrix} \frac{\partial^2}{\partial u^2} G_\sigma(u,v) \\ \frac{\partial^2}{\partial v^2} G_\sigma(u,v) \end{pmatrix}$$

Point Features

Image Feature Extraction:

- Edges
- **Points: Shi-Tomasi & Harris corners**
  SIFT features
  - **and more recent algorithms from the state of the art…**



Point features | applications

Point features are widely used in:

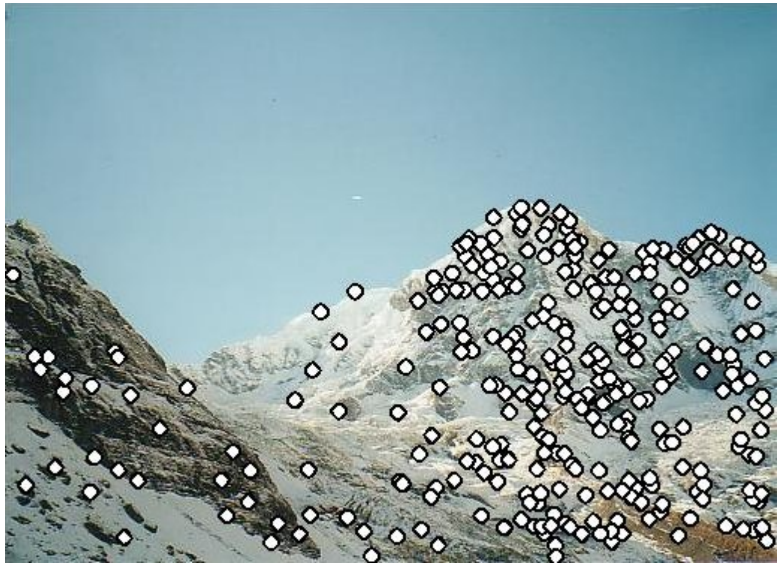
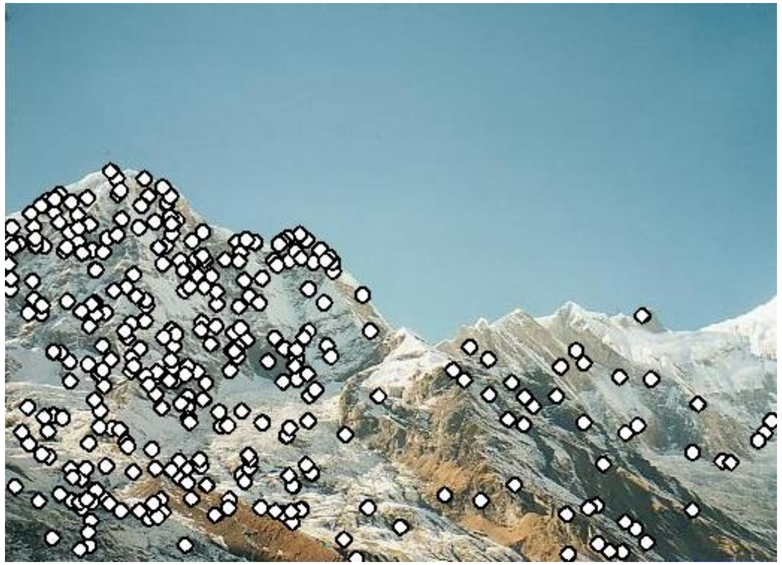
- Robot navigation
- Object recognition
- 3D reconstruction
- Motion tracking
- Indexing and database retrieval ⇨ Google Images
- …
- Image stitching: this panorama was generated using **AUTOSTITCH** (freeware)

Point features | how to build a panorama?

- We need to match (align) images

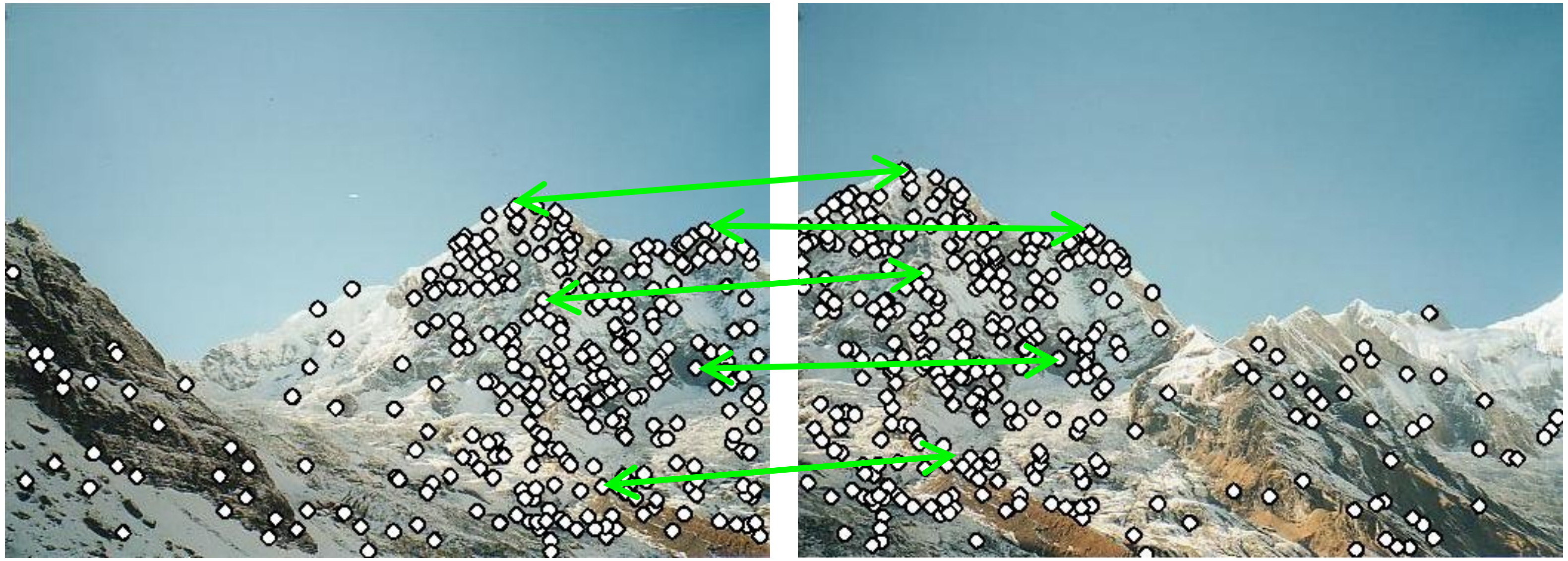
Point features | how to build a panorama?

- Detect feature points in both images

Point features | how to build a panorama?

- Detect feature points in both images
- Find corresponding pairs



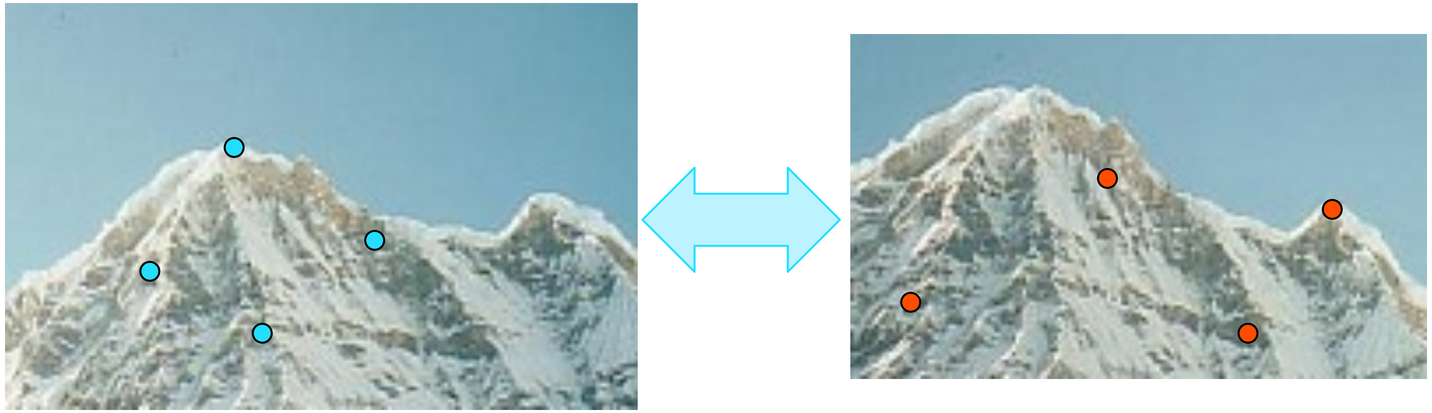
Point features | how to build a panorama?

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

Point features | feature extraction

Problem 1:

- Detect the **same** points **independently** in both images, if they are in the field of view



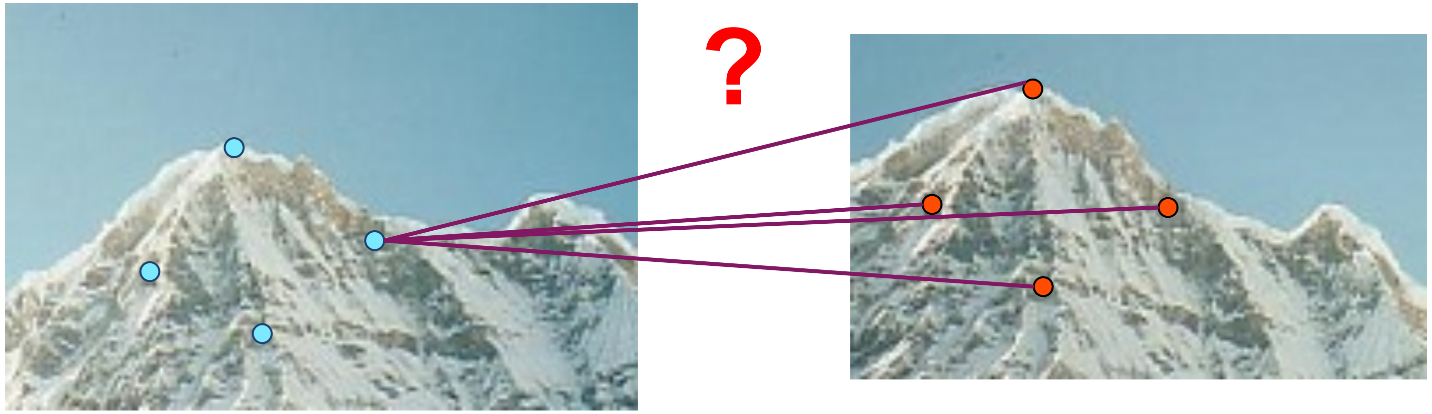
no chance to match!

We need a **repeatable** feature detector

Point features | feature matching

Problem 2:

- For each point, identify its correct correspondence in the other image(s)



We need a **reliable** and **distinctive** feature descriptor

Point features | what is a distinctive feature?

- Some patches can be localized or matched with higher accuracy than others

Image 1

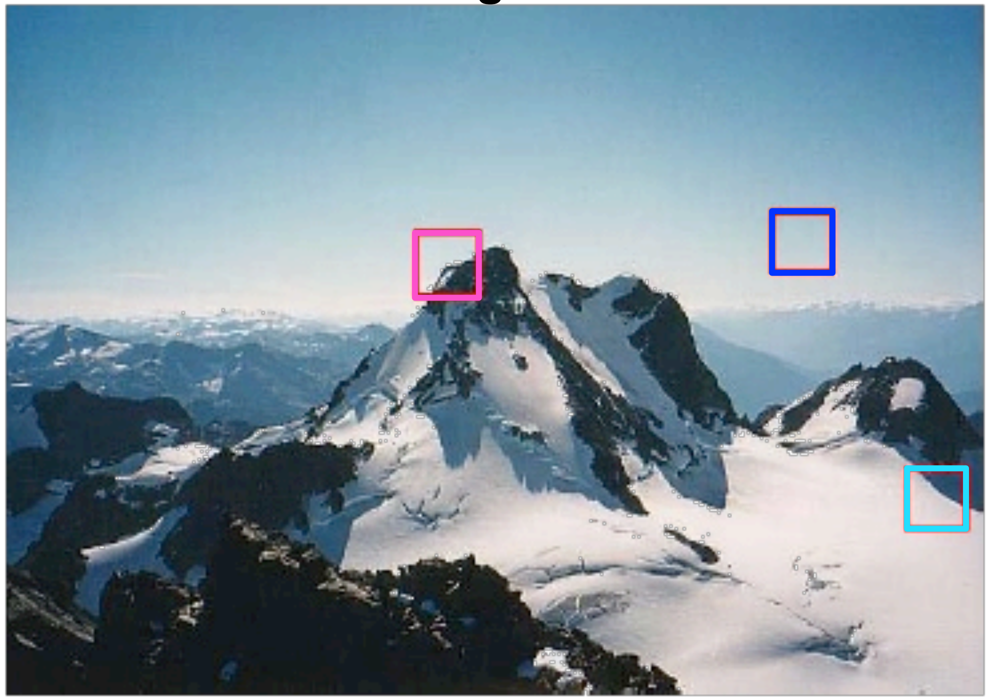
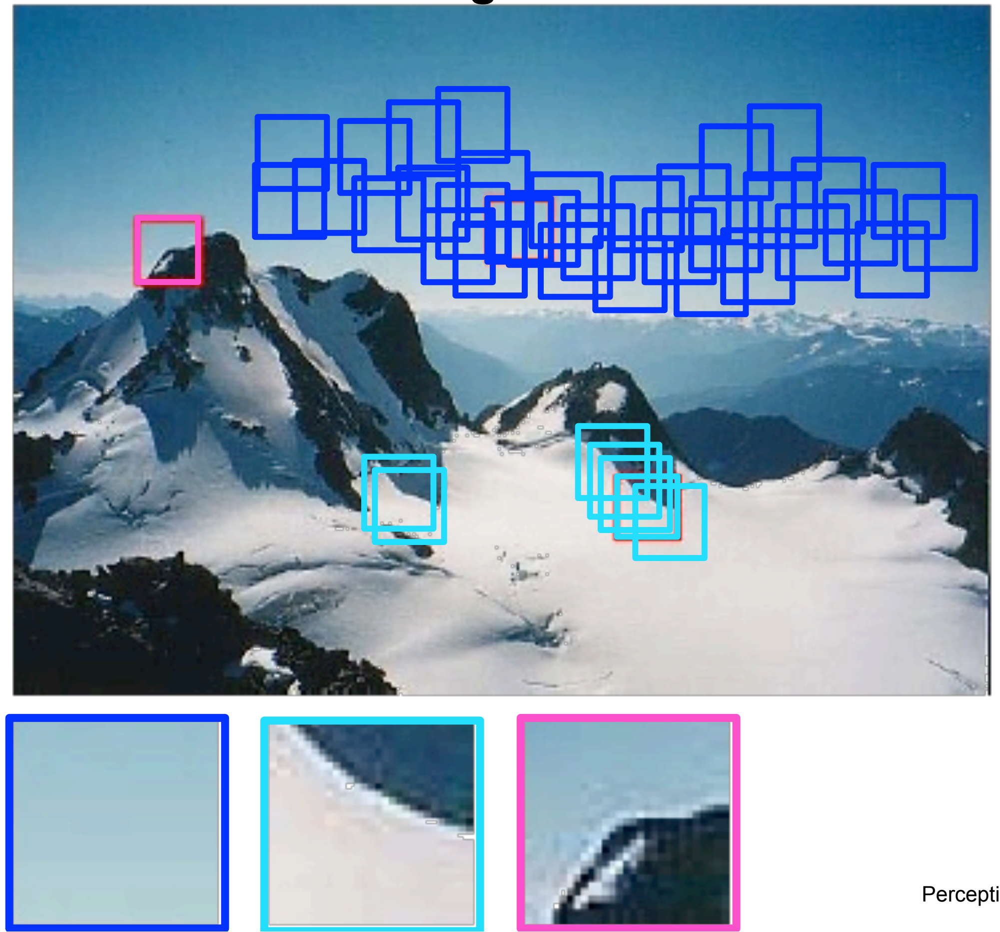


