3D Data Processing

Feature extraction

Hyoseok Hwang

Knowledge

- What are in the image?
- How to know?

- By comparing the video with your knowledge

Features

- What do we compare?
  - Planes?
  - Edges?
  - Point(region)?
  - Circles, ellipses, lines, blobs etc

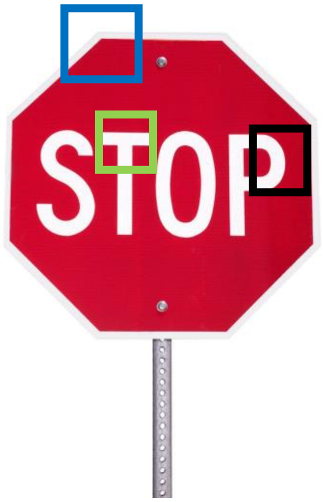
Features

- Comparison from features

Features

Features

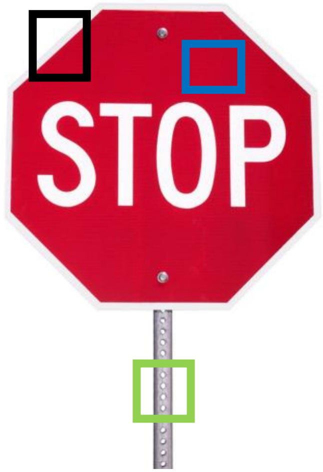
- Comparison from features



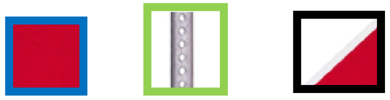
Features

Features

- Comparison from features

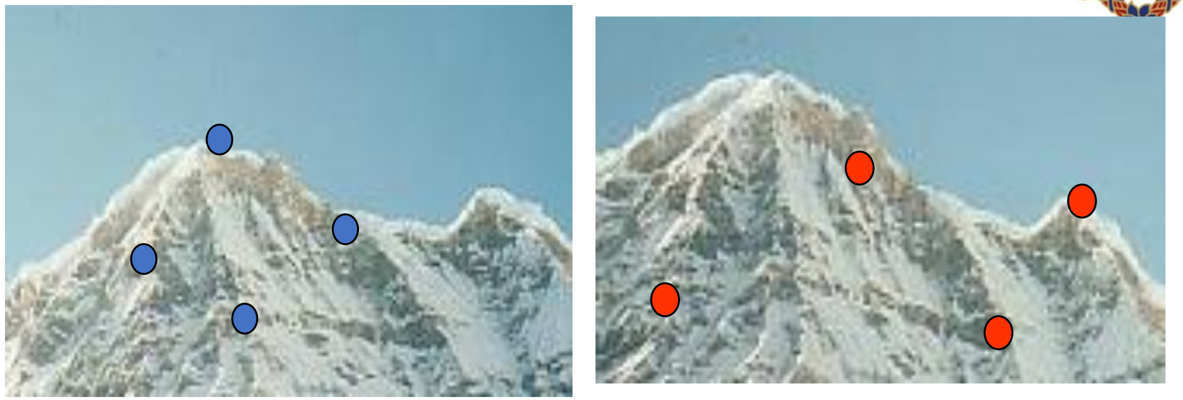


Features



Features : Preview

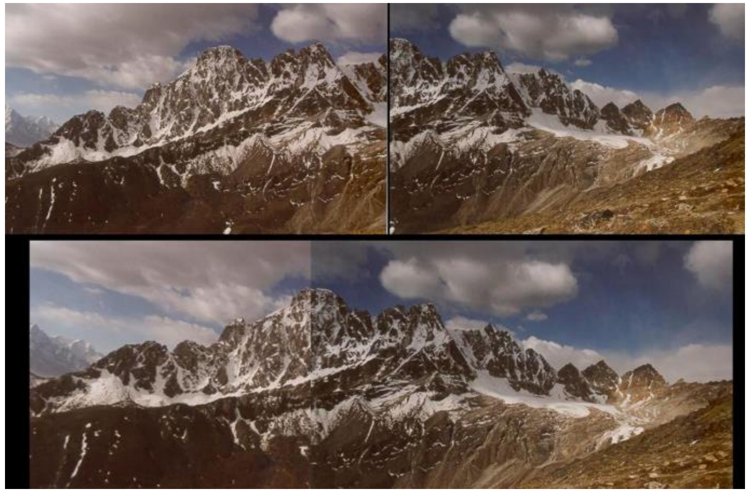
- Detection (extraction)
  - Define interest points (region)
- Description
  - Feature descriptor
- Matching
  - Determine correspondence between descriptors



