Fundamentals of Computer Vision

Unit 6: Feature Extraction

Jorge Bernal

Index

01

1. Introduction

02

2. Corner detectors

03

3. Edge detectors

04

4. Blob detectors

1

Introduction

Introduction

Definition of feature:

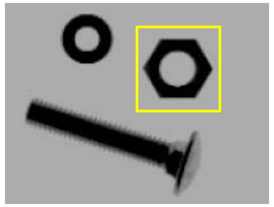
- Piece of information that is useful to solve a given task
- Interesting part of the image

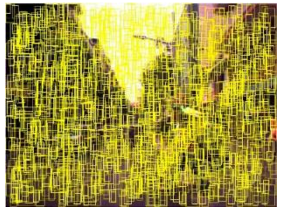
Types of features:

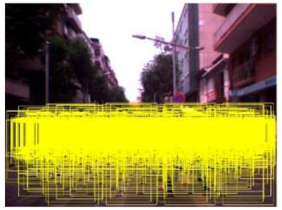
- **Global:** global properties of the whole image
    - Mean grey level, mean colour, main colours, histogram
- **Local:** properties of a part of the image with their own entity
    - Points, edges, regions

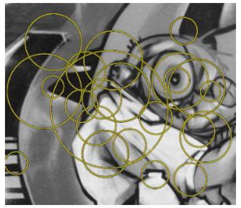
Introduction

GLOBAL **LOCAL**









Introduction

- **Local features:**
  - Part of an image that differs from its surroundings.
  - They are associated to a change in a certain property (intensity, color, texture)
  - Examples:
    - Points (corners, interest points)
    - Edges, ridges
    - Small regions (blobs)

Introduction

- How can we find local features?
  - Feature detection/extraction algorithms

- Detection/Extraction: locate the position of the feature
- Description (Unit 7): measures that are taken from the detected feature that allow us to distinguish it or compare with others

Introduction

- Why do we use features?
  - They have been used with success in several disciplines and applications:
    - Edge detection associated to roads in aerial images
    - Quality control
    - Polyp Detection
  - Interest points play a key role for certain Applications:
    - Tracking
    - 3D reconstruction
  - They are a first step to achive a robust image representation:
    - Object recognition
    - Scene classification
    - Texture analysis
    - Image search

Introduction

- Ideal properties:
  - Repeteability:
    - Invariance to transformations
    - Robustness
  - Differentiation (higly different from another)
  - Precise localization
  - Enough points for the needed task
  - Efficient
- Scale: very important factor to achieve robustness, invariance and precision. Allows us to work with different images at several distances.

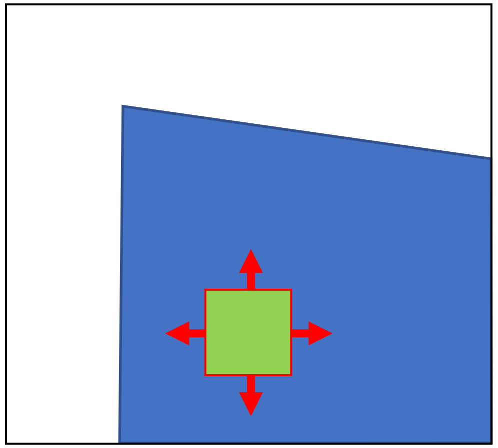
Introduction

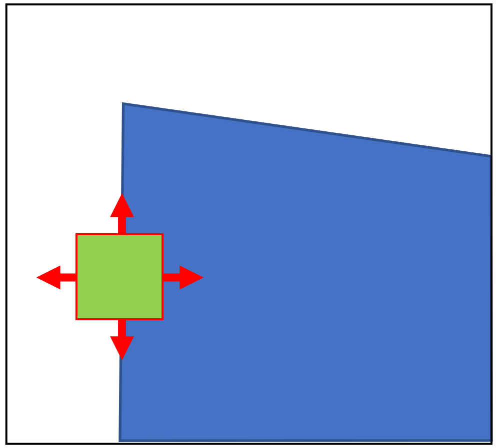
Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	√			√			+++	+++	+++	++
Hessian		√		√			++	++	++	+
SUSAN	√			√			++	++	++	+++
Harris-Laplace	√	(√)		√	√		+++	+++	++	+
Hessian-Laplace	(√)	√		√	√		+++	+++	+++	+
DoG	(√)	√		√	√		++	++	++	++
SURF	(√)	√		√	√		++	++	++	+++
Harris-Affine	√	(√)		√	√	√	+++	+++	++	++
Hessian-Affine	(√)	√		√	√	√	+++	+++	+++	++
Salient Regions	(√)	√		√	√	(√)	+	+	++	+
Edge-based	√			√	√	√	+++	+++	+	+
MSER			√	√	√	√	+++	+++	++	+++
Intensity-based			√	√	√	√	++	++	++	++
Superpixels			√	√	(√)	(√)	+	+	+	+

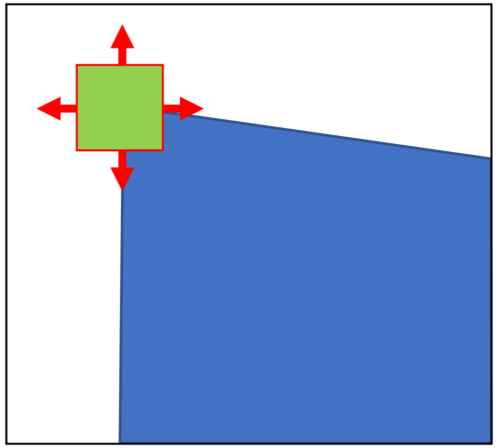
2

Corner Detection

Corner Detection







Plain region
No changes in all directions

Edge
No change in edge direction

Corner
Significant changes in all directions

Corner Detection

- Harris (1988): Based on the analysis of the 2D structural tensor (second derivative matrix, second moment matrix)
- SUSAN (Smallest Univalue Segment Assimilating Nucleos): morphologic focus
- Harris-Laplace: Use of Harris for a first detection; scale is fixed using laplacian
- Harris-Affine: Use of Harris-Laplace; then it tries to estimate the most affine shape (with an ellipse that is later normalized to a circle)

Corner Detection: Harris

- In an image intensity corner, intensity changes significantly in all directions.
- Here we are focused in intensity changes in a local window.
- We use SSD: sum of squared differences

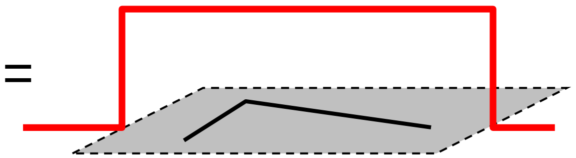
$$S(x,y) = \sum_u \sum_v w(u,v)\ \left(I(u+x, v+y) - I(u,v)\right)^2$$

window

Shifted intensities

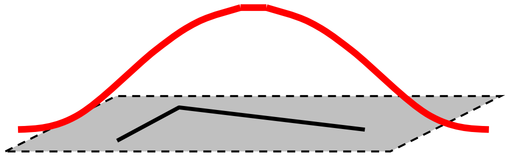
intensity

Window function *w(u,v)* =



1 inside , 0 outside

o



gaussian

Corner Detection: Harris

Shifted intensity is approximated using a Taylor Expansion:

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

So, at the end:

$$S(x,y) \approx \sum_u \sum_v w(u,v) \, \left(I_x(u,v)x + I_y(u,v)y\right)^2,$$

We can write this in matrix format as:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix},$$

, where A is the 2D structural tensor

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

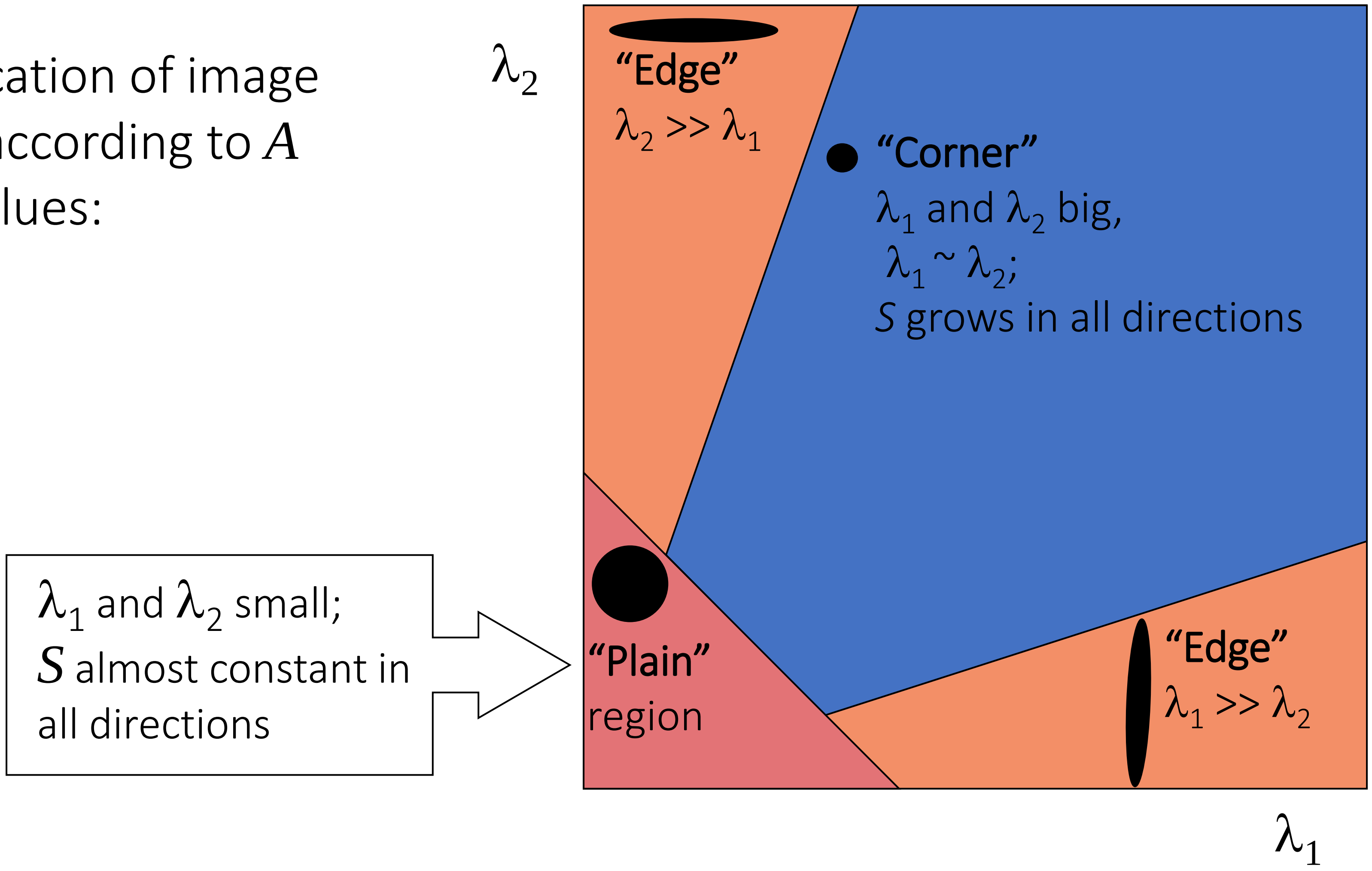
Corner Detection: Harris

We change the problem of examining intensity changes due to traslations to analyze the behaviour of matrix A → analysis of eigenvalues

$$\lambda_1, \lambda_2 \quad \text{eigenvalues of } A$$

Corner Detection: Harris

Classification of image points according to A eigenvalues:



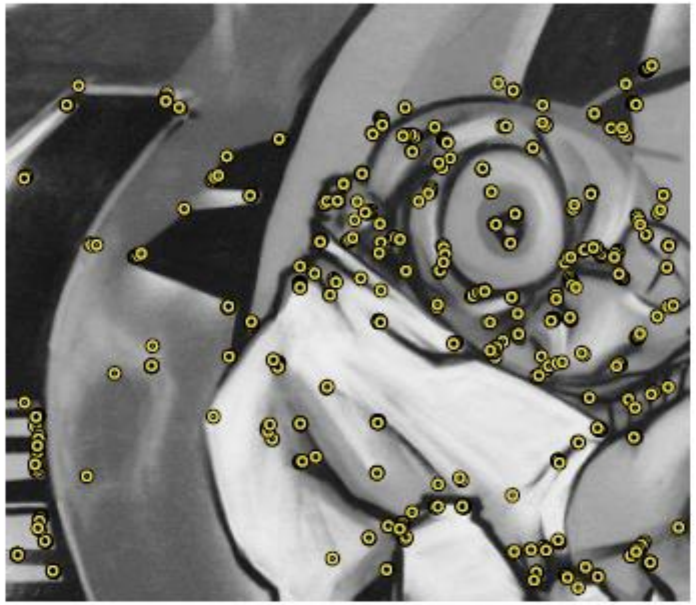
Corner Detection: Harris

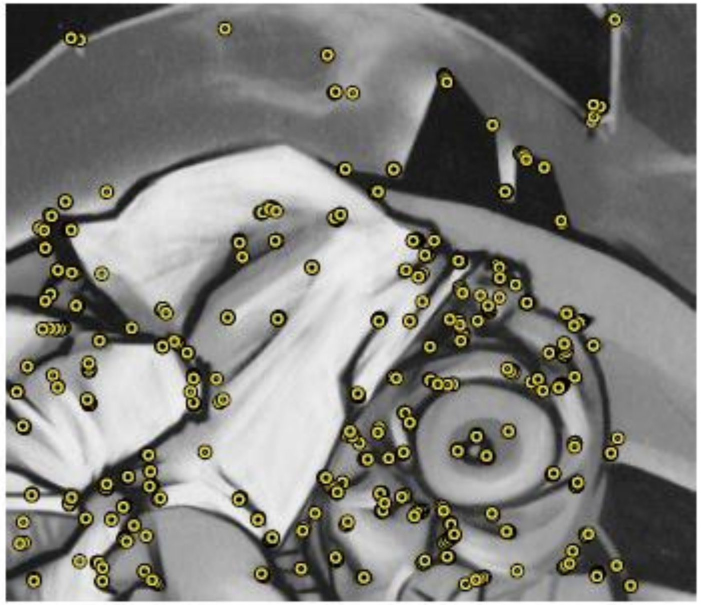
Response function at corners (R):