Image 2



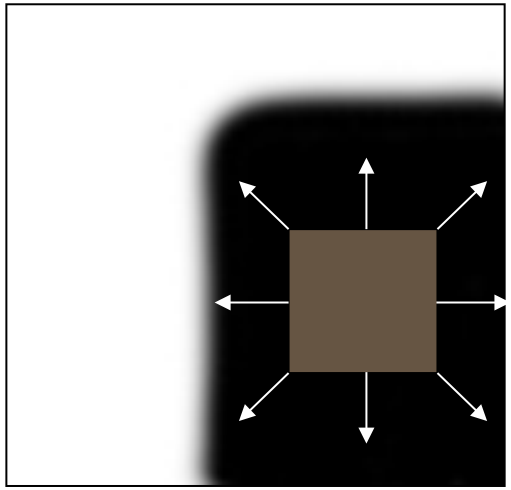
Corner detection

Corner detection | identifying corners

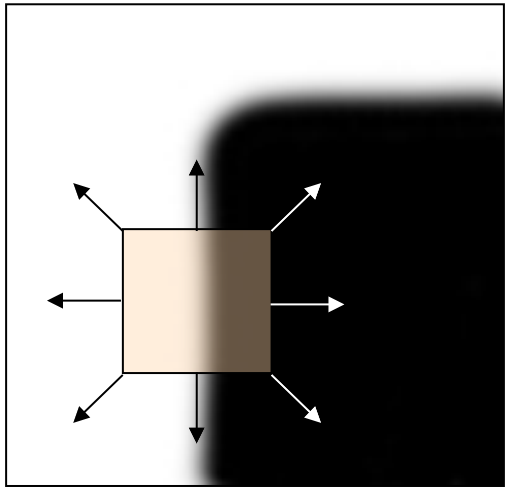
- **Key property:** in the region around a corner, image gradient has **two or more** dominant directions
- Corners are **repeatable** and **distinctive**

Corner detection | identifying corners

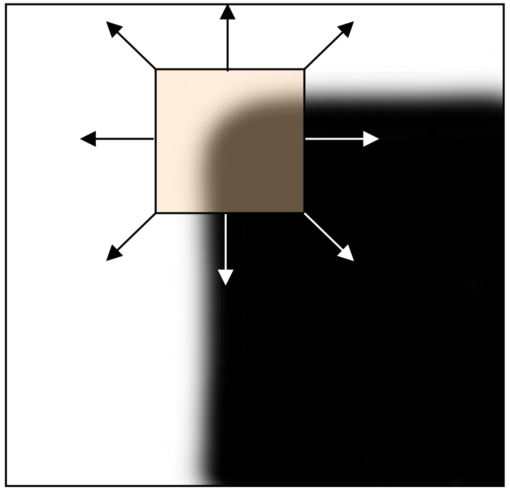
- How do we identify corners?
- Shifting a window in **any direction** should give a **large change** of intensity in at least 2 directions



"flat" region:
no intensity change



"edge":
no change along the edge direction



"corner":
significant change in at least 2 directions

Corner detection | how do we implement this?

- Two image patches of size P one centered at (x, y) and one centered at $(x+\Delta x, y+\Delta y)$
- The Sum of Squared Differences between them is:

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

- Let $I_x = \frac{\partial I(x,y)}{\partial x}$ and $I_y = \frac{\partial I(x,y)}{\partial y}$. 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

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

Corner detection | how do we implement this?

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

- This can be written in a matrix form as

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

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

$$M = \sum_{x,y \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

"Second moment matrix"

Corner detection | interpreting matrix M

- Since M is symmetric $\Rightarrow M = \sum_{x,y \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

- The Harris (and the "Shi-Tomasi") detector analyses the **eigenvalues,** λ_1 and λ_2, to decide if we are in presence of a corner ⇨ i.e. look for large intensity changes in at least 2 directions

Corner detection | eigen decomposition

- The **eigenvectors** $\boldsymbol{v}$ and **eigenvalues** λ of a square matrix A satisfy:

$$A\boldsymbol{v}=\lambda\boldsymbol{v}$$

- Then is an eigenvector $\boldsymbol{v}$ of A and λ is the corresponding eigenvalue.
- The eigenvalues are found by solving: $\det(A-\lambda I)=0$
- In this case, $A=M$ is a 2 x 2 matrix, so:

$$\det\begin{bmatrix} m_{11}-\lambda & m_{12} \\ m_{21} & m_{22}-\lambda \end{bmatrix}=0$$

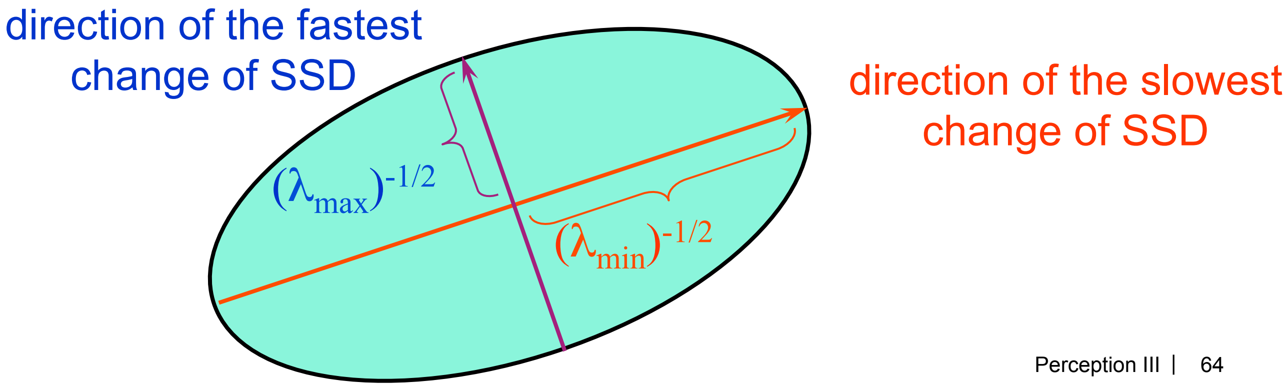
$$\lambda_{1,2}=\frac{1}{2}\left[(m_{11}+m_{22})\pm\sqrt{4m_{12}m_{21}+(m_{11}-m_{22})^2}\right]=0$$

- For each λ we can find its corresponding $\boldsymbol{v}$ (forming the columns in R) by

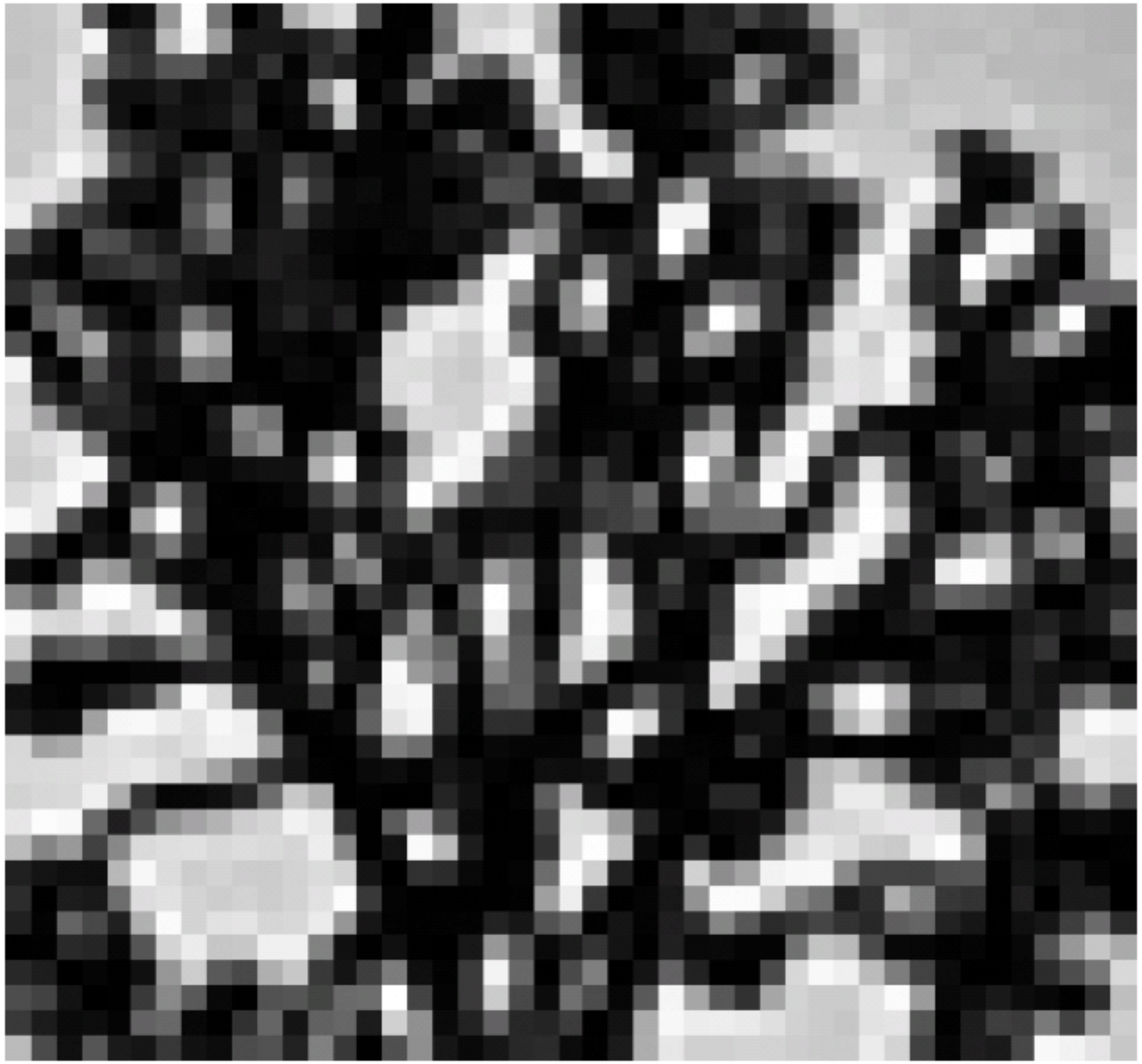
$$\begin{bmatrix} m_{11}-\lambda & m_{12} \\ m_{21} & m_{22}-\lambda \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=0$$