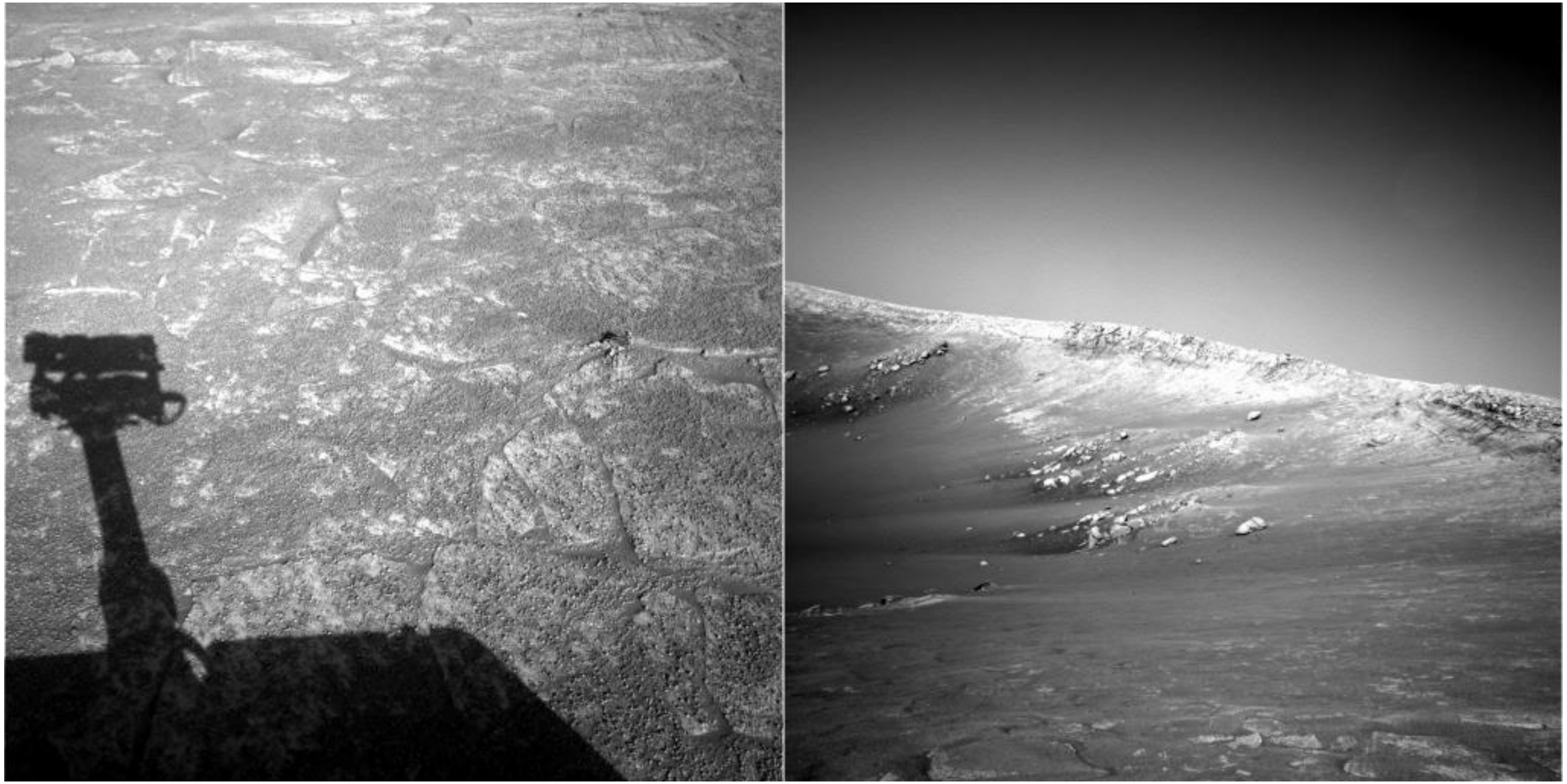
No chance to find true matches!