$$R = \det(A) - k\,(\operatorname{trace} A)^2$$

$$R = \lambda_1\lambda_2 - k\,(\lambda_1 + \lambda_2)^2$$

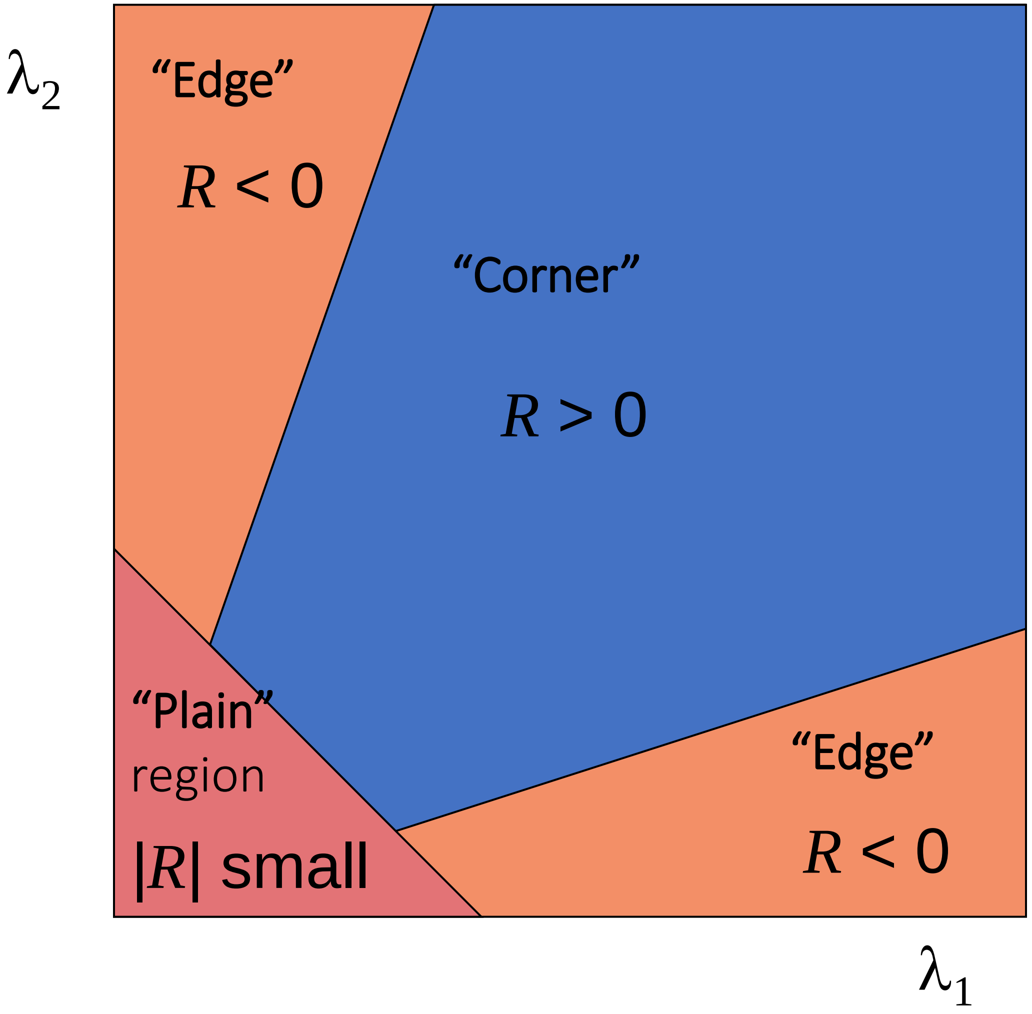
where k is a constant value (empiric) $k = [0.04, 0.06]$





Corner Detection: Harris

- R depends only of A eigenvalues
- R is big at corners
- R is negative with high value at edges
- $|R|$ is small at plain regions



Corner Detection: Harris

- First derivatives at an image point(u,v):

$$I_x(u,v) = \frac{\partial I}{\partial x}(u,v)$$
$$I_y(u,v) = \frac{\partial I}{\partial y}(u,v)$$

- We can compute:

$$A(u,v) = I_x^2(u,v),$$
$$B(u,v) = I_y^2(u,v),$$
$$C(u,v) = I_x(u,v) \cdot I_y(u,v)$$

- Local structre matrix (M) [a.k.a. A]

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$$

- Smoothing with a gaussian (G)

$$\overline{M} = \begin{pmatrix} A*G & C*G \\ C*G & B*G \end{pmatrix} = \begin{pmatrix} \overline{A} & \overline{C} \\ \overline{C} & \overline{B} \end{pmatrix}$$

Corner Detection: Harris

- Diagonal of $\overline{M}$

$$\overline{M} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

- Where $\lambda_{1,}$ λ_2 are the eigenvalues of $\overline{M}$ defined by:

$$\frac{1}{2}\left(\overline{A} + \overline{B} \pm \sqrt{\overline{A}^2 - 2\overline{A}\overline{B} + \overline{B}^2 + 4\overline{C}^2}\right)$$

- Describes a point according to eigenvalues, using corners response function

$$R = \lambda_1 \lambda_2 - k\left(\lambda_1 + \lambda_2\right)^2$$

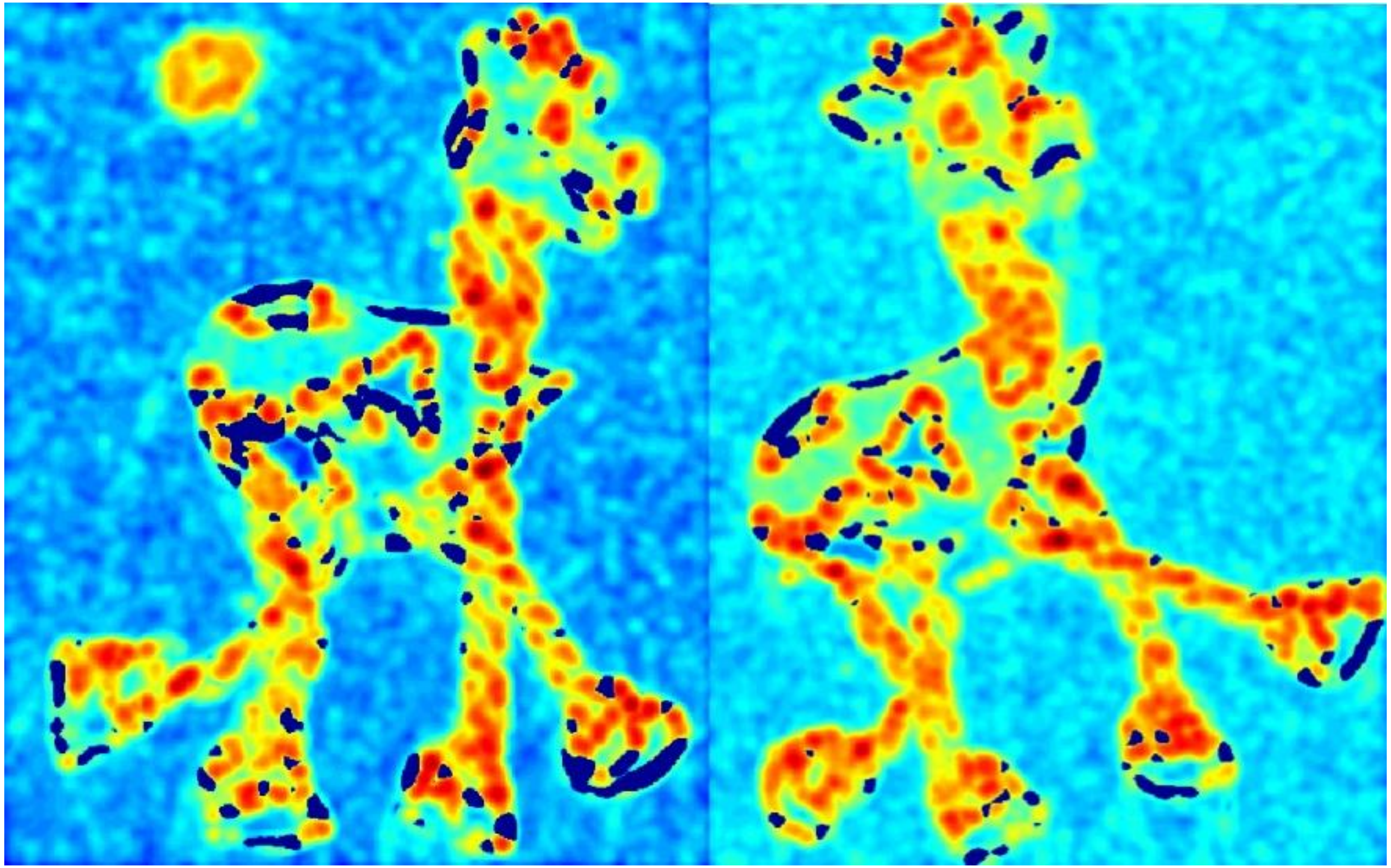
- A good corner has big changes of intensity in all directions → R should be big and positive.