Corner detection | interpreting matrix M

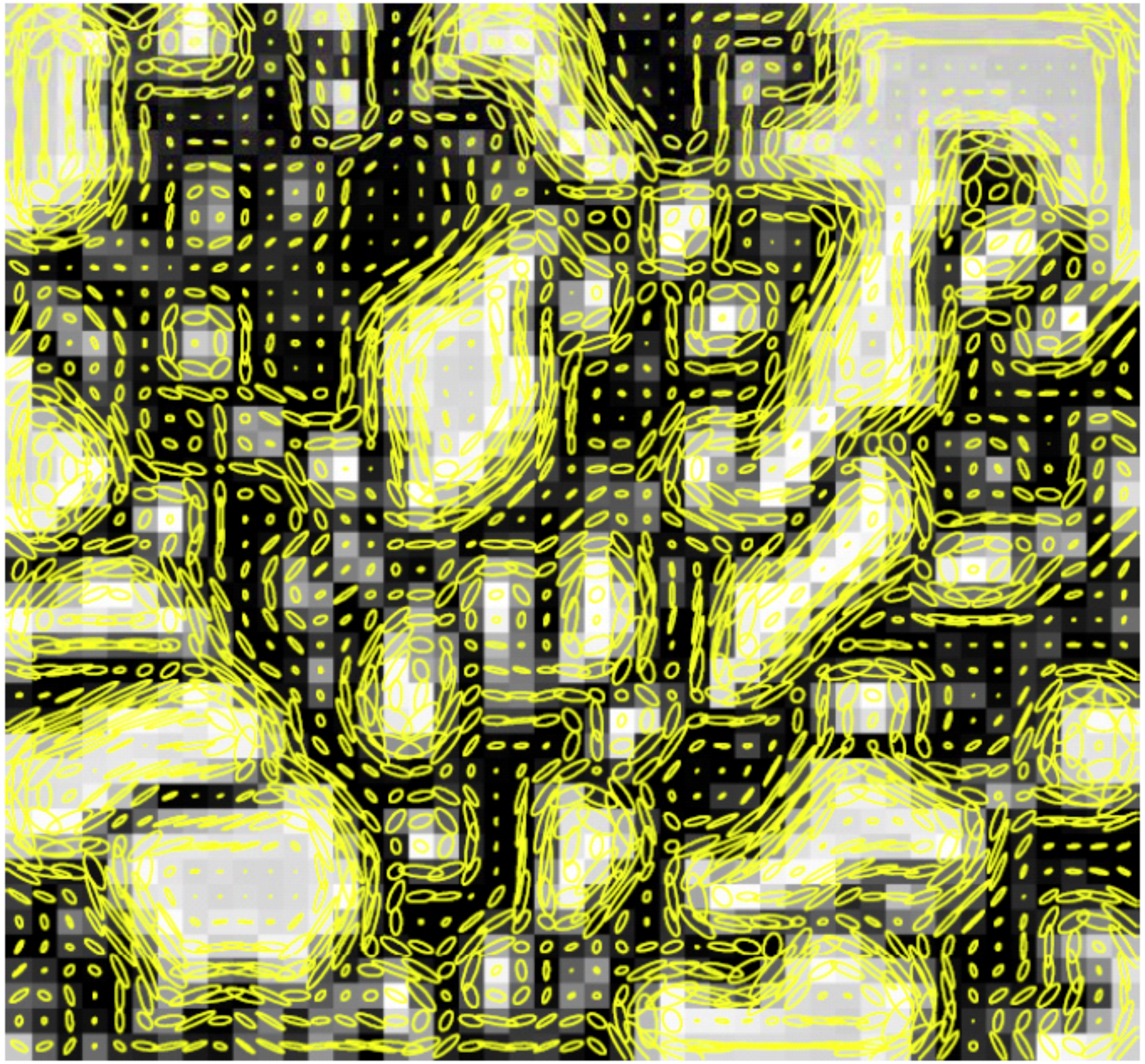
- Since M is symmetric $\Rightarrow M = \sum_{x,y \in P} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$
- The Harris (and the "Shi-Tomasi") detector analyses the **eigenvalues,** λ_1 and λ_2 , to decide if we are in presence of a corner ⇨ i.e. look for large intensity changes in at least 2 directions
- We can visualize $\begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = const$ as an **ellipse** with axis-lengths determined by λ_1 and λ_2 and the axes' orientations determined by R (i.e. the **eigenvectors** of M)
- The (two) eigenvectors identify the orthogonal directions of largest and smallest changes of SSD



Corner detection | visualization of 2nd moment matrices



Corner detection | visualization of 2nd moment matrices

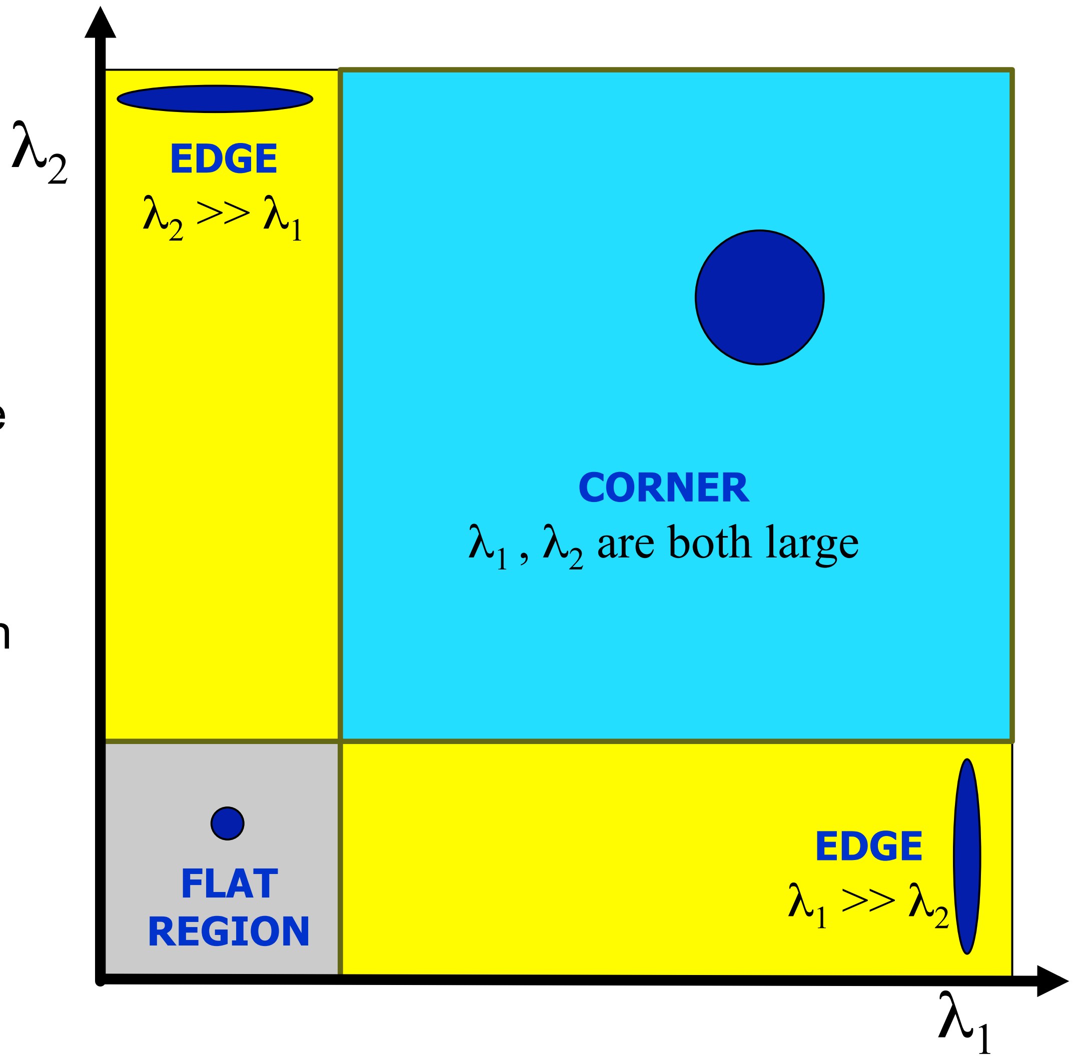


Corner detection | interpreting the eigenvalues

Does 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:** $\lambda_1 >> \lambda_2$ 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**.
- **Shi-Tomasi** [1] cornerness criterion:

$$C_{SHI\text{-}TOMASI} = min(\lambda_1,\lambda_2) > thresh.$$



[1] J. Shi and C. Tomasi. "Good Features to Track,". CVPR 1994

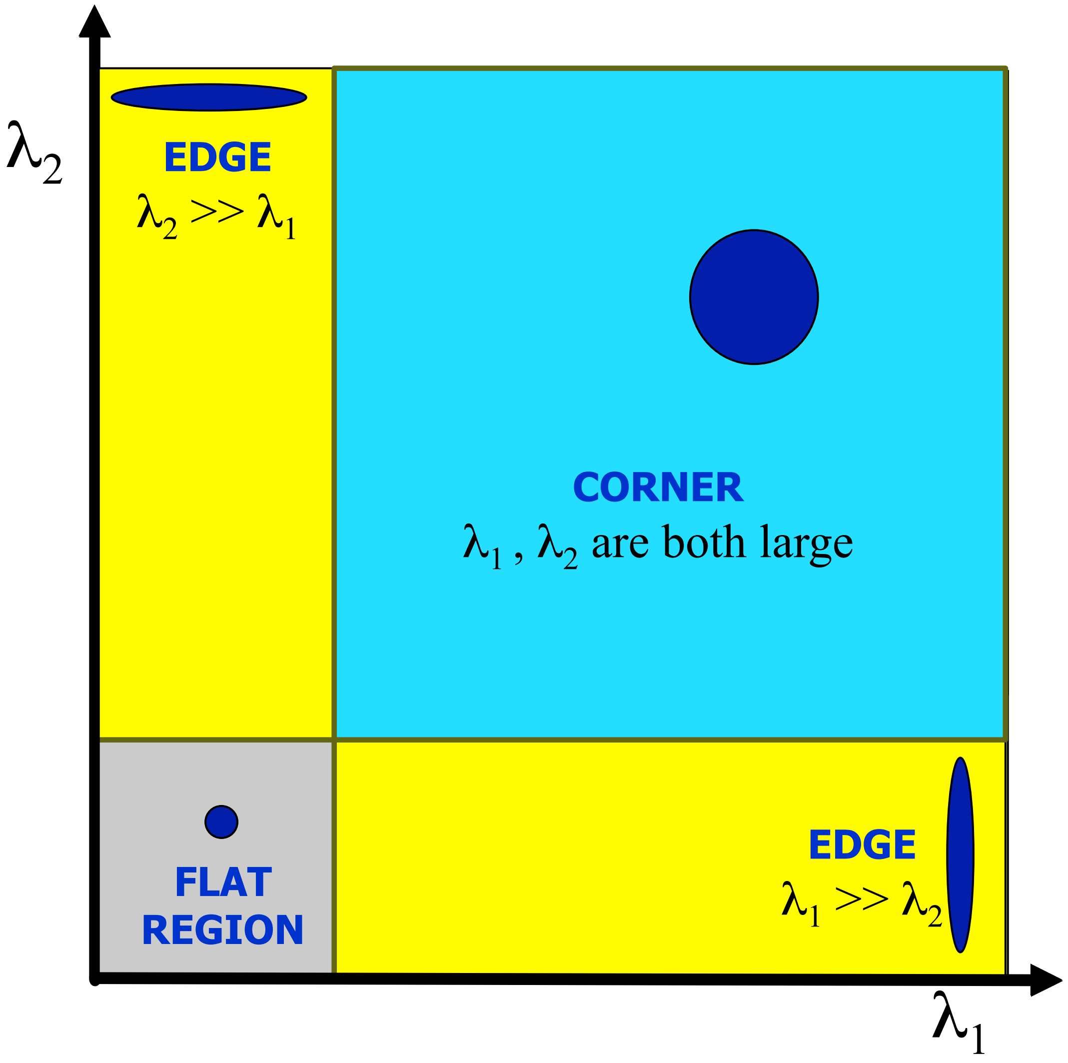
Corner detection | corner response function

- Computing λ_1 and λ_2 is expensive
  ⇨ Harris & Stephens suggested using a "**cornerness function**" instead:

$$C_{HARRIS} = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(M) - \kappa \cdot trace^2(M)$$

  where $\kappa = const. \in [0.04, 0.15]$

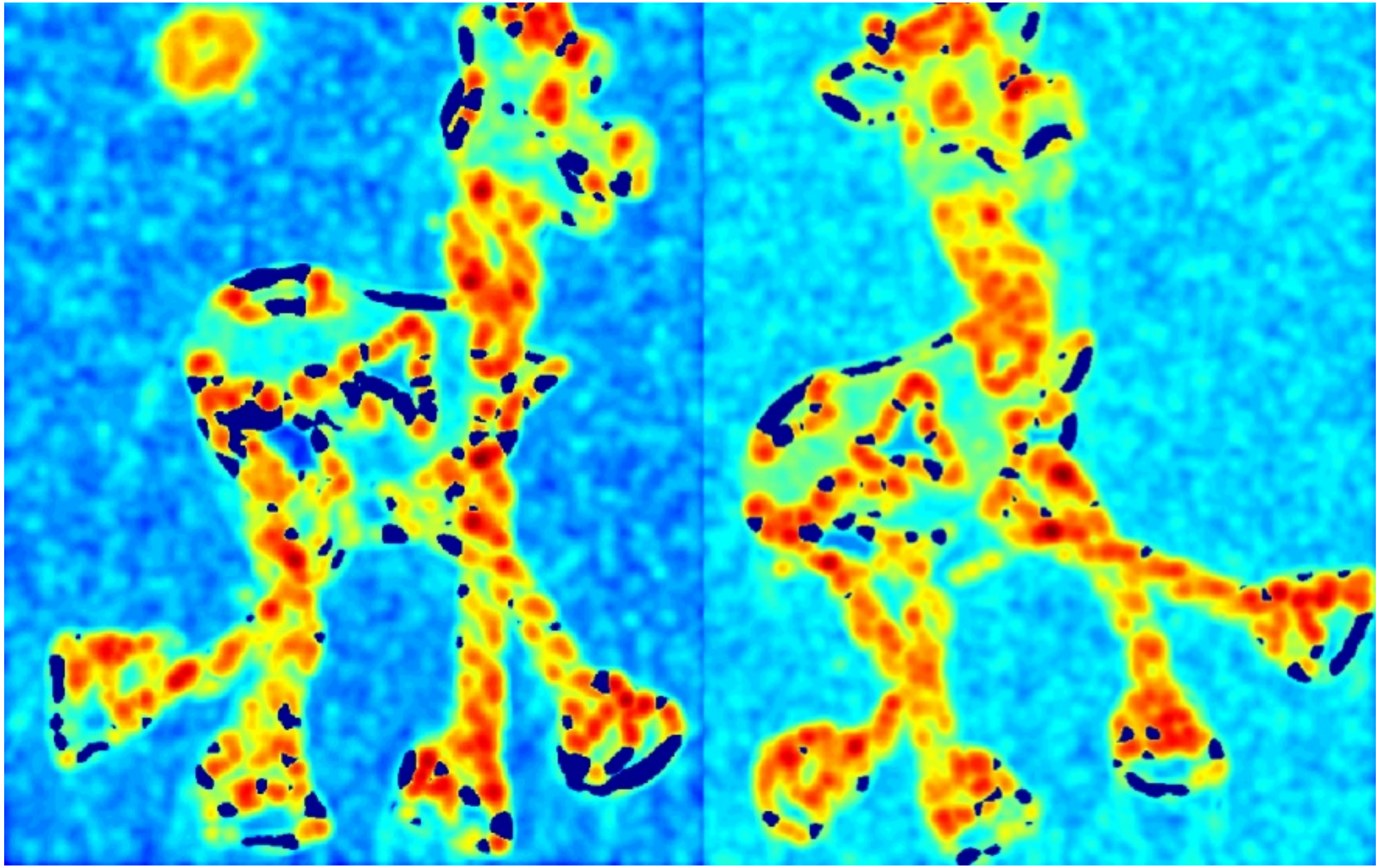
- **Harris** cornerness criterion [2]
- Last step of Harris corner detector: extract local maxima of the cornerness function