Corner Detection: Harris

Original

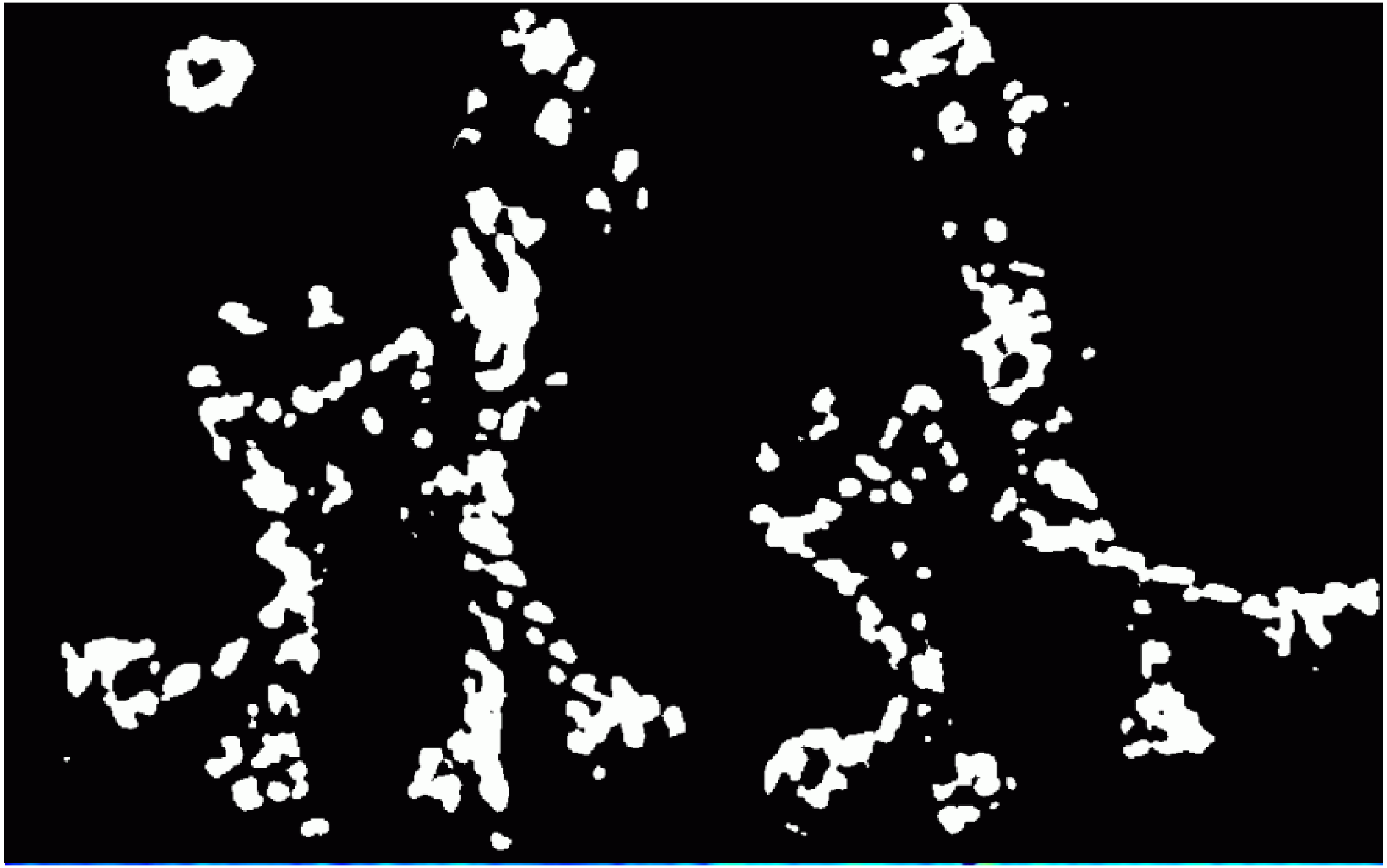
Corner Detection: Harris

R



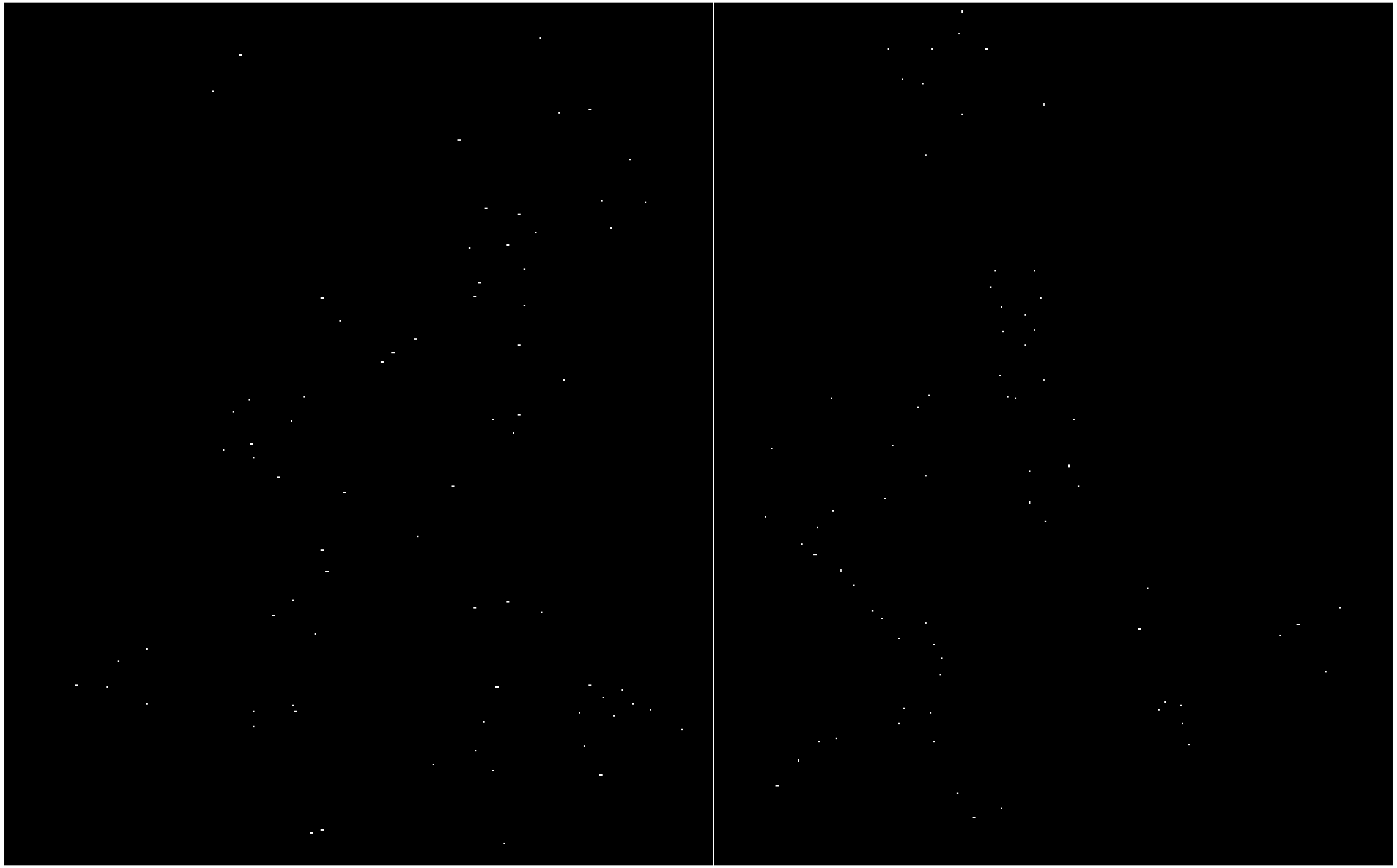
Corner Detection: Harris

Points with R > threshold



Corner Detection: Harris

R local maxima

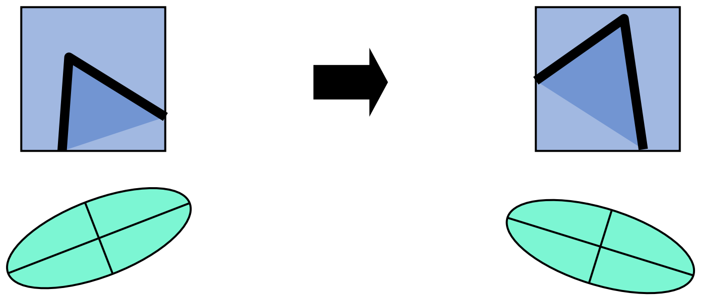


Corner Detection: Harris

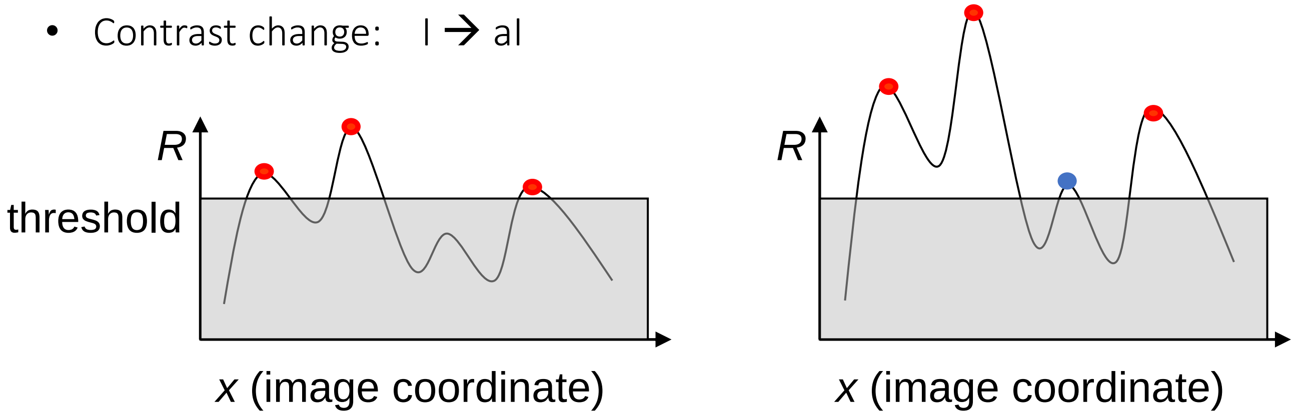
Final result

Corner Detection: Harris-Laplace

- Properties:
  - Rotation invariant:

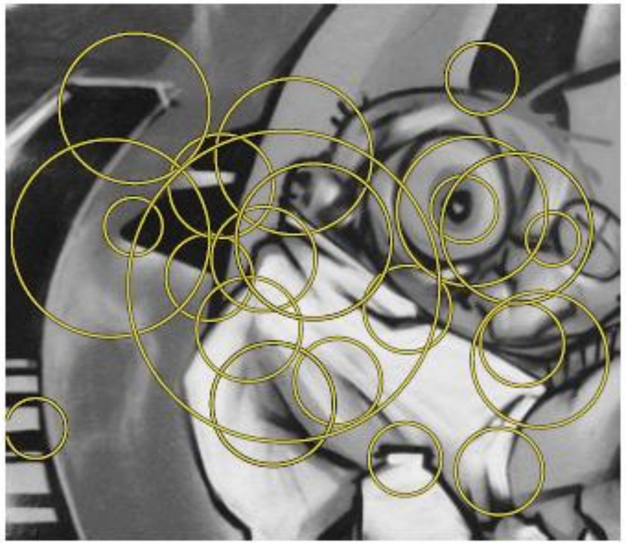


  - Partial invariance to affine intensity changes (derivatives):
    - Invariance to shifts in intensity I → I+b
    - Contrast change: I → aI



Corner Detection: Harris-Laplace

- Combines Harris with a gaussian scale-space.
- We use gaussian Windows with predetermined scales
- We choose the scale that maximizes LoG in this range



- We obtain both the corners and the scale in which it is better represented

Corner Detection: Harris-Affine

- Initial detection using Harris-Laplace
- Affine shape estimated using 2D structure matrix
- Normalize affine regions to a circular shape
- Detect new corner position and scales in the previous image
- If eigenvalues change, go back to point 2

Harris Corner Detector in OpenCV

```
import numpy as np
import cv2 as cv
filename = 'chessboard.png'
img = cv.imread(filename)
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)
dst = cv.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]
cv.imshow('dst',img)
if cv.waitKey(0) & 0xff == 27:
    cv.destroyAllWindows()
```

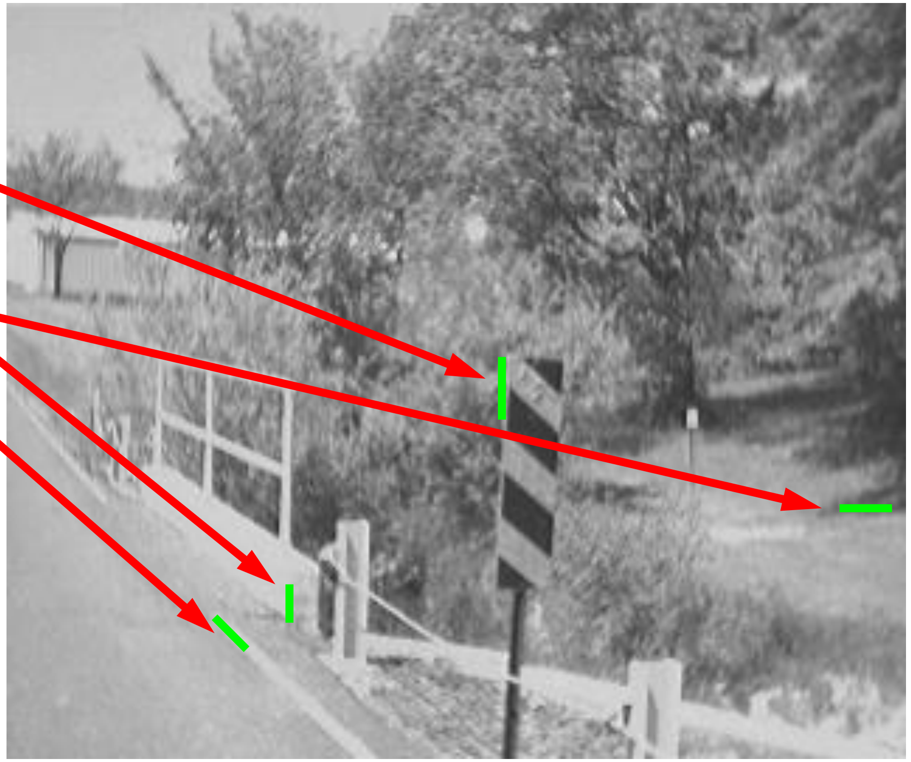
3

Edge Detection

Edge Detection

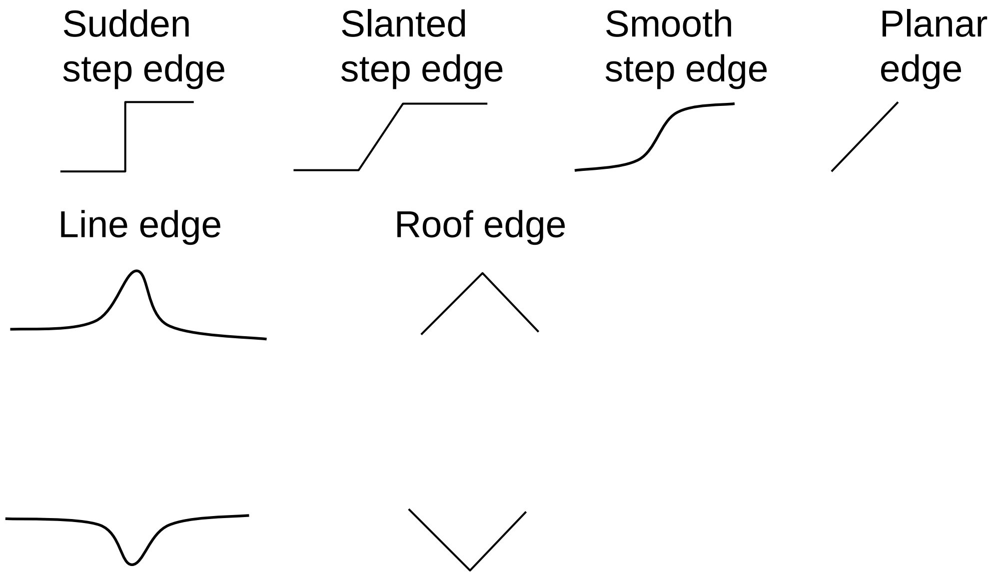
Why do contours appear in images?

- Change in Depth
- Change in Orientation
- Change in Reflectance
- Change in Illumination



Edge Detection

- Boundaries (edges)
  - Image regions where gradient magnitude has maximum value
- Valleys / Creasts (ridges)
  - Curve that represents a local maxima or mínima
- Models
  - Step
  - Creast
  - Valley



Edge Detection

- Gradient
  - Vector that points in the direction of the highest change

$$\text{grad}(I) = \nabla(I) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) = (I_x, I_y)$$

  - We can calculate its magnitude and orientation

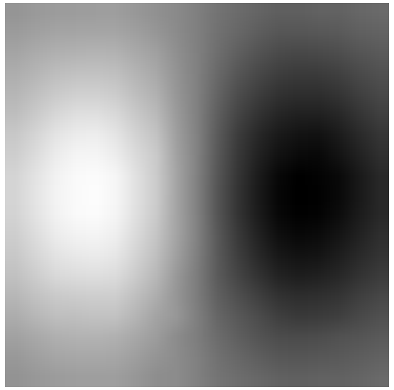
$$|\nabla| = \sqrt{I_x^{\;2} + I_y^{\;2}}$$

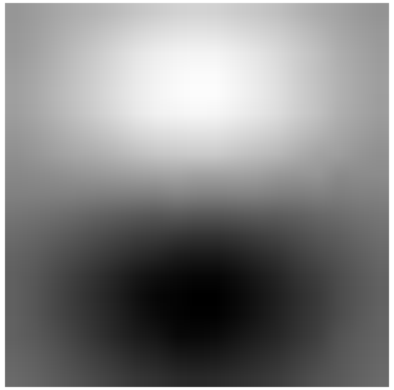
$$\theta = \arctan(I_y / I_x)$$

  - Boundaries are associated to high magnitude grdients

Edge Detection

- Smoothing / Regularization
  - Allows us to decrease noise and control analysis scale
  - First derivative increases noise. We can smooth before derivating (regularization)
  - Smoothing can be done using a Gaussian with good properties (certain frequencies are not amplified)
    - We can also derivate the convolution with the derivative of the gaussian





Edge Detection

- Algorithms
  - Differential gradient operator
    - Roberts
    - Sobel
    - Prewitt
  - Laplacian of Gaussian
  - Canny

Edge Detection

$$Prewitt(im) = \left(im * \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, im * \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \right)$$

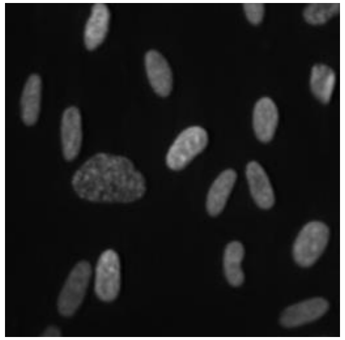
$$Sobel(im) = \left(im * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, im * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \right)$$

$$Roberts(im) = \left(im * \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, im * \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \right)$$

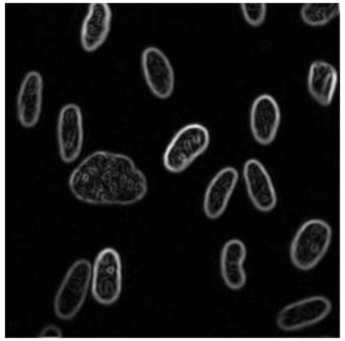
Edge Detection

$$Sobel(im) = \sqrt{\left(im * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}\right)^2 + \left(im * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}\right)^2}$$

Original



Sobel



Original

Sobel

Edge Detection

- Laplacian

$$Laplacian(I) = \Delta(I) = \nabla^2(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

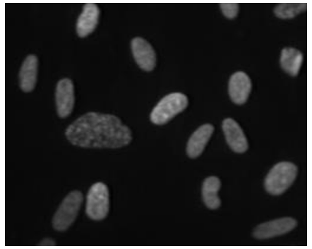
- Numerical approximation

$$Laplacian(im) = im * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

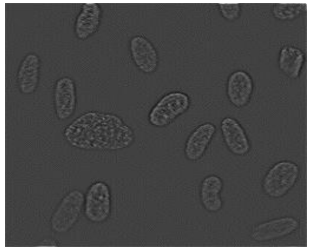
- Laplacian’s zero crossings provide us image boundaries