[2] C.Harris and M.Stephens. "A Combined Corner and Edge Detector.", *Proceedings of the Alvey Vision Conference, 1988*

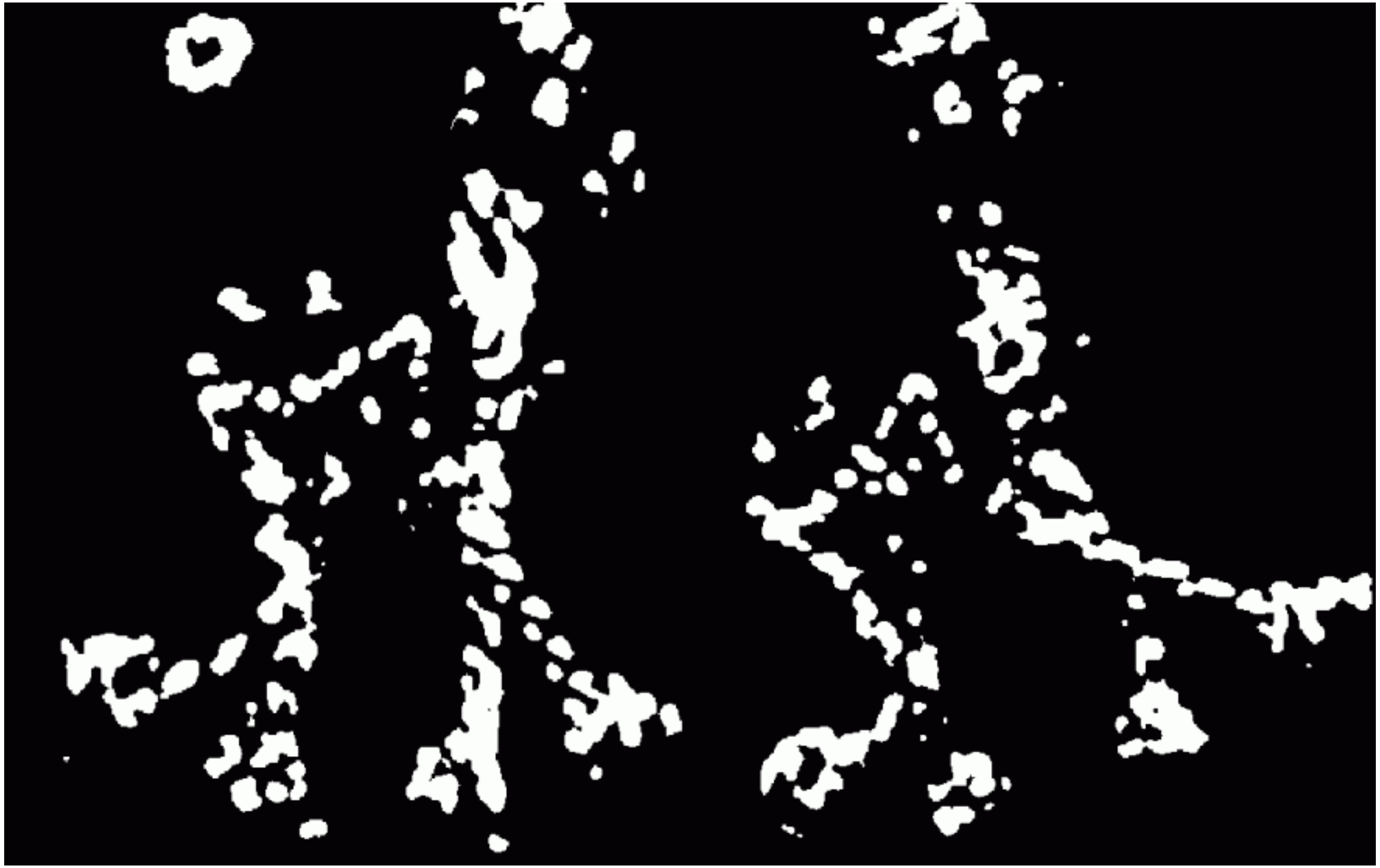
Harris corners | workflow

Harris corners | workflow



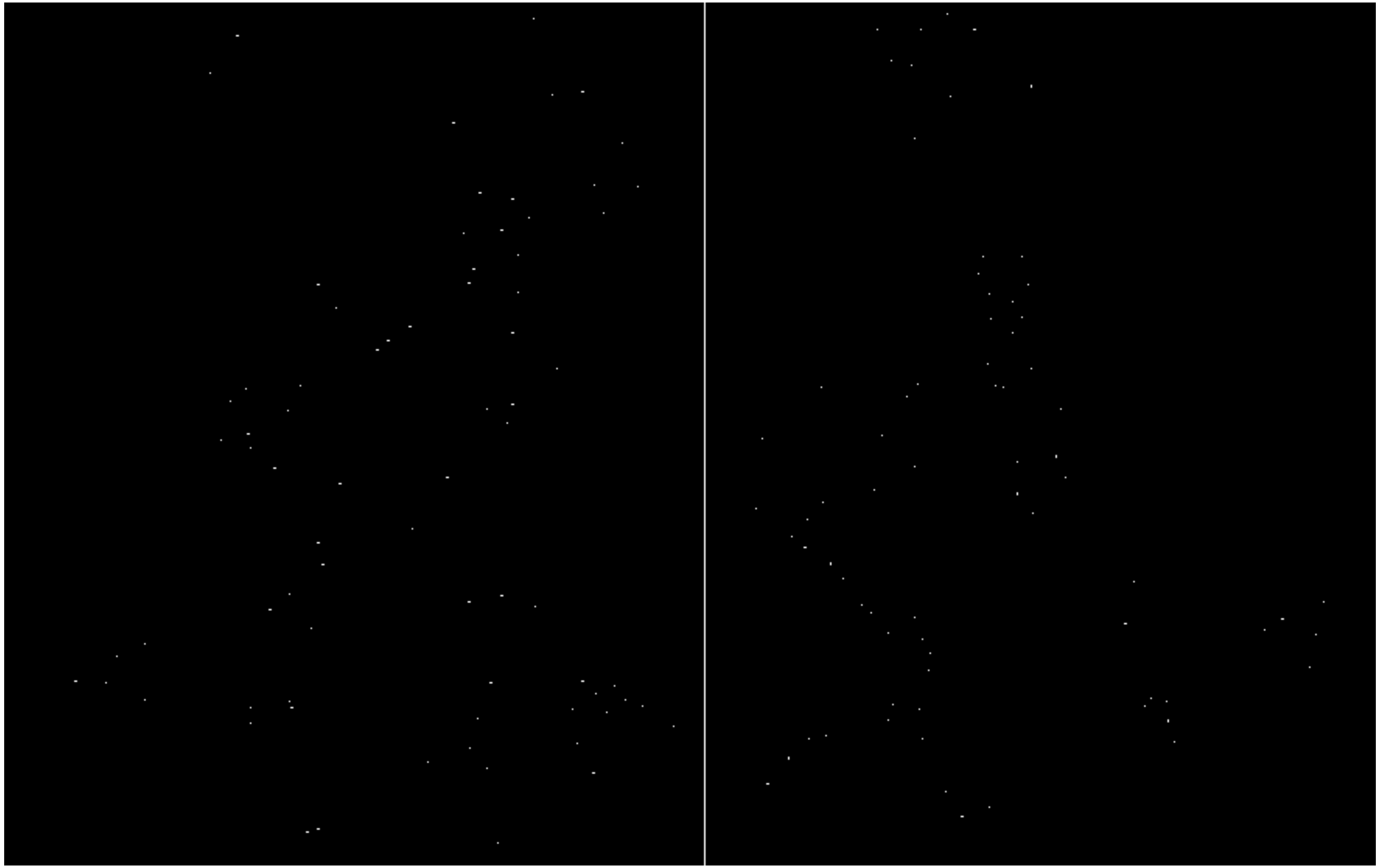
- Compute corner response C

Harris corners | workflow



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

Harris corners | workflow

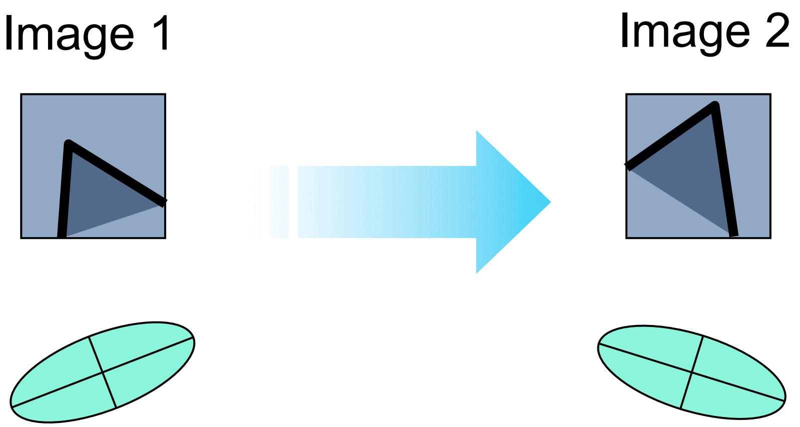


- Take only the points of local maxima of thresholded C

Harris corners | workflow

Harris corners | properties

- Rotation invariance?

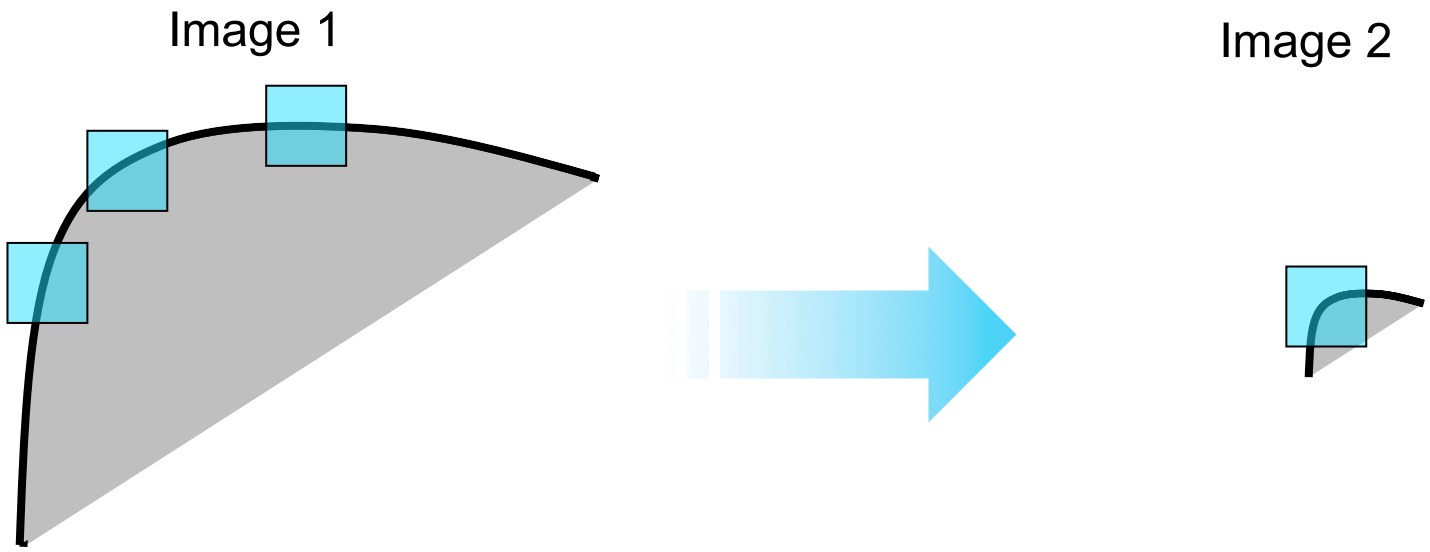


Ellipse rotates, but its shape (i.e. eigenvalues) remains the same

Harris corners are **invariant to image rotation**

Harris corners | properties

- Scale invariance?