Edge Detection

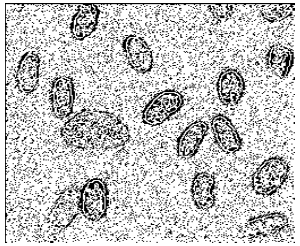
- Laplacian
  - Disadvantage: result is noisier



Original

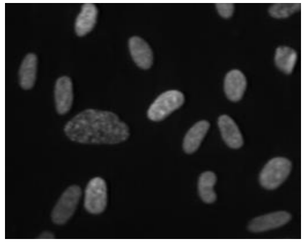


Laplacian

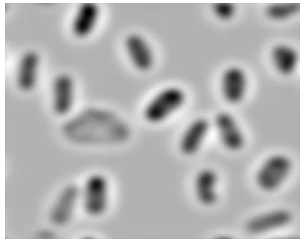


Crossings

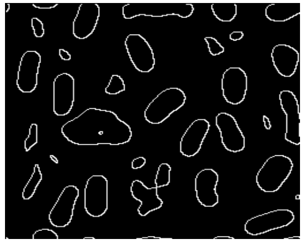
  - Solution: smoothing with a gaussian



Original



LoG



Crossings

  - Advantage: provides as result closed contours

Edge Detection

- Canny Edge Detector:
  - We calculate the gradient with gaussian derivatives
  - We apply non maximum suppression
    - Selection of a single entity out of many overlapping ones
  - Join and binarize
    - We define upper and lower thresholds
    - We accept all contours above the lower threshold that are connected to other boundaries above the upper threshold

Edge Detection

- Canny Edge Detector:

1.- Original

2.- Norm of the gradient

3.- Thresholding

4.- Thinning (non-maximum suppression)

Edge Detection

- Canny Edge Detector:
  - Scale

original

low σ

high σ

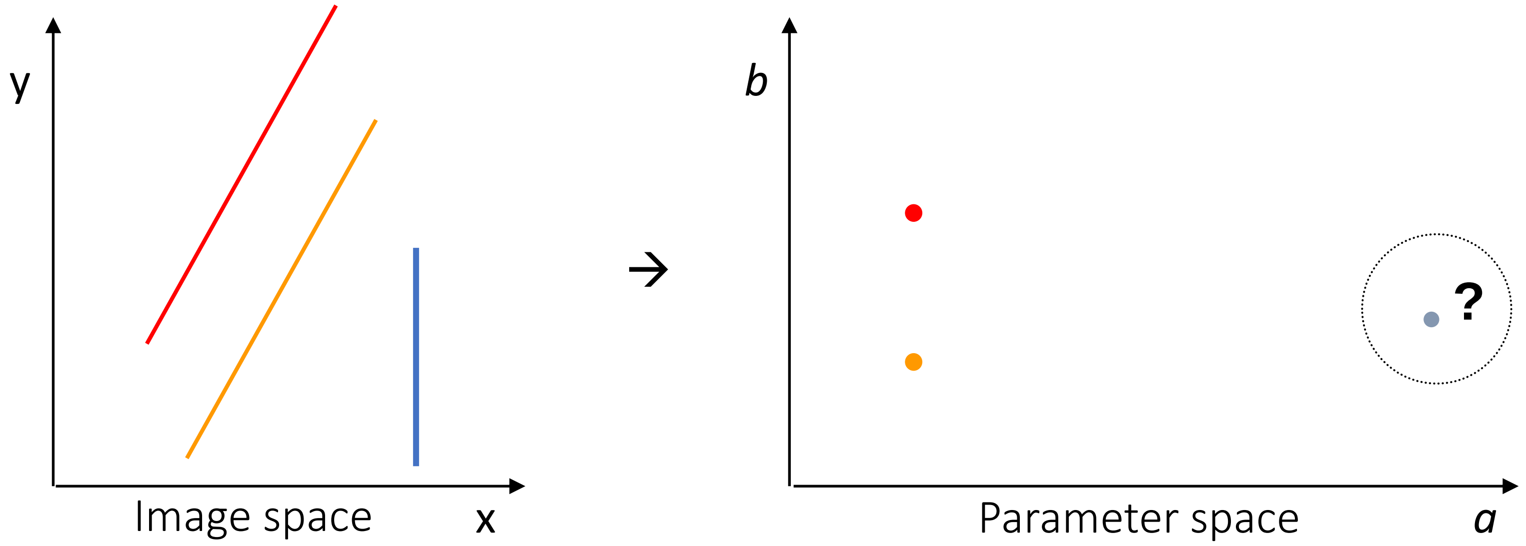
  - Choice σ of depends of the desired behaviour
    - High σ detects high scale boundaries
    - Low σ detects low scale boundaries (noisier appearance)

Edge Detection

- Grouping:
  - Primitive detection from boundary parts or a set of points
    - Hough Transform for lines (SLHT)
    - Hough Transform for circles (CHT)
    - Generalized Hough Transform (GHT)

Edge Detection

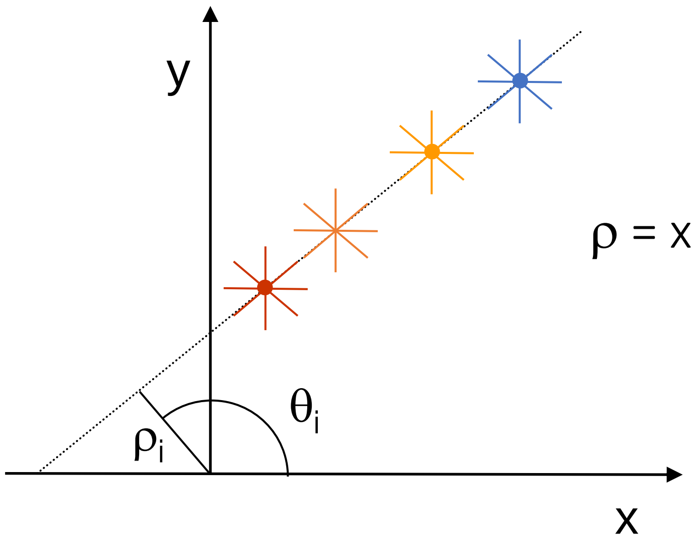
- Hough Transform for lines
  - Transform points associated to a pattern within a parameter space where they can be represented in a compact shape
  - Example for lines $y = ax + b$



  - Solution: lin → $\rho = x \sin\theta + y \cos\theta$

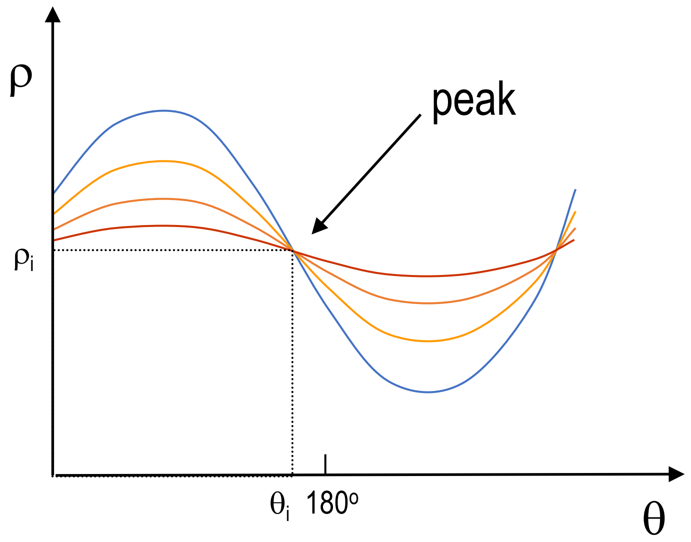
Edge Detection

- Straight Line Hough Transform
  - Key point detections: pixel selection according to local properties (gradient magnitude, orientation)
  - Transformation mapping: each keypoint is mapped in the feature space (accumulation or voting array)
  - Peak detection: local/global binarization in accumulation array



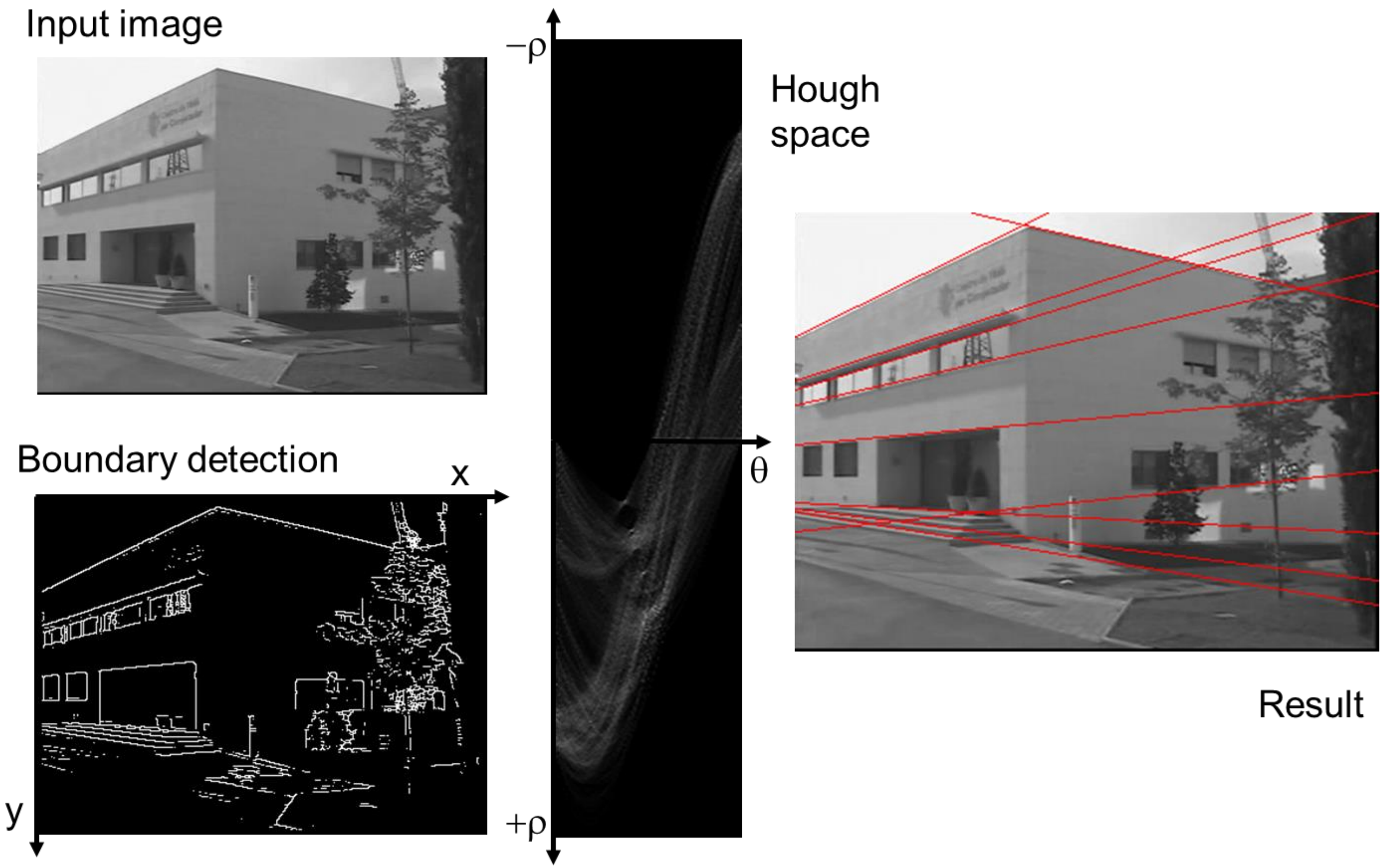
→

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



Edge Detection

- Example



Edge Detection

- Another Hough Transforms
  - Circles (CHT):
    - Tridimensional voting space (x,y,r)
    - Each point contributes to this voting space within a cone
  - General (GHT):
    - Model definition:
      - For an object (closed or open boundary) we define an inner center
      - For each boundary point we calculate the gradient (contour direction)
      - From the center to each point we calculate radii and angle
      - We store for each direction all radii-angle pairs
    - Voting
      - We generate image either of boundaries or from boundaries. We calculate gradients
      - For each point we vote all radii-angle associated to a particular direction

Edge Detection

- Python implementation
  - Canny
    - edge = cv2.Canny(image, low_th, high_th)
  - Sobel
    - edge = cv2.Sobel(image,precision_out_image,d_x,d_y)
      - d_x and d_y specify if the first derivative of a specific direction is computed
  - Laplacian
    - edge = cv.Laplacian(src_gray, precision_out_image, ksize)
      - ksize: kernel size of the Sobel operator to be applied internally (commonly 3)
  - Hough Transform lines
    - lines = cv.HoughLines(edges, rho, np.pi / 180, 150, None, 0, 0)
      - edges: output of edge detector
      - rho: resolution of the parameter r in pixels (commonly 1)
      - theta: resolution of the parameter θ in radians (commonly 1 degree, pi/180)
      - threshold: minimum number of intersections to detect a line
      - srn and stn: set to 0

4

Region Detectors

Region Detection

- Laplacian of Gaussian
- Blob detection in binary images
- Maximally Stable Extremal Region (MSER)

Region Detection

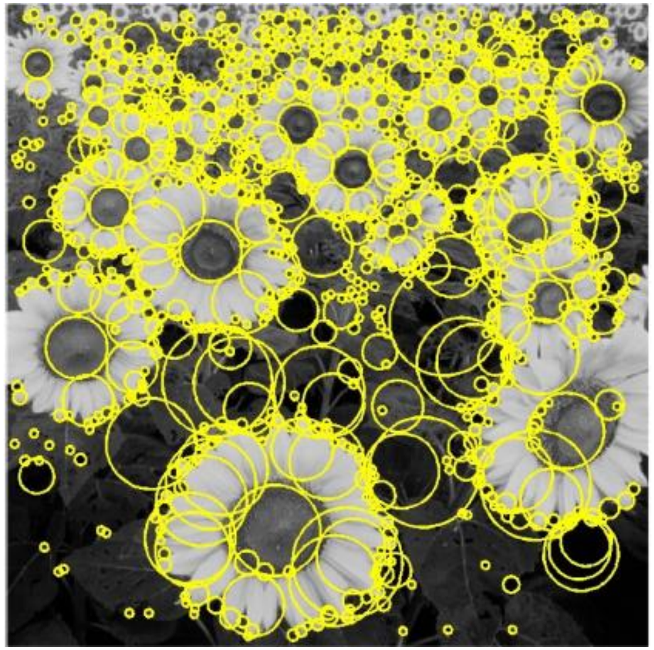
- Blobs
  - Regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions.
  - All the points in a blob can be considered in some sense to be similar to each other