All points will be classified as **edges**

Corner!

Harris corners are **not invariant to scale change**

Harris corners | properties' summary

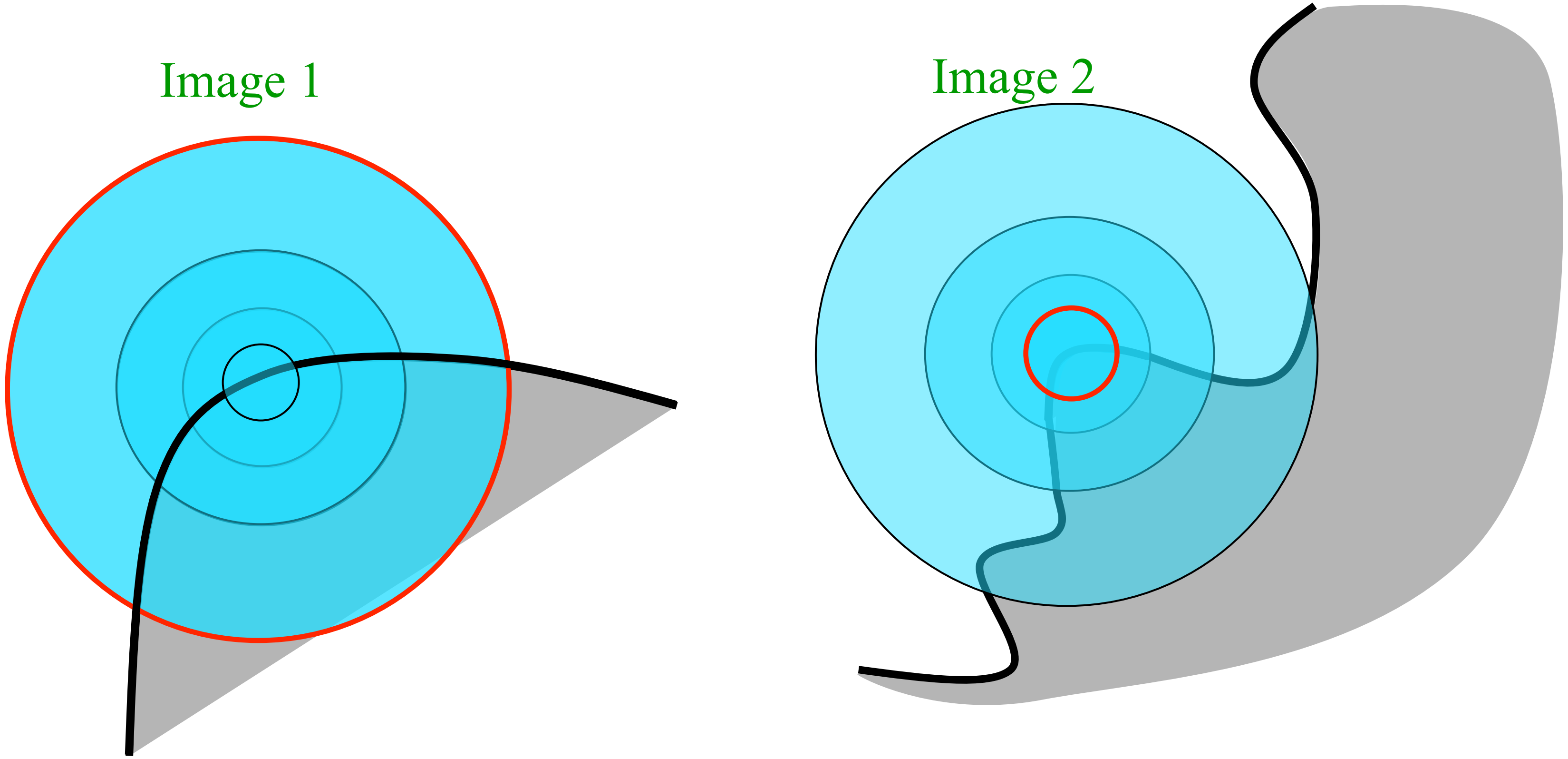
- Harris detector: probably the most widely used and known corner detector

- The detection is invariant to
  - Rotation
  - Linear intensity changes
  - **note:** to make the matching invariant to these we need suitable descriptor & matching

- The detection is NOT invariant to
  - Scale changes
  - Geometric affine changes: an image transformation, which distorts the neighborhood of the corner, can distort its '*cornerness*' response

Scale-invariant feature detection

- Consider regions (e.g. discs) of different sizes around a point
- **Aim**: corresponding regions look the same in image space, when the appropriate scale-change is applied



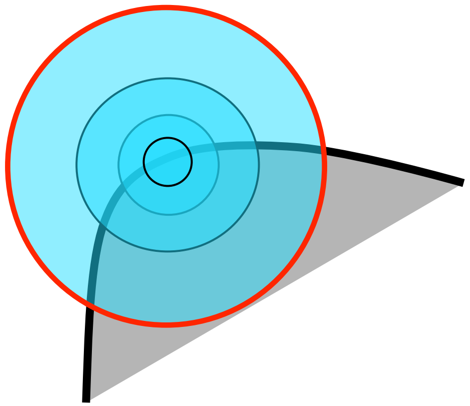
- Choose corresponding regions (discs) **independently** in each image

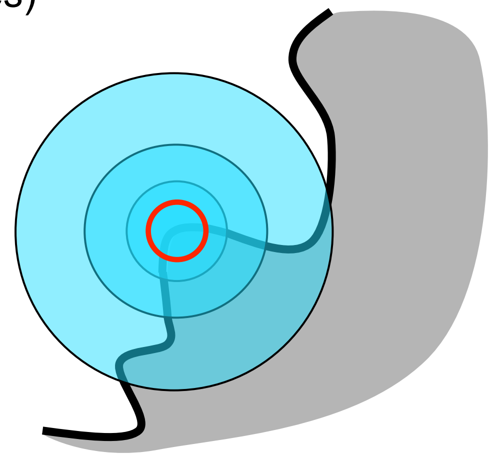
Scale-invariant feature detection

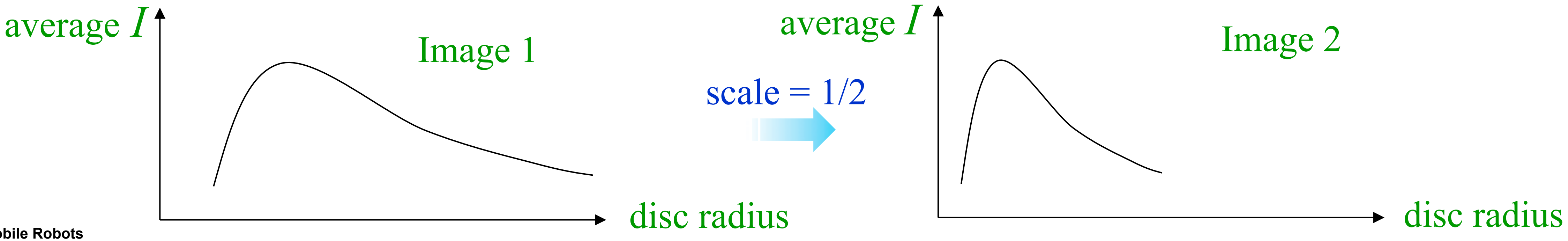
- Approach: design a function to apply on the region (disc) , which is “scale invariant”

(i.e. remains constant for corresponding regions, even if they are at different scales)

<u>example</u>: **average image intensity** over corresponding regions (at different scales) should remain constant



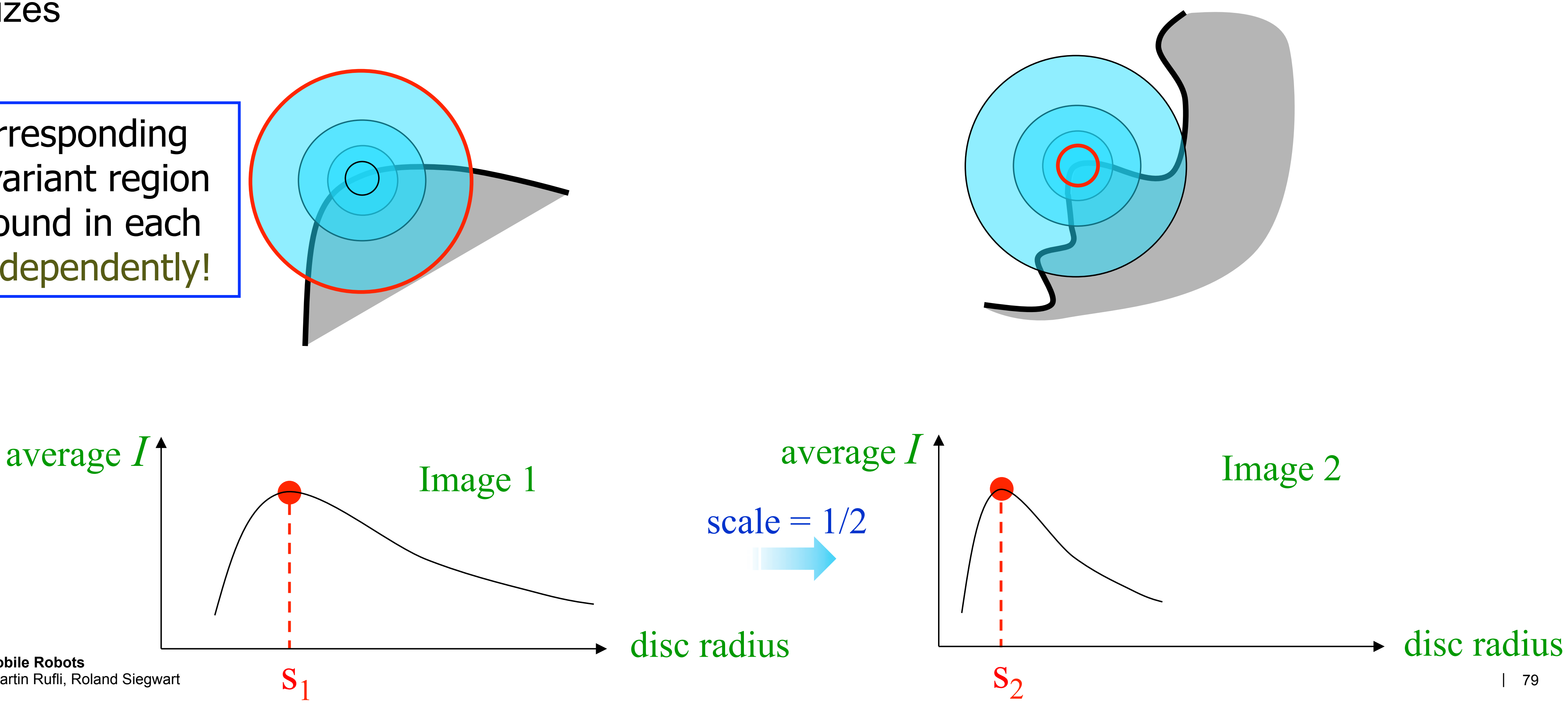




Scale-invariant feature detection

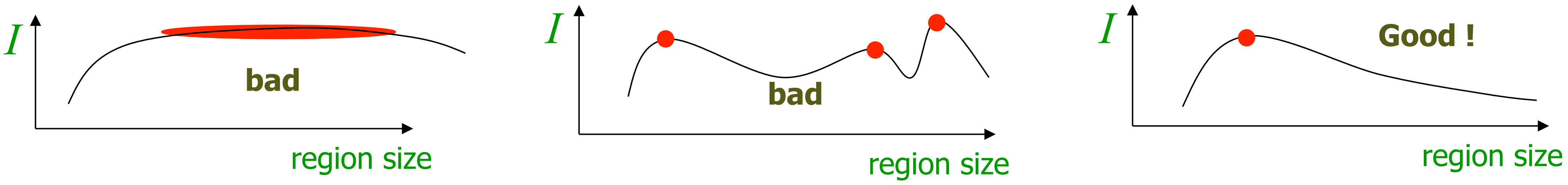
- Identify the local maximum in each response ⇨ these occur at corresponding region sizes

The corresponding scale-invariant region size is found in each image independently!



Scale-invariant feature detection

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



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

⇨ look for blobs or corners (i.e. sharp intensity discontinuities)

Scale-invariant feature detection | LoG scale detector

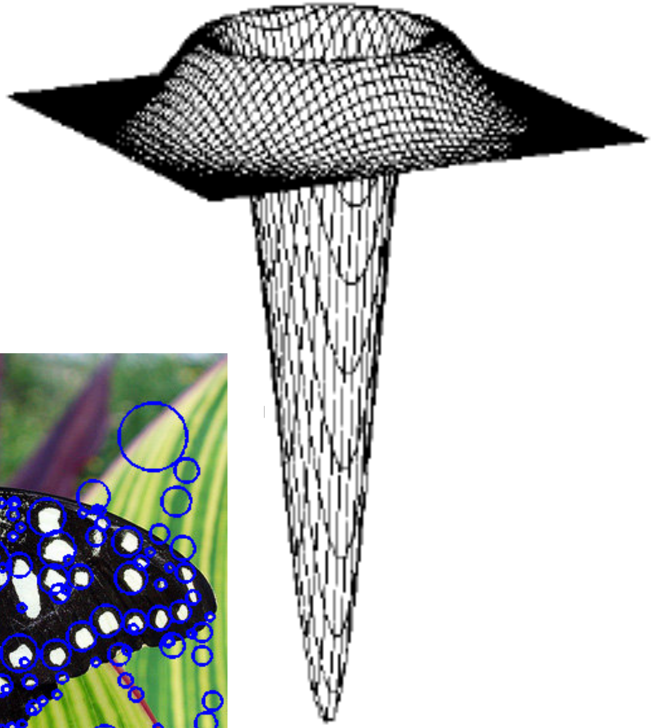
- Functions of determining scale: convolve image with kernel to identify sharp intensity discontinuities

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

- Detected scale corresponds to **local maxima or minima** of the convolved image region

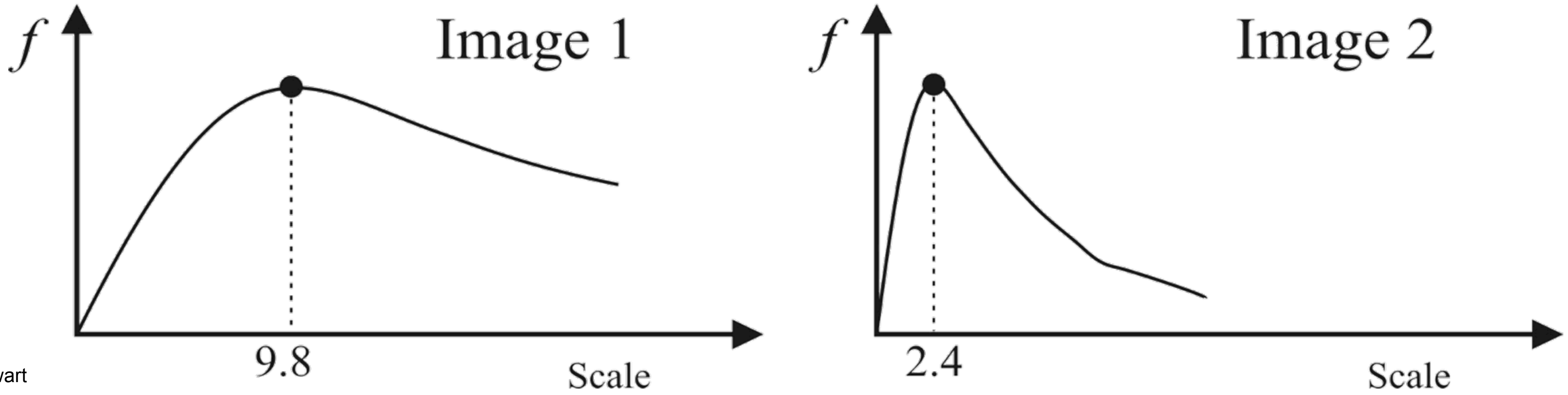
$$LoG = \nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$

Laplacian of Gaussian



Scale-invariant feature detection | LoG scale detector

- Response of LoG for corresponding regions:



Scale-invariant feature detection | DoG scale detector

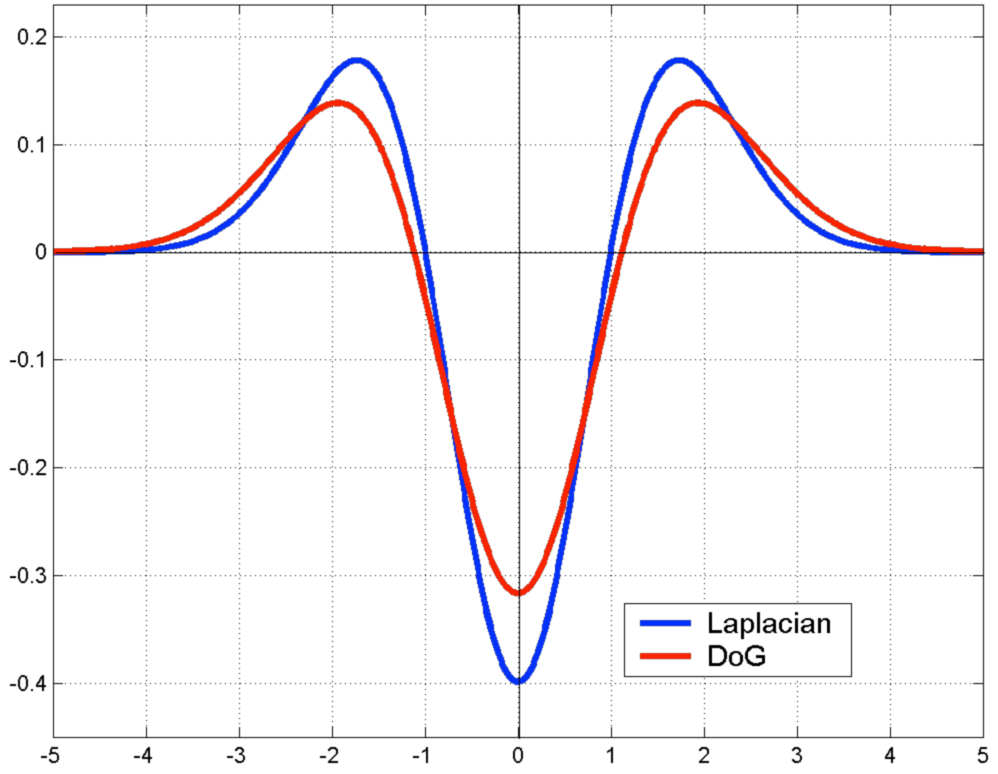
- Approximation to the LoG kernel for efficiency:

  Difference of Gaussians (DoG) kernel:

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

- Used in the **SIFT** feature detector [Lowe et al., IJCV 2004]

- The **SURF** feature detector [Bay et al, CVIU 2008] implements the DoG kernel using a linear combination of rectangular functions

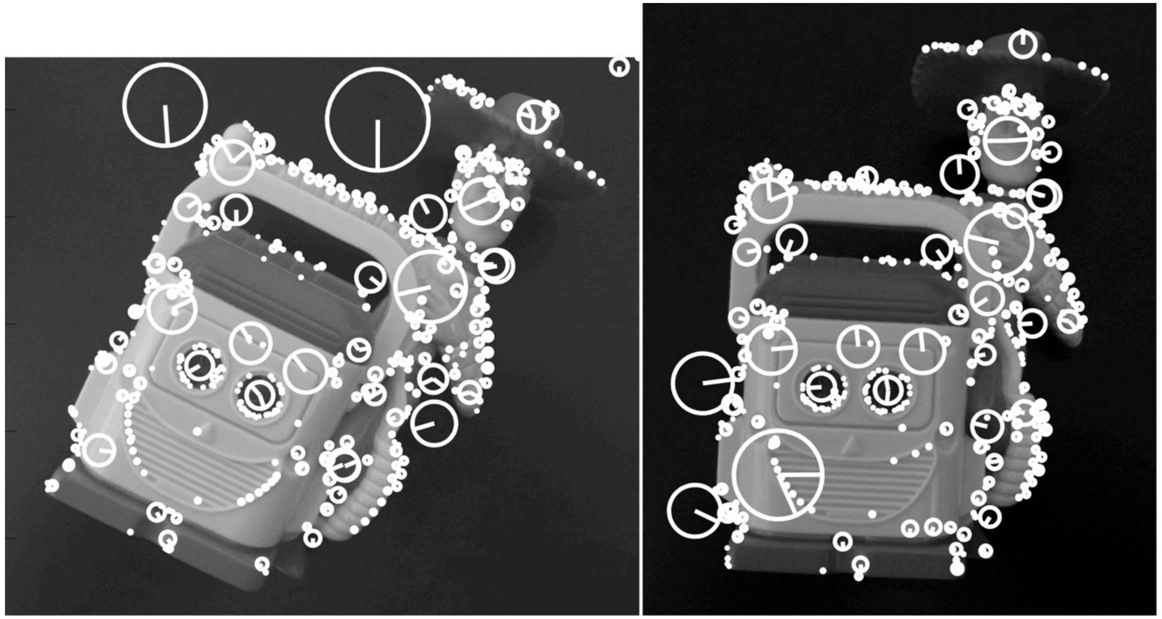


SIFT features [Lowe et al., IJCV 2004]

- **SIFT** = Scale Invariant Feature Transform
an approach for detecting and describing regions of interest in an image
- SIFT features are reasonably **invariant** to changes in:
rotation, scaling, changes in viewpoint, illumination
- SIFT **detector** uses **DoG kernel**, SIFT **descriptor** is based on **gradient orientations**
- Very powerful in capturing + describing **distinctive** structure, but also **computationally demanding**

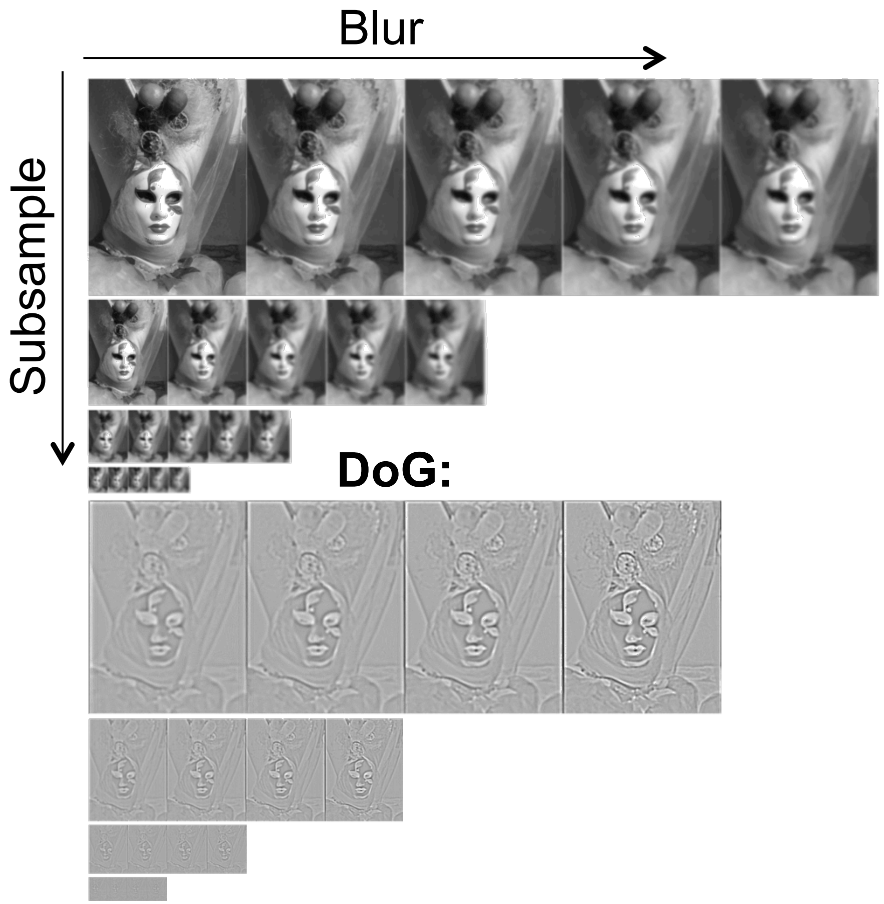
Main SIFT stages:

1. Extract keypoints + scale
2. Assign keypoint orientation
3. Generate keypoint descriptor

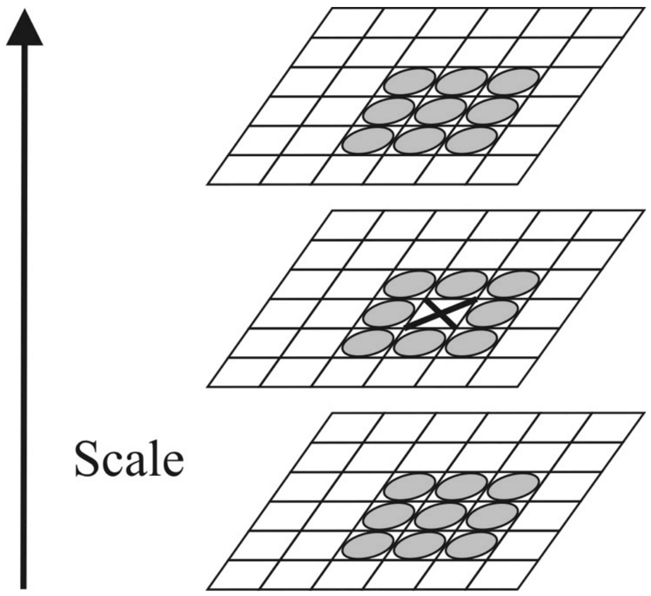


SIFT features | detector (keypoint location + scale)

1. Scale-space pyramid: subsample and blur original image



3. Keypoints: local extrema in the DoG pyramid



SIFT features | keypoint orientation assignment

Define “orientation” of keypoint to achieve **rotation invariance**

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

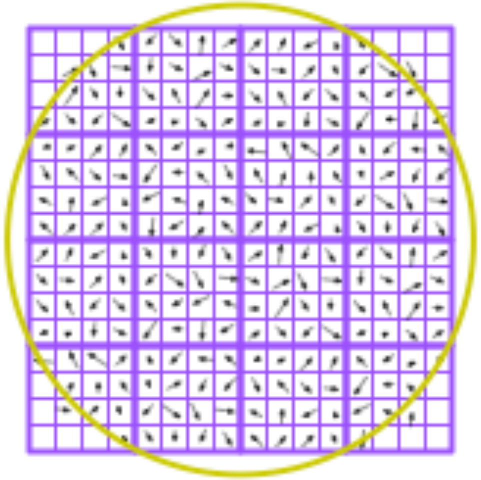
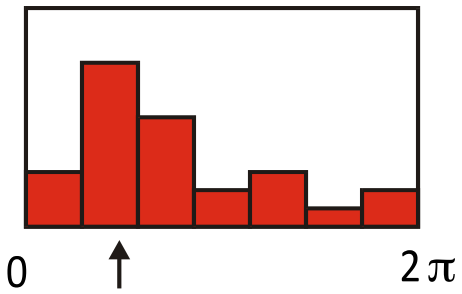