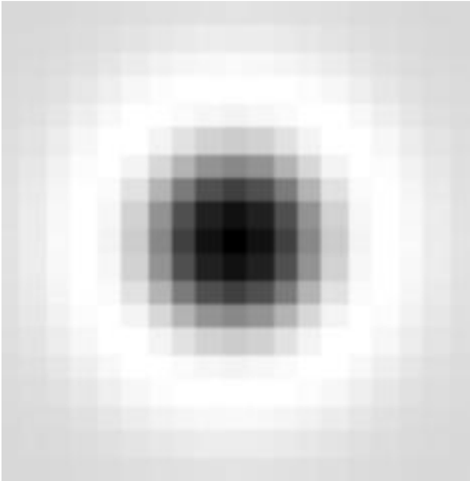
Region Detection

- Blobs
  - Two classes of blob detectors:
    - Differential: based on derivatives of the function with respect to position
    - Local-extrema based: finding the local maxima and minima of the function

Region Detection

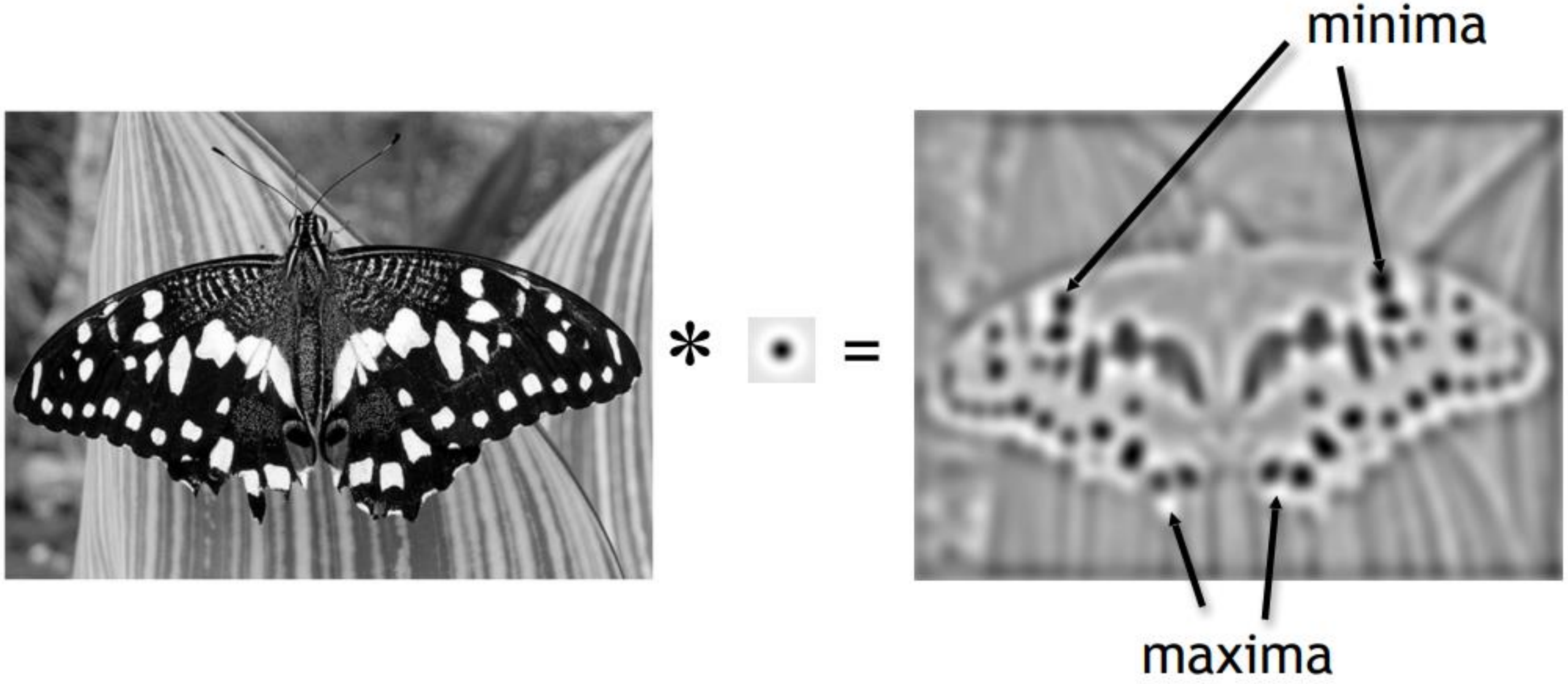
- Basic idea of Blob Detection: convolution of the image with a "blob filter" at multiple scales and look for extrema of filter response in the resulting *scale space*





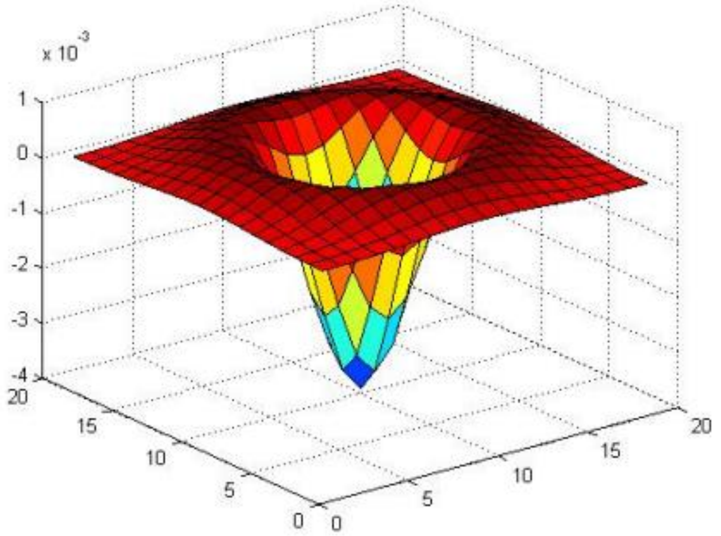
Region Detection

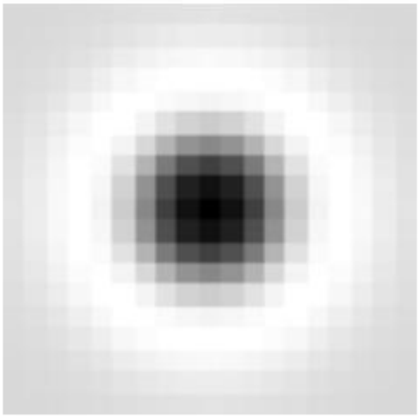
- Find maxima and minima of blob filter response in space and scale



Region Detection: Laplacian of Gaussian

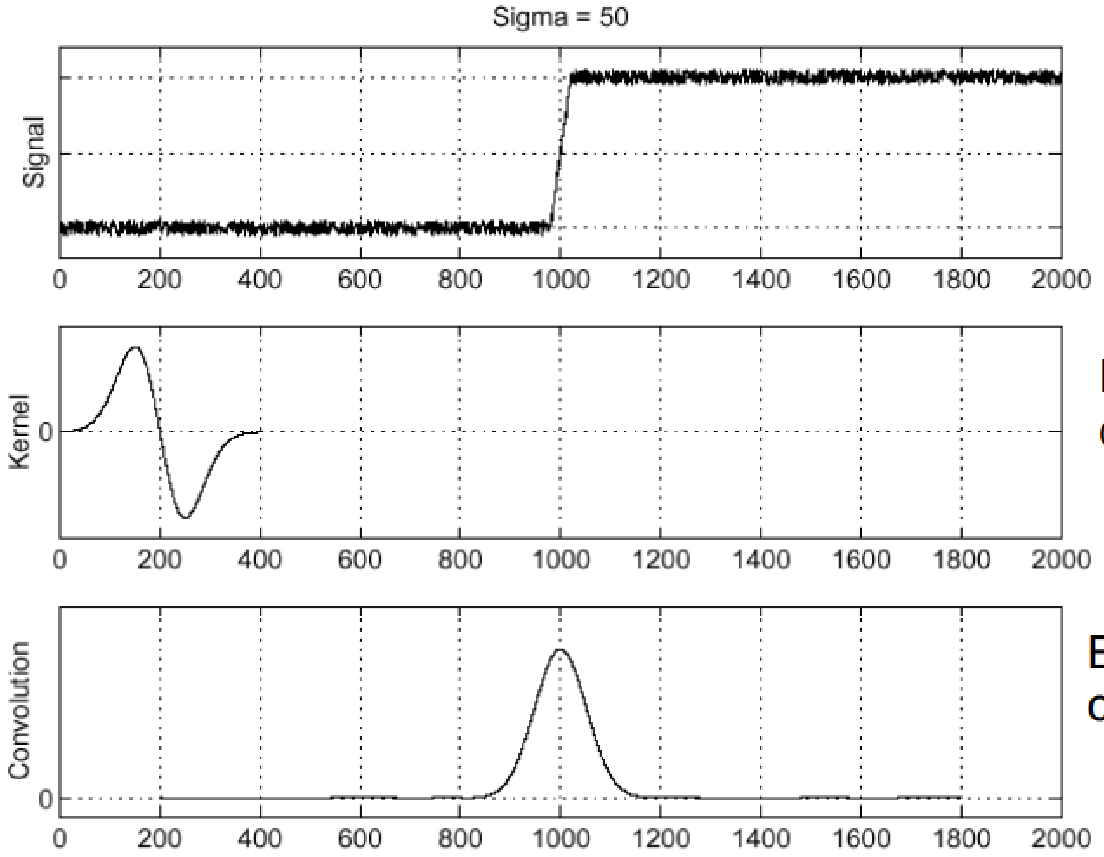
- Circularly symmetric operator for blob detection in 2D





$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Region Detection: Laplacian of Gaussian

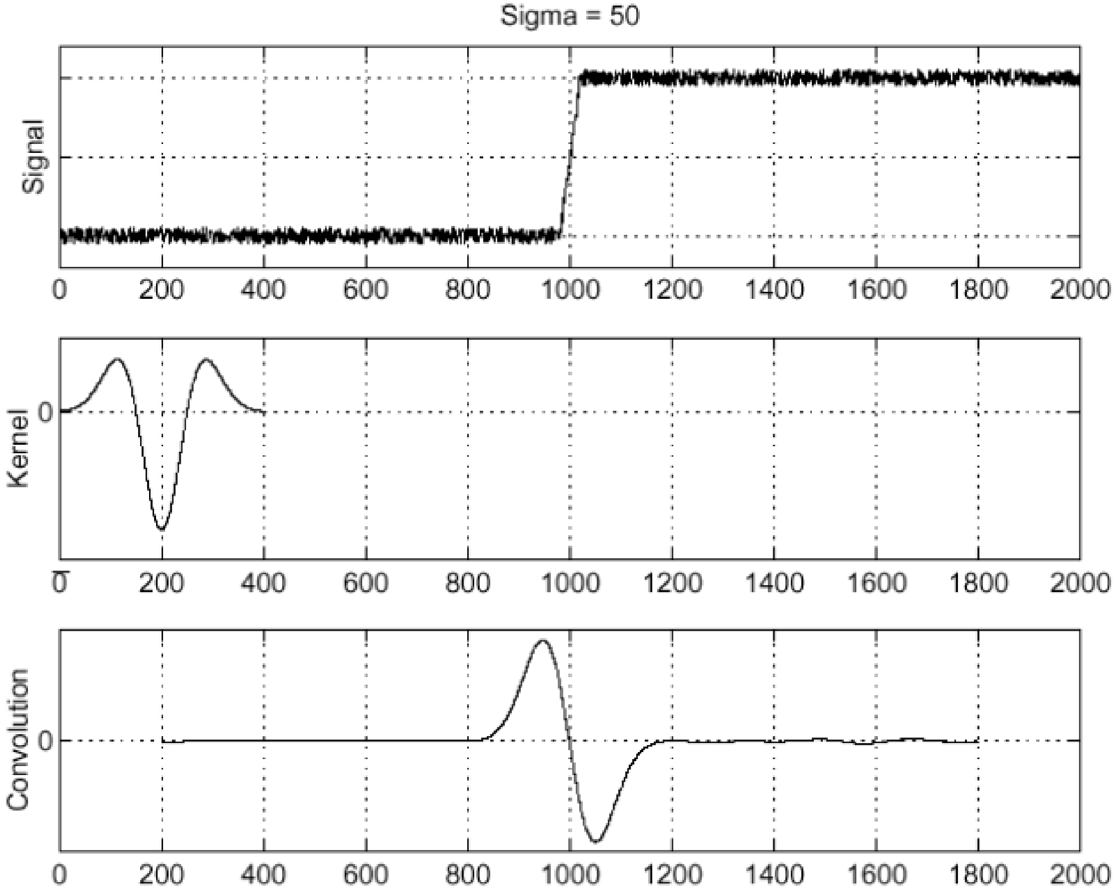


f — Edge

$\frac{d}{dx}g$ — Derivative of Gaussian

$f * \frac{d}{dx}g$ — Edge = maximum of derivative

Region Detection: Laplacian of Gaussian



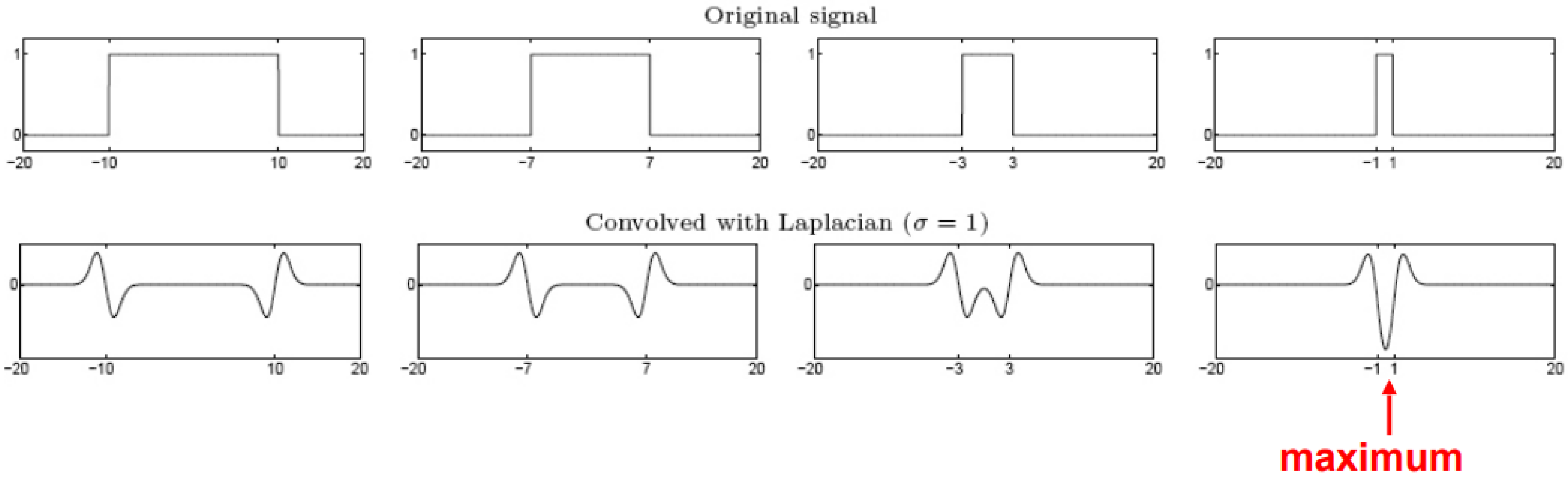
f — Edge

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

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

Region Detection: Laplacian of Gaussian

- Edge = ripple
- Blob = superposition of two ripples

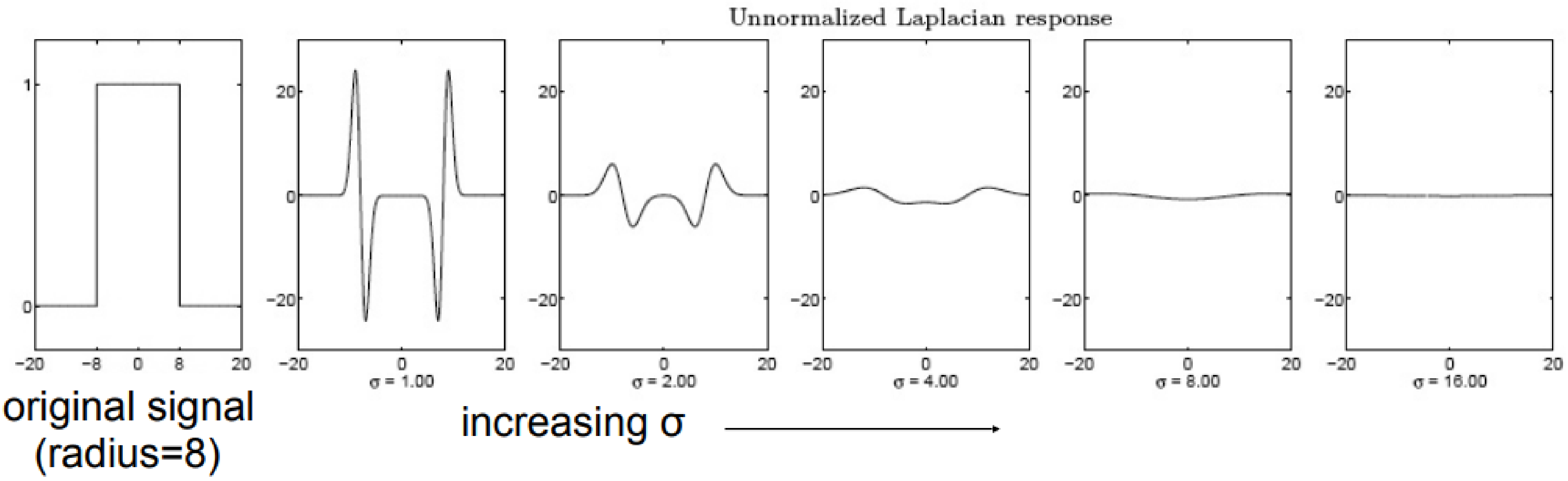


Spatial selection: the magnitude of the Laplacian response will achieve a maximum at the center of the blob, provided the scale of the Laplacian matches the scale of the blob

Region Detection: Laplacian of Gaussian

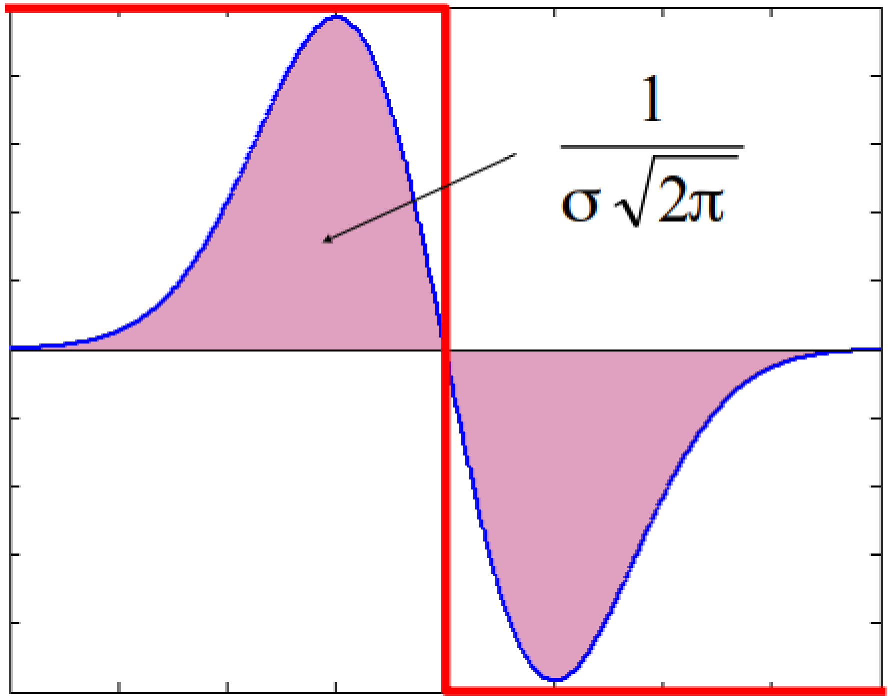
Scale selection: we want to find the characteristic scale of the blob by convolving it with Laplacians at several scales and looking for the maximum response

BUT Laplacian response decays as scale increases



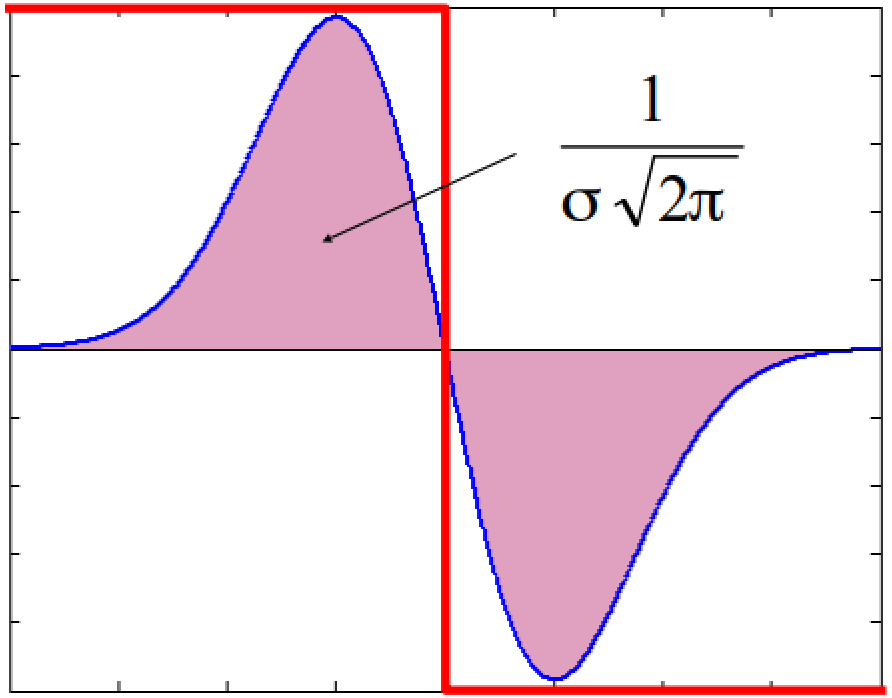
Region Detection: Laplacian of Gaussian

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



Region Detection: Laplacian of Gaussian

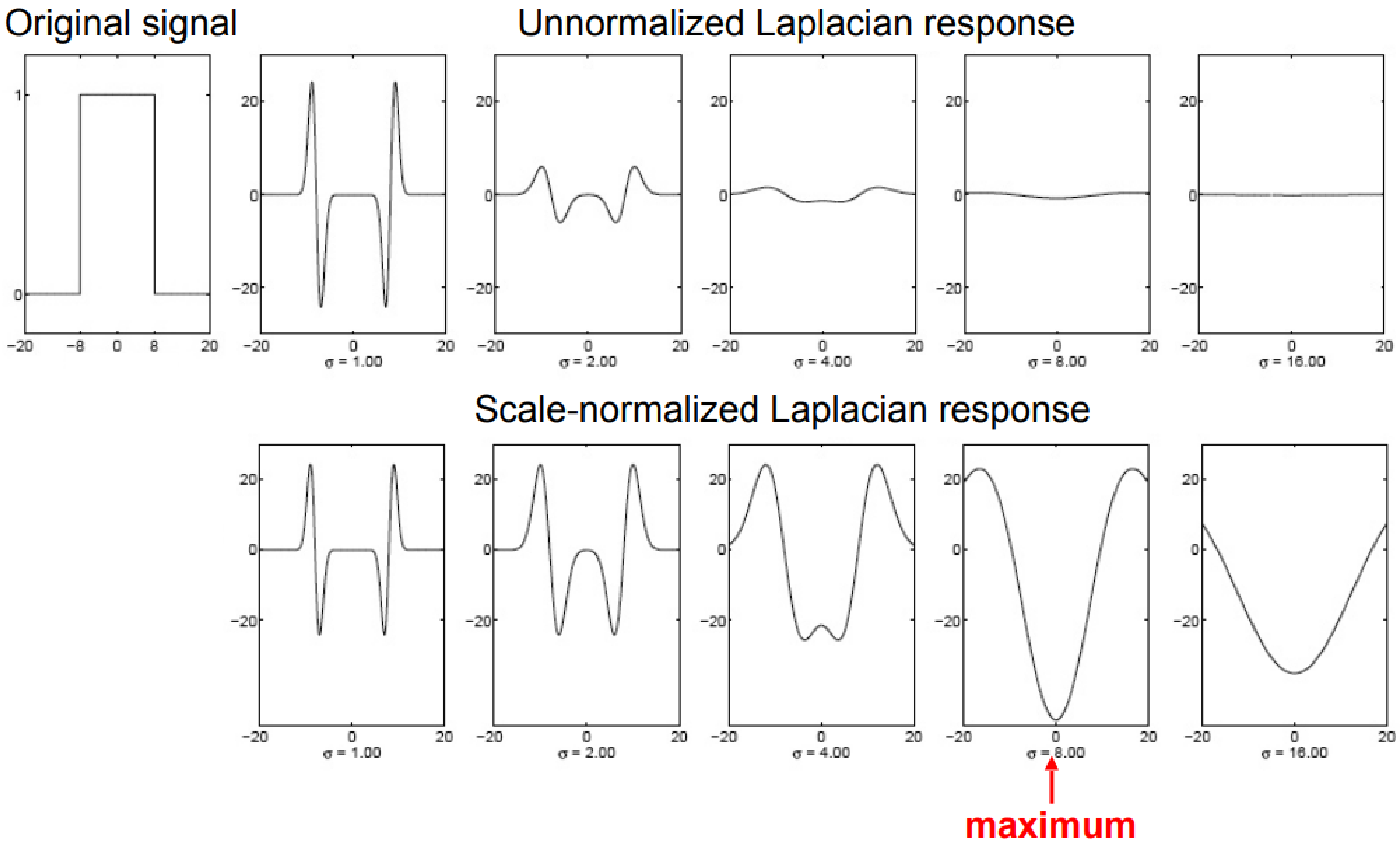
The response of a derivative of Gaussian filter to a perfect step Edge decreases as σ increases



How to make it scale-invariant?

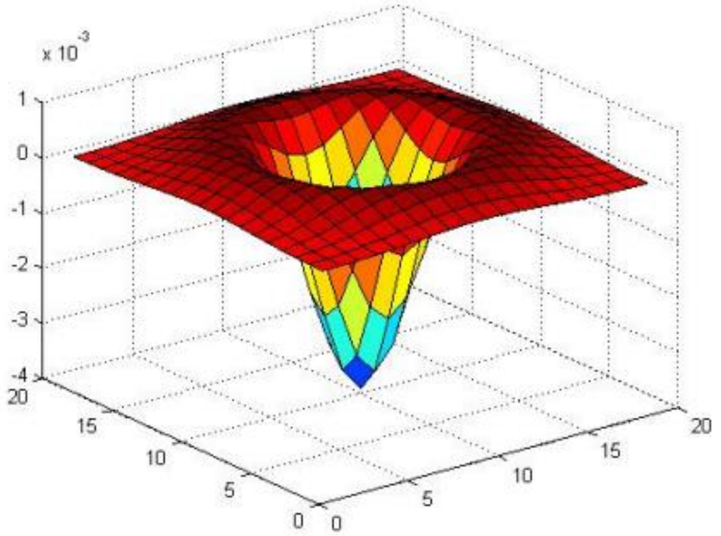
- Multiply Gaussian derivative by σ
- As Laplacian is the second Gaussian derivative, it should be multiplied by σ^2

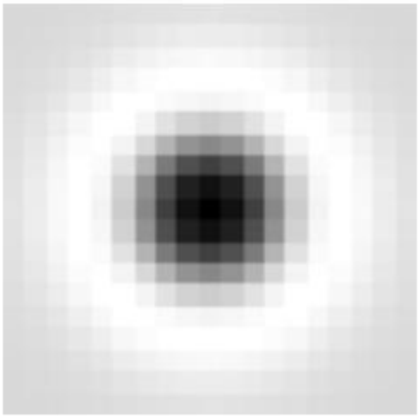
Region Detection: Laplacian of Gaussian



Region Detection: Laplacian of Gaussian

- Circularly symmetric operator for blob detection in 2D

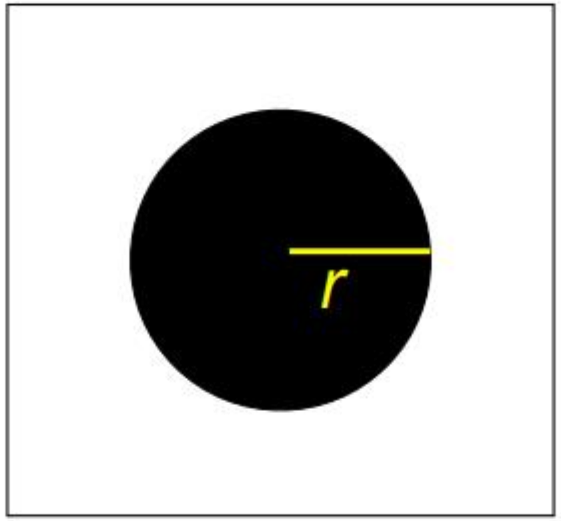




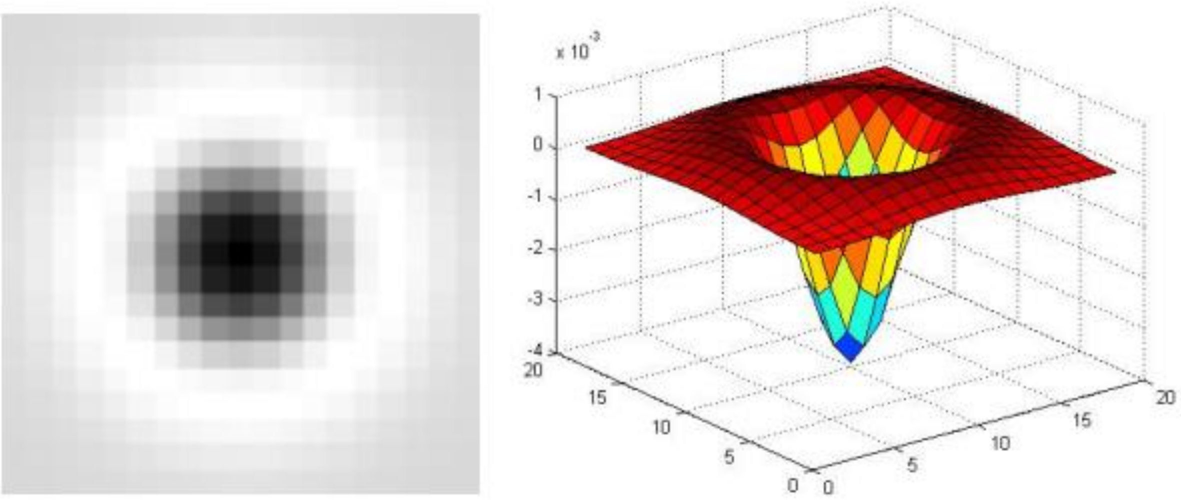
Scale-normalized: $$\nabla^2_{\text{norm}} g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