Image gradients

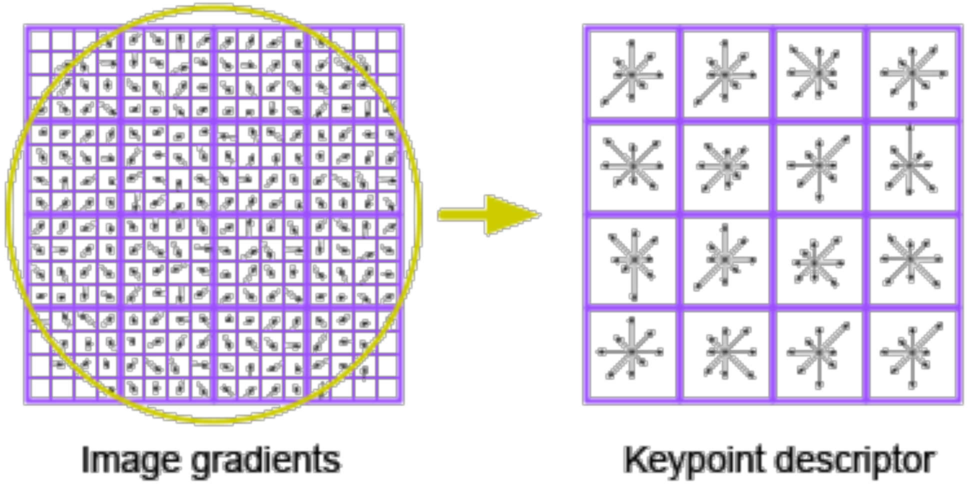
- Peaks in histogram: dominant orientations
- **Keypoint orientation = histogram peak**
- If there are multiple candidate peaks, construct a different keypoint for each such orientation



SIFT features | descriptor

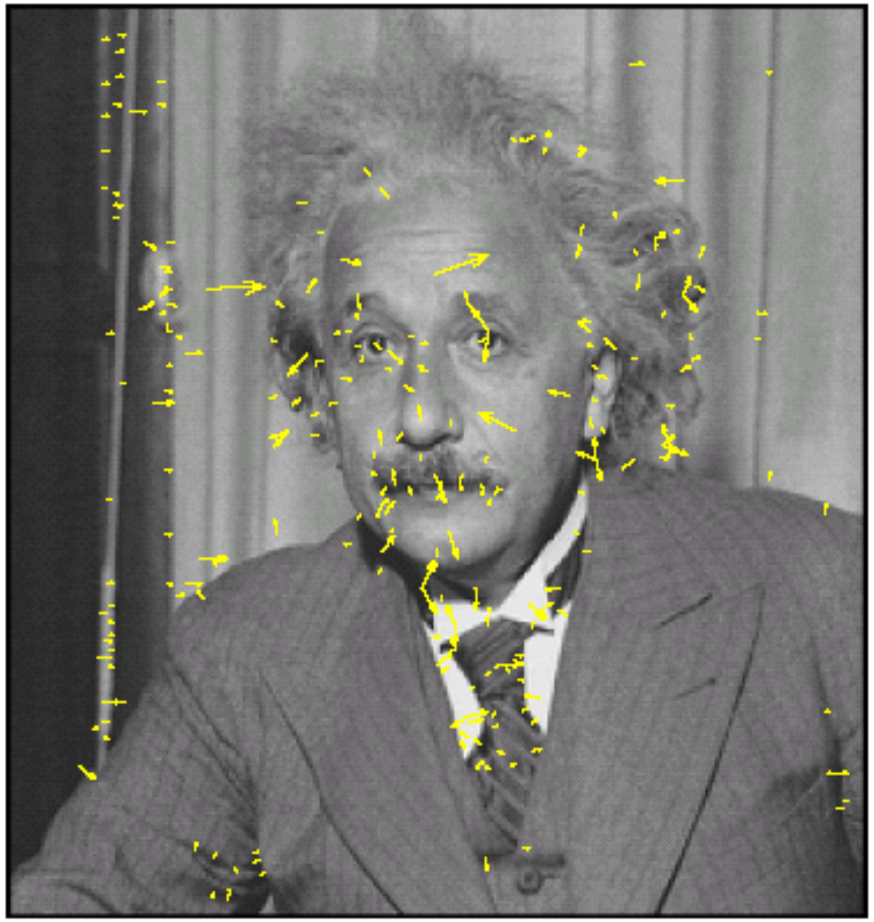
- **Descriptor** : "identity card" of keypoint
- Simplest descriptor: matrix of intensity values around a keypoint (image patch)
- Ideally, a descriptor should be
  - highly distinctive +
  - tolerant/invariant to common image transformations

- **SIFT descriptor**: 128-element vector
- Describe all gradient orientations **relative to the keypoint orientation**
- Divide keypoint neighborhood in **4×4** regions and compute orientation histograms along **8** directions
- SIFT descriptor: concatenation of all **4×4×8 (=128)** values
- Descriptor Matching: L_2-distance (i.e. SSD) between these descriptor vectors

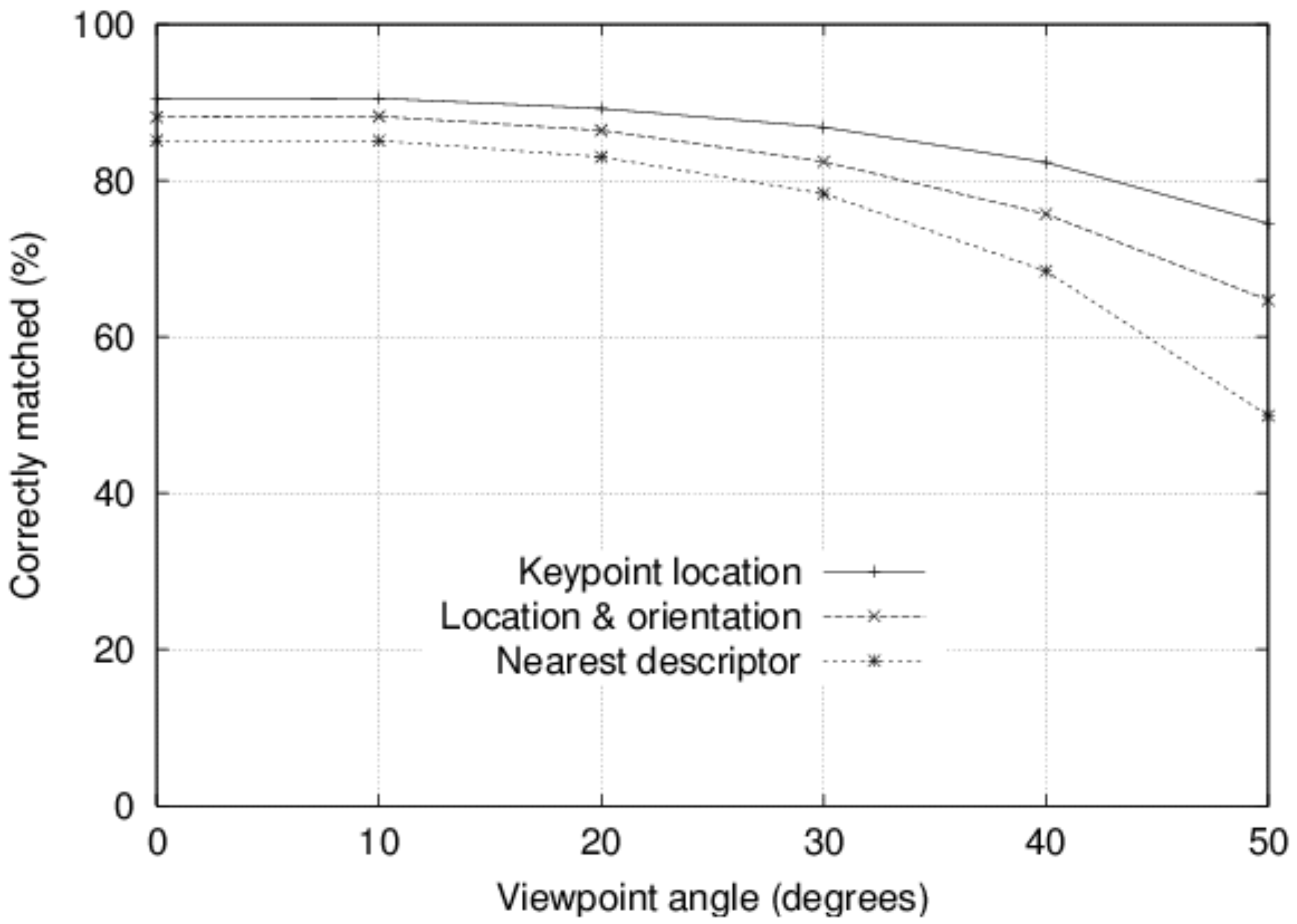


SIFT features

- Final SIFT keypoints with detected orientation & scale

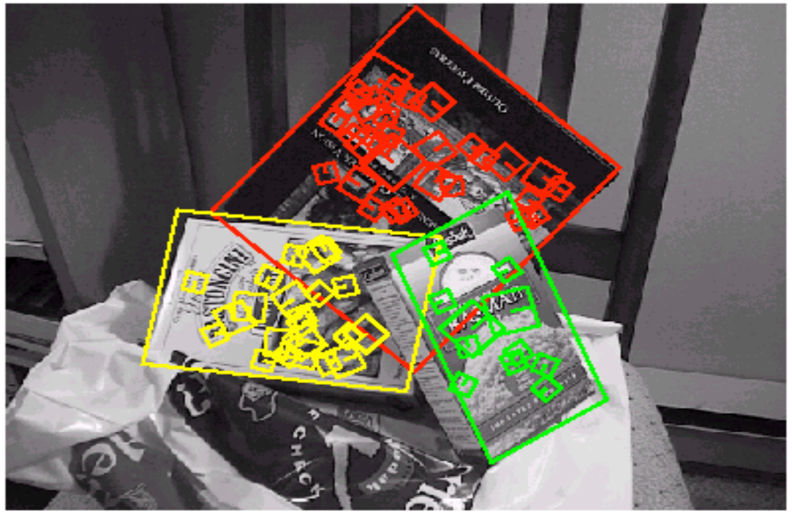


SIFT features | features' stability to viewpoint change



SIFT features | use in planar recognition

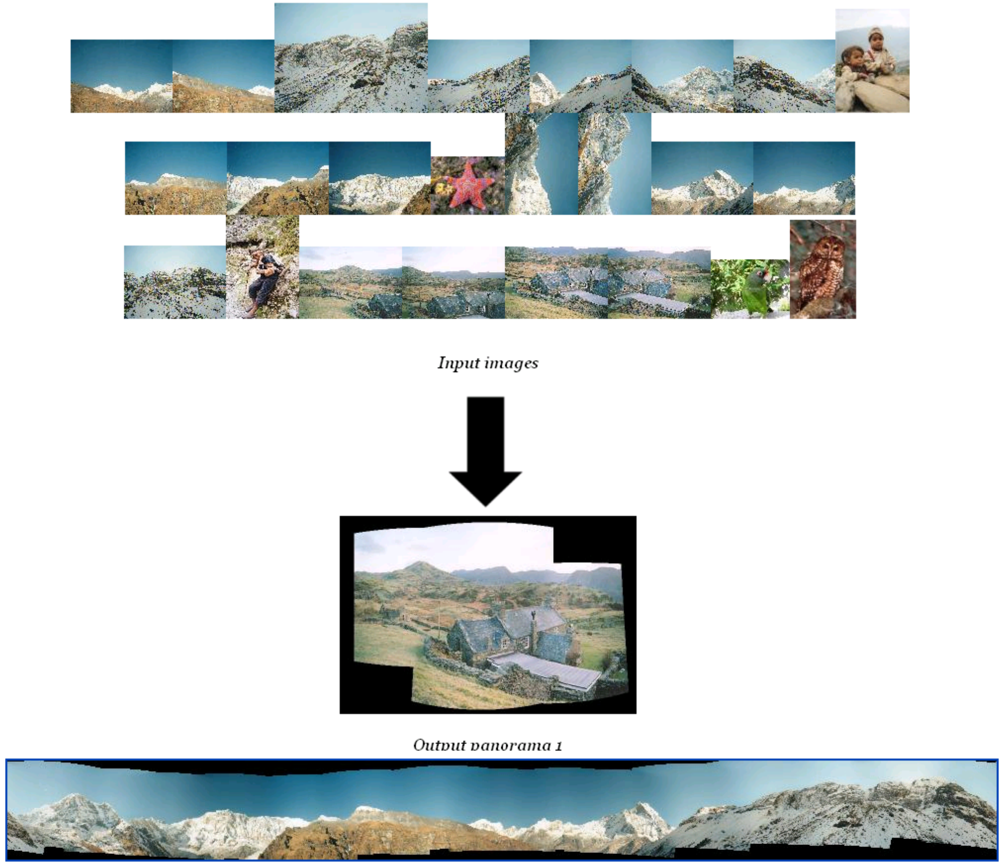
- **Planar** surfaces can be reliably recognized at a rotation of **60°** away from the camera
- Only 3 points are needed for recognition
- But objects need to have enough **texture**
- Recognition under occlusion