Region Detection: Laplacian of Gaussian

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



image



Laplacian

Region Detection: Laplacian of Gaussian

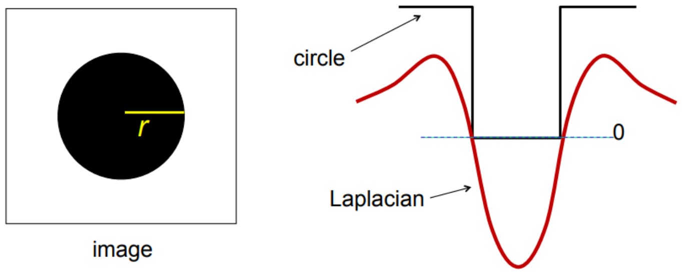
- At what scale does the Laplacian achieve a maximum response to a binary circle of radius r?

- For maximum response: align the zeros of the Laplacian with the circle

- The Laplacian in 2-D is given by (up to scale):

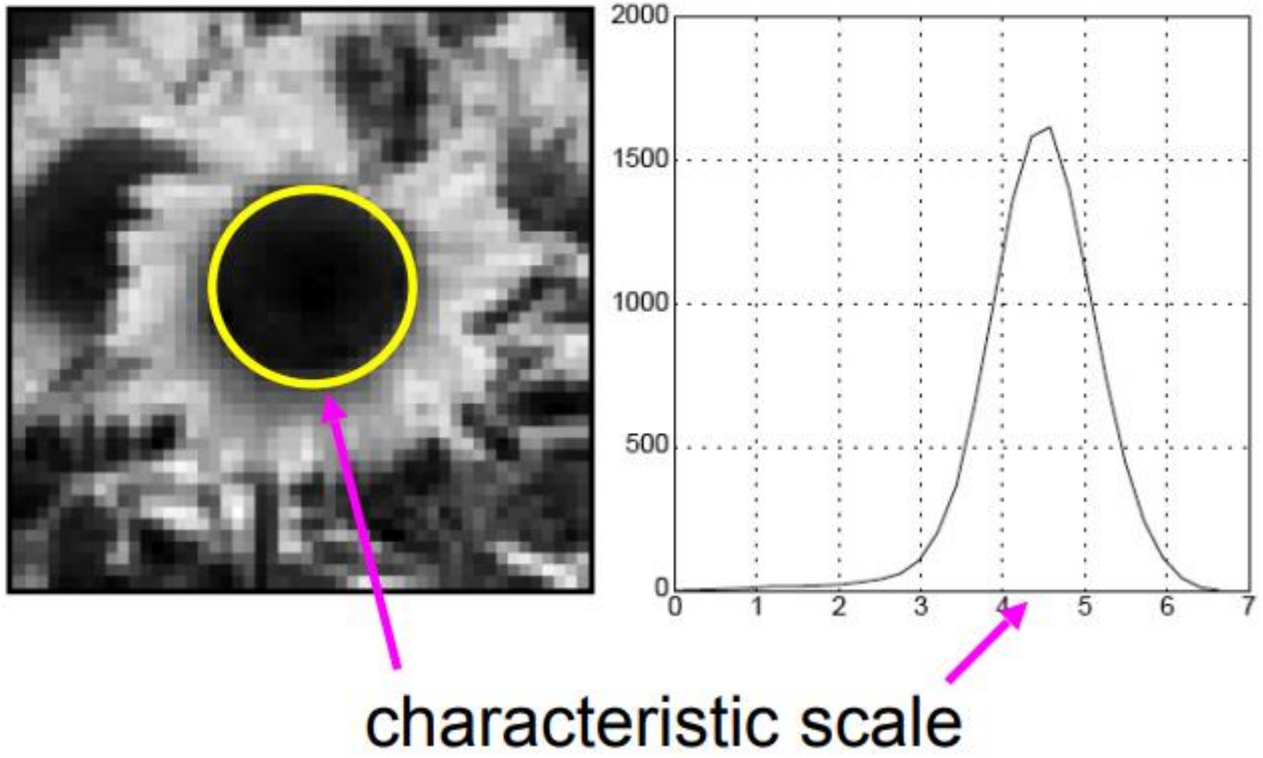
$$(x^2 + y^2 - 2\sigma^2)e^{-(x^2+y^2)/(2\sigma^2)}$$

- Therefore, the maximum response occurs at $\sigma = r/\sqrt{2}$.



Region Detection: Laplacian of Gaussian

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



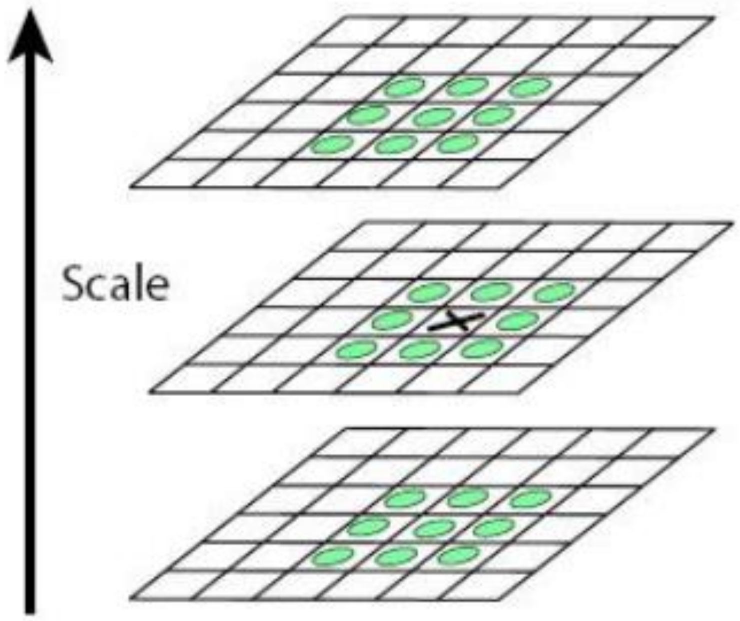
Region Detection: Laplacian of Gaussian

- 1/ Convolve image with scale-normalized Laplacian at several scales

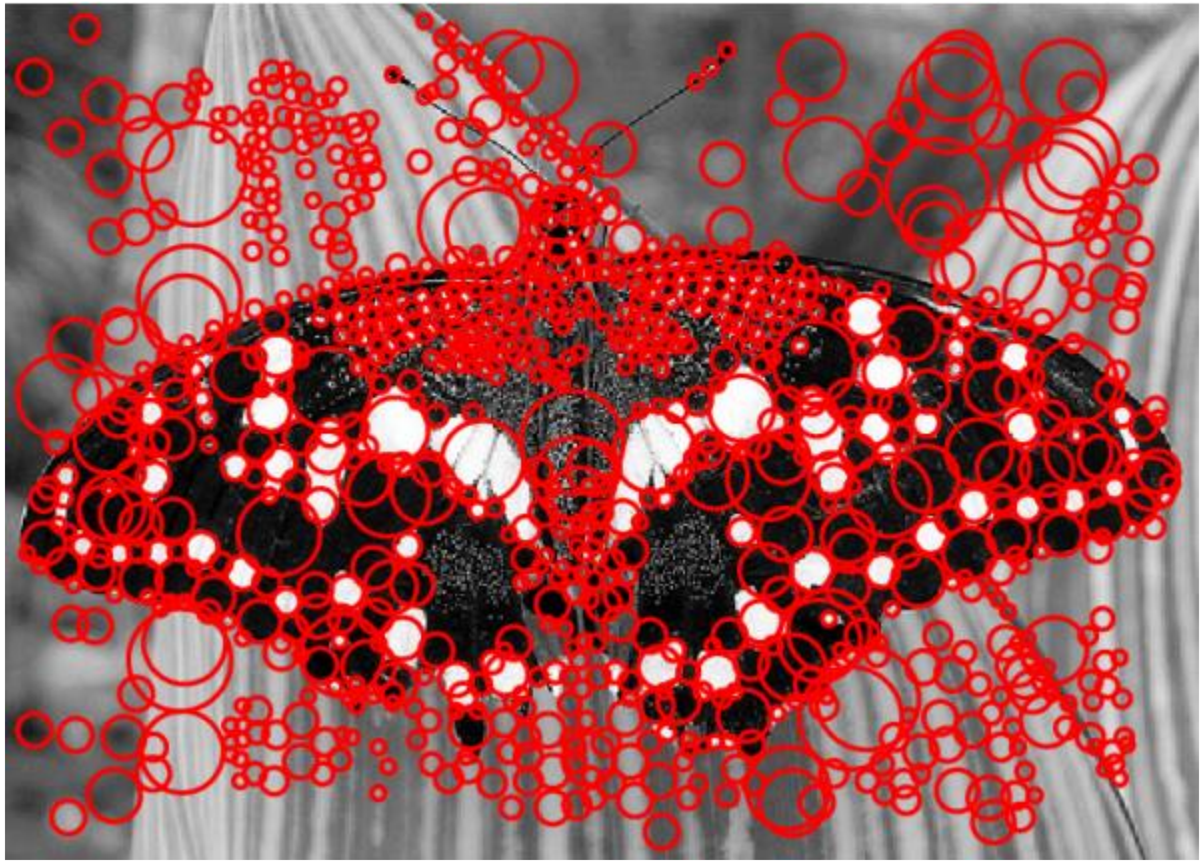
sigma = 11.9912

Region Detection: Laplacian of Gaussian

- 2/ Find maxima of squared Laplacian response in scale-space

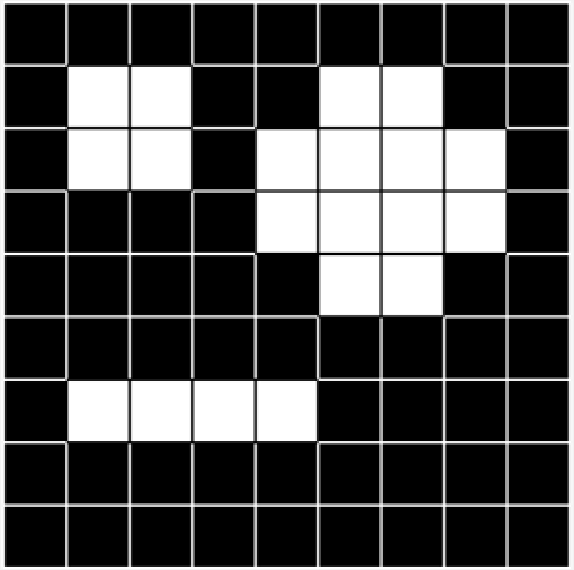


Region Detection: Laplacian of Gaussian

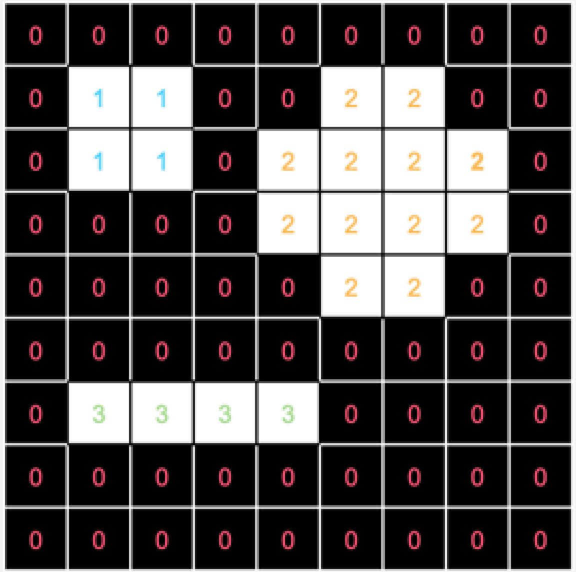


Region Detection: Blob Detection in Binary Images

- Blob in binary images:
  - Collection of white pixels connected to each other
  - The same label is given to all the pixels belonging to the same blob
  - Useful to count the number of object in the scene



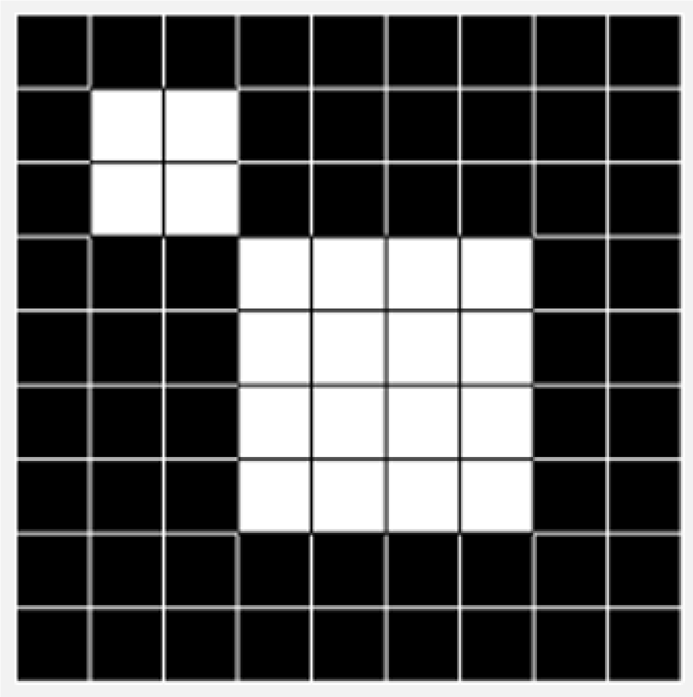
Original image



Labeled image

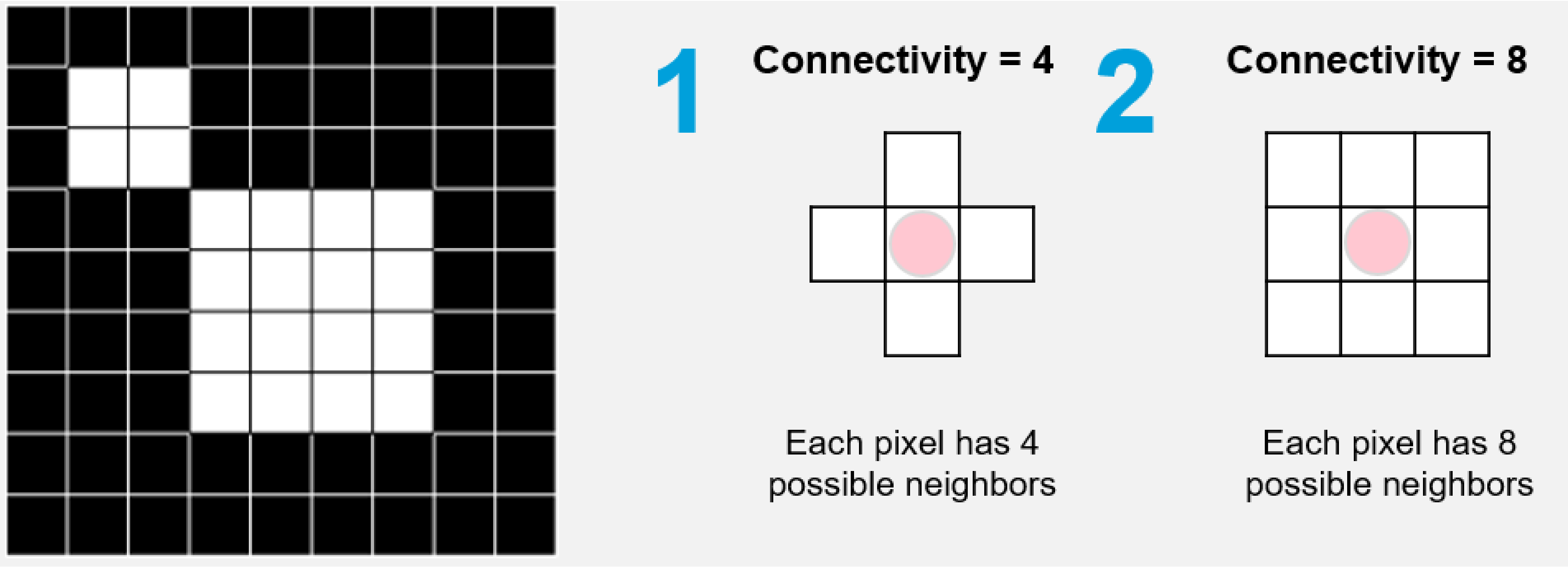
Region Detection: Blob Detection in Binary Images

- One or two objects?



Region Detection: Blob Detection in Binary Images

- Connected component analysis

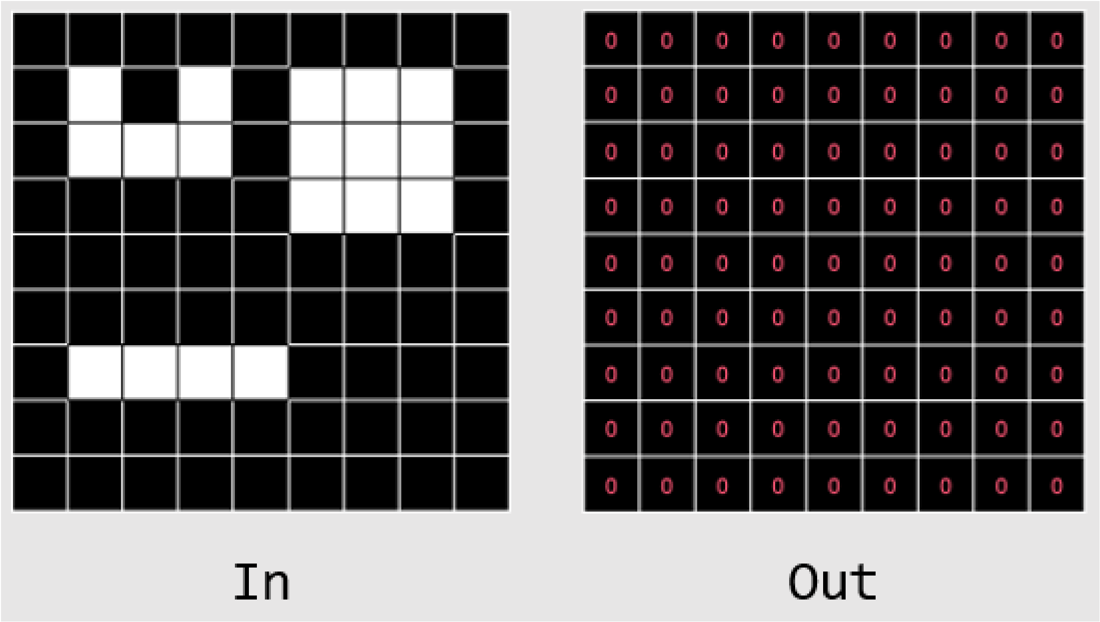


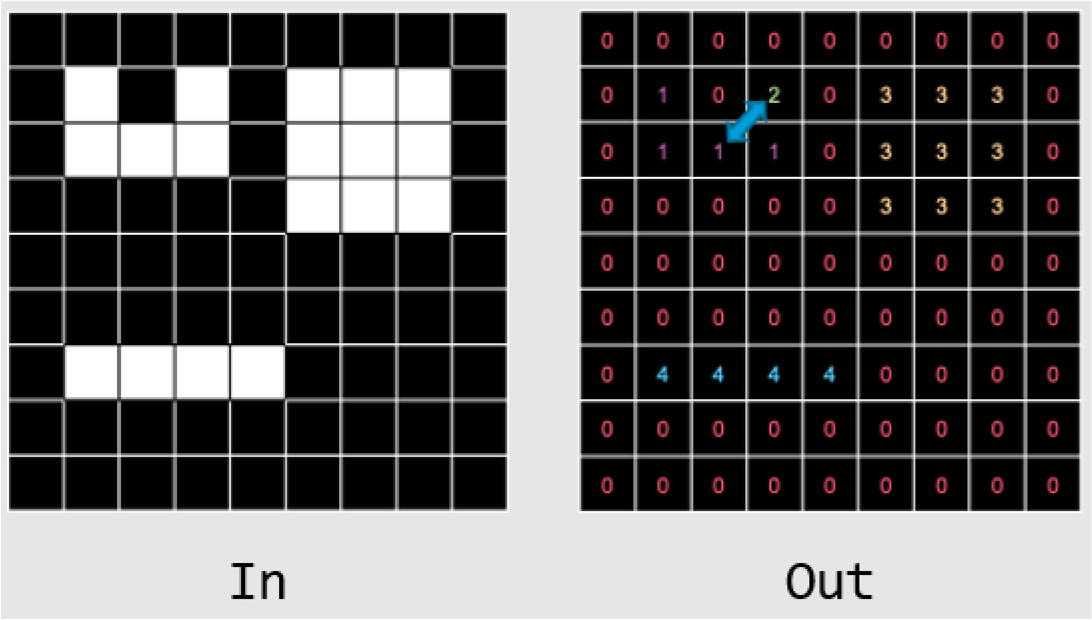
Region Detection: Blob Detection in Binary Images

- Connected component analysis algorithm: First stage

```
1.Create an image with the same dimension and inizialized to 0

2.Label = 1

3.Move over the image pixel by pixel in direction left-right and
  top-down

4.if In(x,y) == 0 go to the next pixel

5.else check the neighbor's labels in Out(x,y) image

    i.  if all the neigbours are 0 assign Label and Label += 1

    ii. elif only one label in the neigbours assign this label to
        Out(x,y)

    iii.else there are more than one label assign the lowest one
        and add to the equivalent table the neighbor's labels
```

Region Detection: Blob Detection in Binary Images





Region Detection: MSER

Maximally Stable Extremal Regions

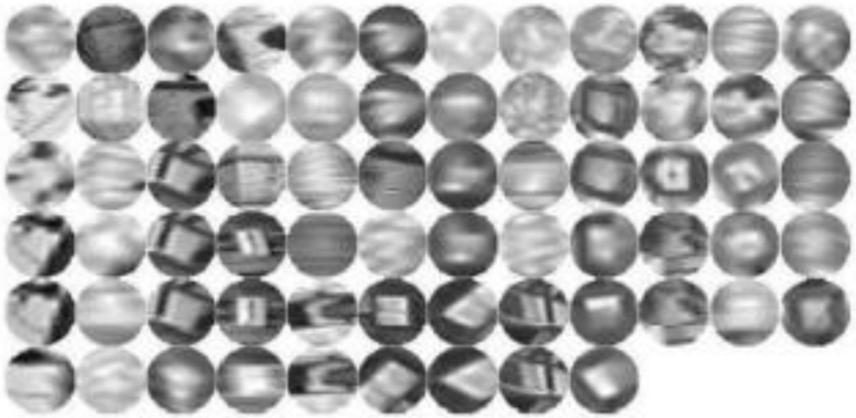
- MSER regions are connected areas characterized by almost uniform intensity, surrounded by contrasting background.

- They are constructed through a process of trying multiple thresholds

- The selected regions are those that maintain unchanged shapes over a large set of thresholds.

Region Detection: MSER

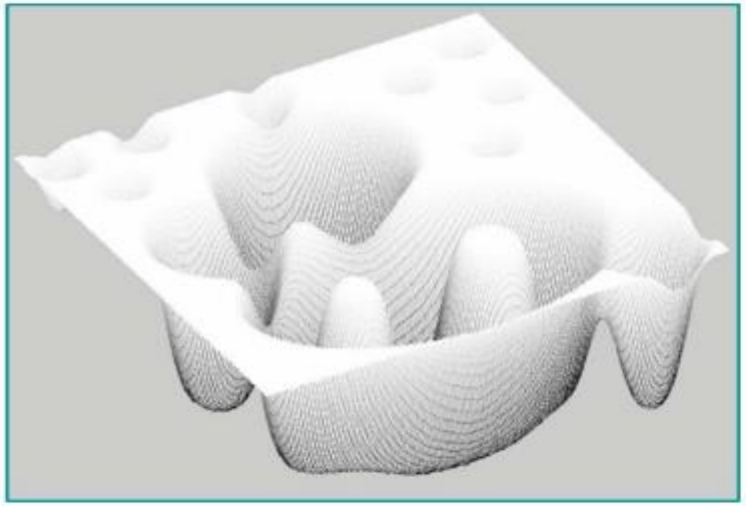
Examples of MSER regions



Region Detection: MSER

How to create MSER regions

intensity image

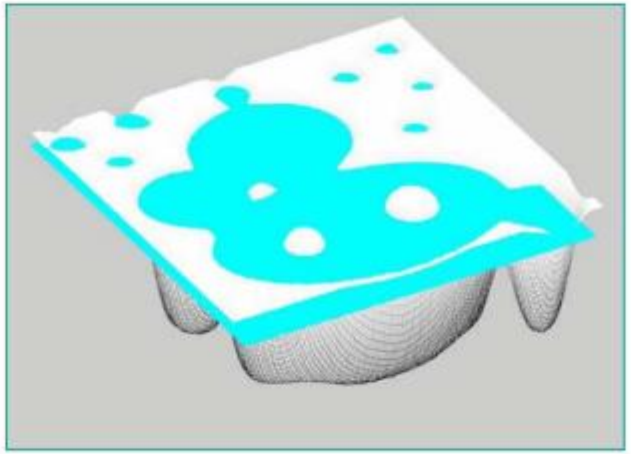


shown as a surface function

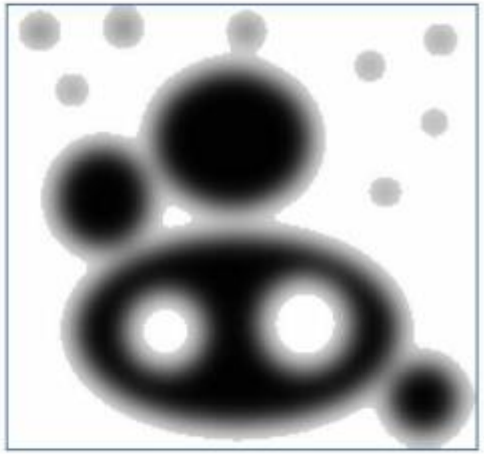
Do you remember watersheds?

Region Detection: MSER

How to create MSER regions



Threshold simulation



External regions (we store region are per each threshold value)

Region Detection: MSER

How to create MSER regions

- For each threshold, compute the connected binary regions
- Calculate Area at each threshold value
- Analyze this function to determine those regions that have a similar value over multiple thresholds
- Regions detected at different thresholds have different areas

Region Detection: Python Example (LoG)

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import blob_log
import cv2 as cv2

im = cv2.imread('C:/Users/Jorge/Desktop/FCV/almendras.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('orig',im)
cv2.waitKey(0)

#binarize the image
im_bw, th = cv2.threshold(im,200,255,cv2.THRESH_BINARY_INV)
print(th.shape)
cv2.imshow('bin',th)
cv2.waitKey(0)
```

Region Detection: Python Example (LoG)

```
blobs = blob_log(th, max_sigma=30, min_sigma = 3, num_sigma=2, threshold=0.3, overlap = 0.1)
fig, ax = plt.subplots()
ax.imshow(th, cmap='gray')
for blob in blobs:
    y, x, area = blob
    ax.add_patch(plt.Circle((x, y), area*np.sqrt(2), color='r',
                      fill=False))

cv2.destroyAllWindows()
```

Region Detection: Python Example (MSER)

```
detector = cv2.MSER_create()
keypoints = detector.detect(th, None)

# We draw all keypoints detected in the image
for keypoint in keypoints:
    radius = int(0.5 * keypoint.size)
    x, y = np.int64(keypoint.pt)
    cv2.circle(im, (x, y), radius, (0, 255, 255), 2)

# We show the original image and the one with the regions marked
cv2.imshow('Imágenes', np.hstack([copia, im]))
```

Fundamentals of Computer Vision

Unit 6: Feature Extraction

Jorge Bernal