SIFT features | code and demos

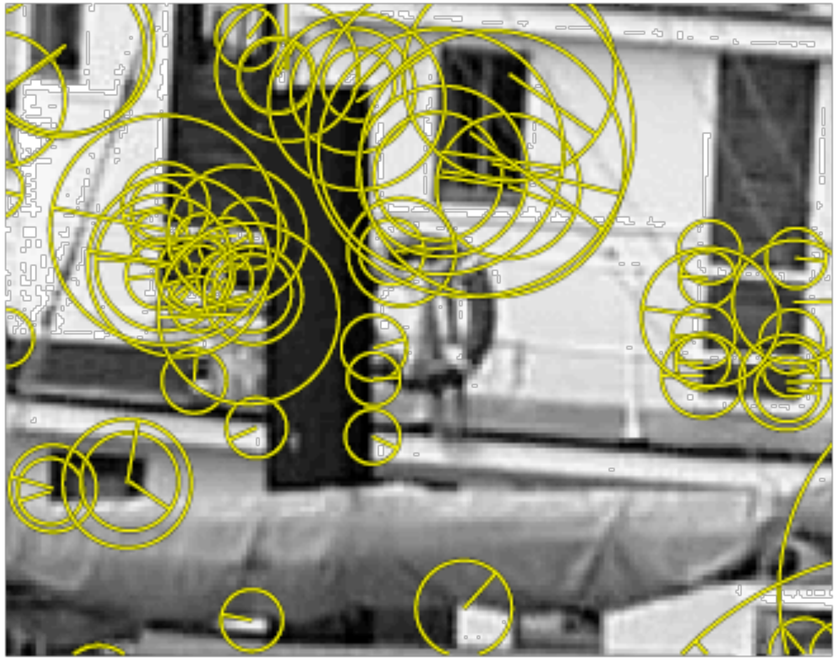
- **SIFT feature detector code:**
  for Matlab & C code to run with compiled binaries for Win and Linux (freeware)
  http://www.cs.ubc.ca/~lowe/keypoints/

- Make your own panorama with **AUTOSTITCH** (freeware):
  http://matthewalunbrown.com/autostitch/autostitch.html



More recent features from SOTA

- …suitable for Robotics applications



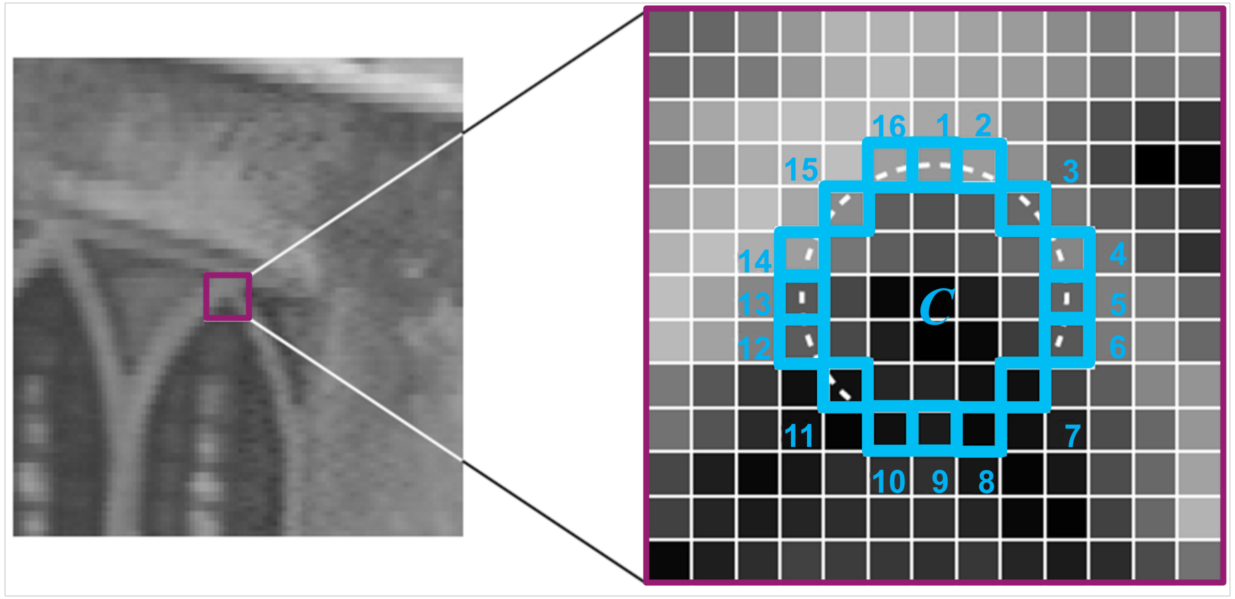
(a) Boat image 1

(b) Boat image 2

FAST corner detector [Rosten et al., PAMI 2010]

- **FAST**: **F**eatures from **A**ccelerated **S**egment **T**est
- Studies intensity of pixels on circle around candidate pixel C

- C is a FAST corner **if** a set of N contiguous pixels on circle are:
  - all brighter than `intensity_of(C)+theshold`, or
  - all darker than `intensity_of(C)+theshold`



- Typical FAST mask: test for **12** contiguous pixels in a **16**-pixel circle
- **Very fast detector** – in the order of 100 Mega-pixels/second

BRIEF descriptor [Calonder et. al, ECCV 2010]

- **BRIEF : B**inary **R**obust **I**ndependent **E**lementary **F**eatures
- Goal: high speed (in description and matching)

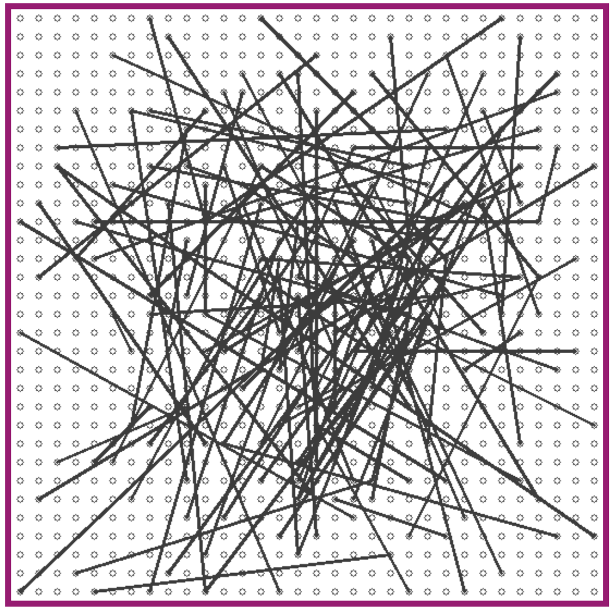
- **Binary** descriptor formation:
  - Smooth image
  - **for each** detected keypoint (e.g. FAST),

    sample all intensity pairs (I_1, I_2) (typically 256 pairs) according to pattern around the keypoint

    for each intensity pair **p**

    - **if** $I_1 < I_2$ **then** ***set*** bit **p** of descriptor to **1**
    - **else** ***set*** bit **p** of descriptor to **0**

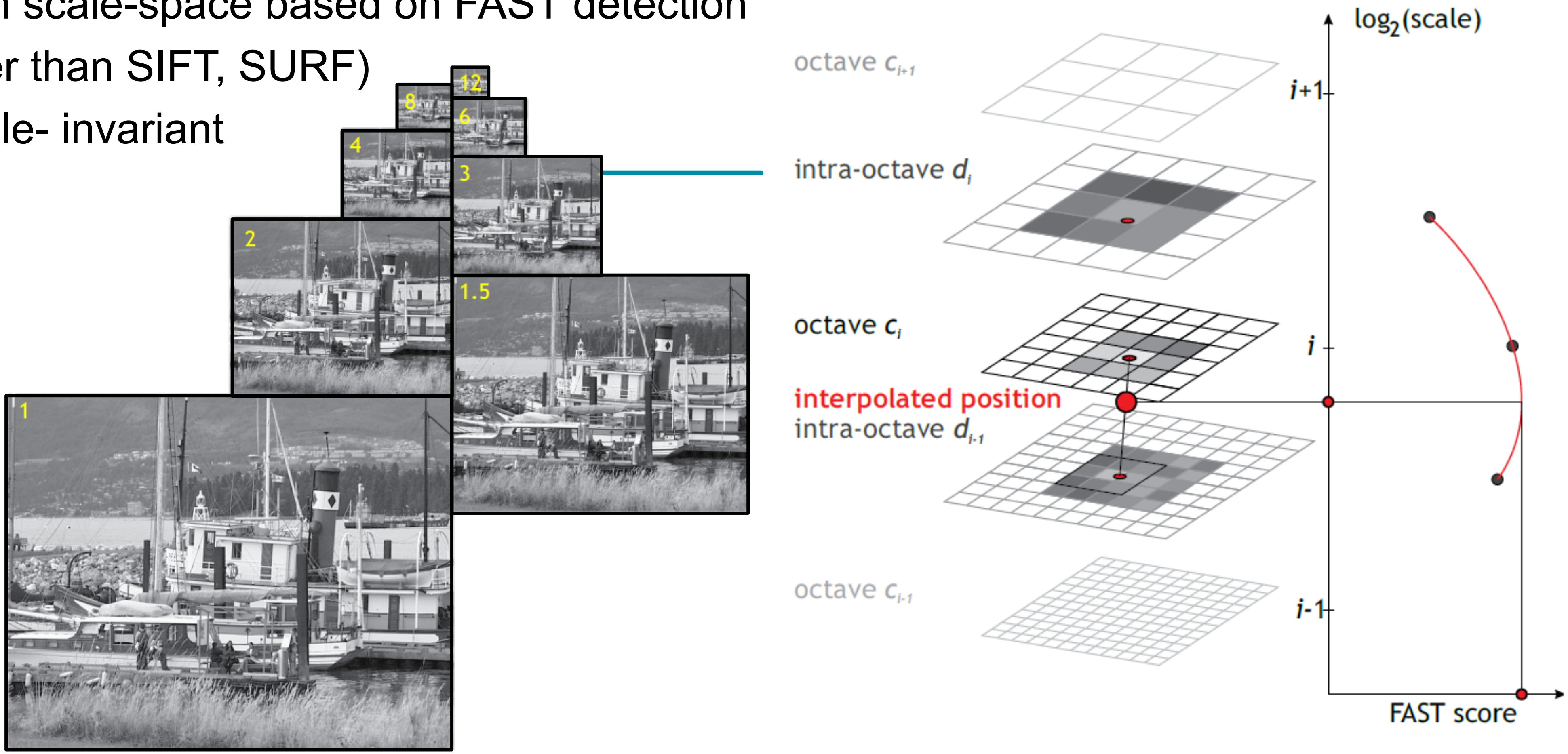
- Not scale/rotation invariant (extensions exist…)
- Allows **very fast** Hamming Distance matching: counting the no. different bits in the descriptors tested



Pattern for intensity pair samples – generated randomly

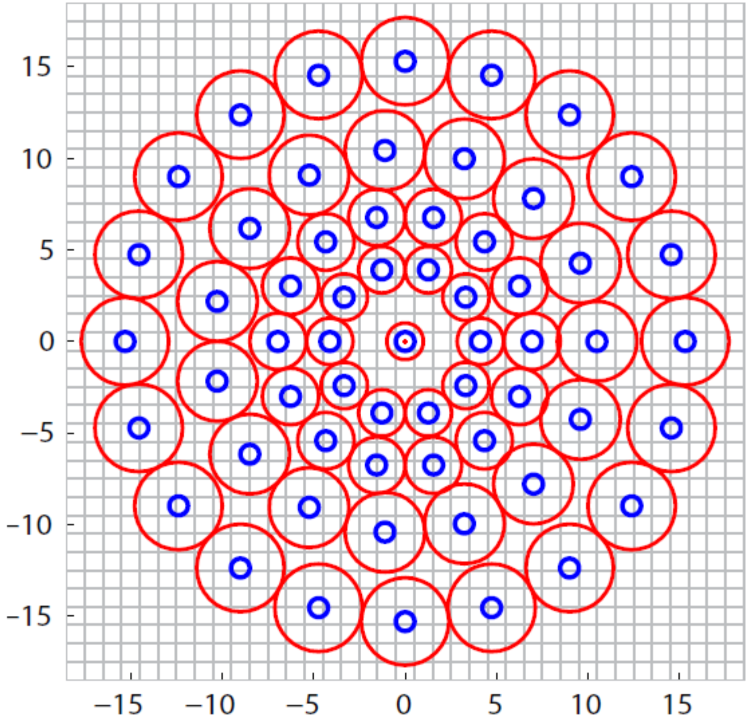
BRISK features [Leutenegger, Chli, Siegwart, ICCV, 2011] | detector

- **BRISK**: **B**inary **R**obust **I**nvariant **S**calable **K**eypoints
- Detects corners in scale-space based on FAST detection
- High-speed (faster than SIFT, SURF)
- Rotation- and scale- invariant



BRISK features | descriptor

- **Binary**, formed by pairwise intensity comparisons (like BRIEF)
- **Pattern** defines intensity comparisons in the keypoint neighborhood
- Red circles: size of the smoothing kernel applied
- Blue circles: smoothed pixel value used
- Compare short- and long-distance pairs for orientation assignment & descriptor formation
- Detection and descriptor speed: ≈10 times faster than SURF (and even faster than SIFT)
- Slower than BRIEF, but scale- and rotation- invariant



BRISK sampling pattern

BRISK feature | in action

Open-source code for FAST, BRIEF, BRISK and many more, available at the OpenCV library