Features : Preview

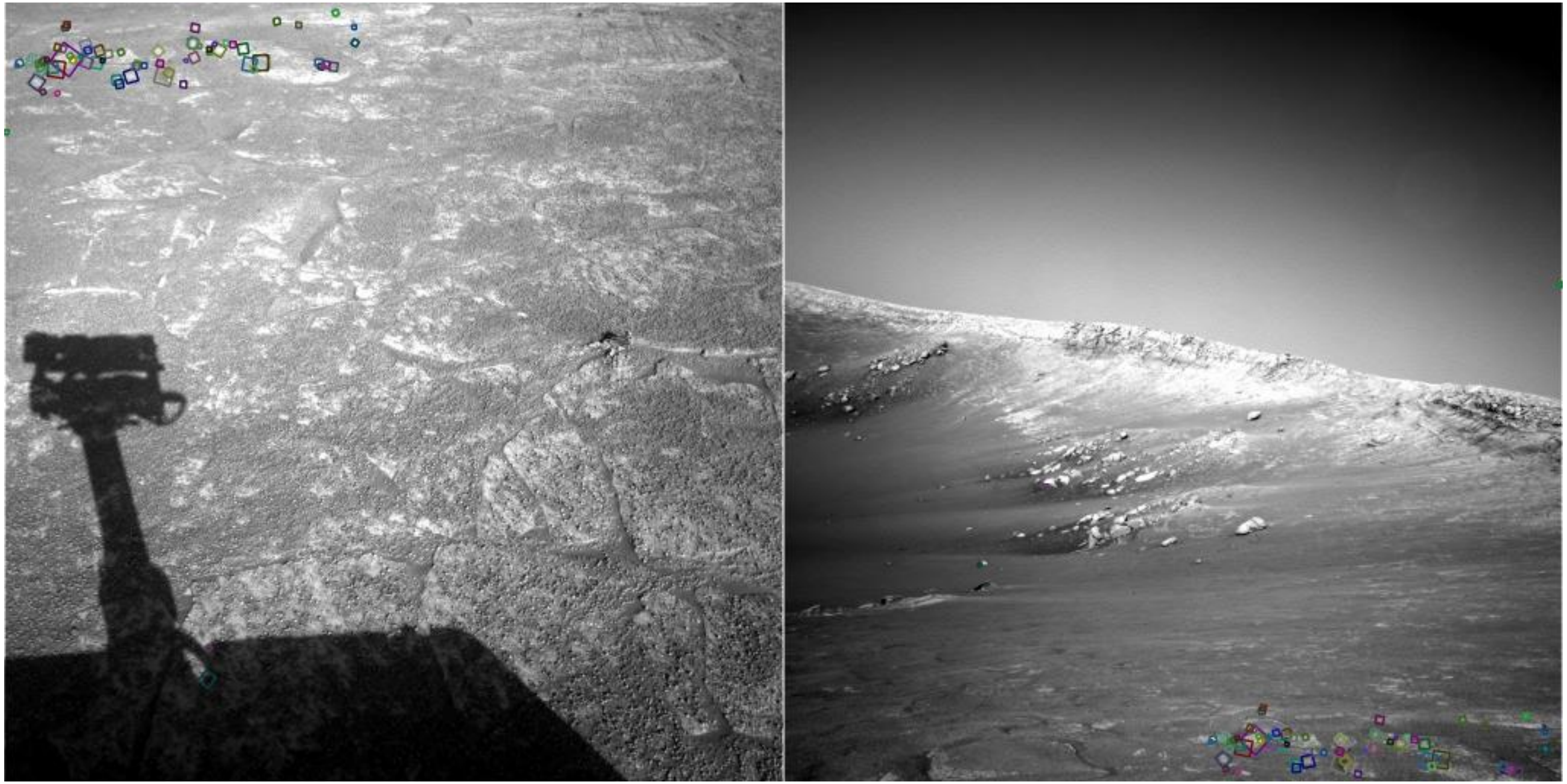
- Applications of Feature Matching
  - Image alignment and stitching
  - **3D structure from motion**
  - Motion tracking
  - Robot navigation
  - Image retrieval
  - Object and face recognition
  - Human action recognition



An example of feature matching



An example of feature matching



NASA Mars Rover images
with SIFT feature matches

Features

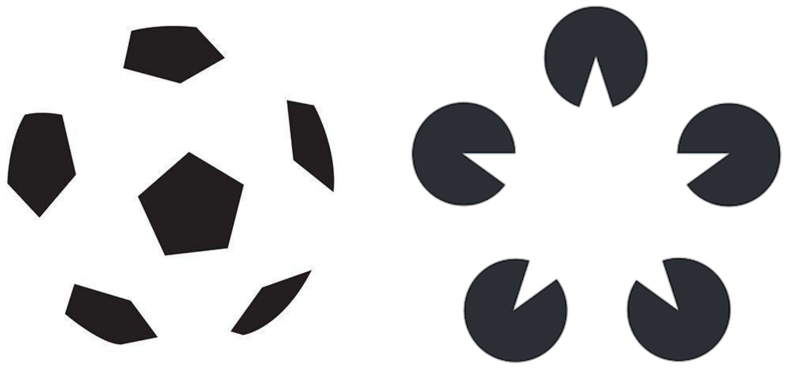
- Requirements
  - **Saliency**
    - Same points in different images should have similar features and vice-versa
  - **Repeatability**
    - Should be detectable at the same locations in different images despite changes in viewpoint and illumination
  - **Rotation invariant**
  - **Scale invariant**
  - Affine invariant

Repeatability?


Scale invariant

Appendix

- Gestalt psychology
  - The way we form our perceptions are guided by certain principles or laws.
  - These principles or laws determine what we see or make of things or situation



Law of closure

Find an animal

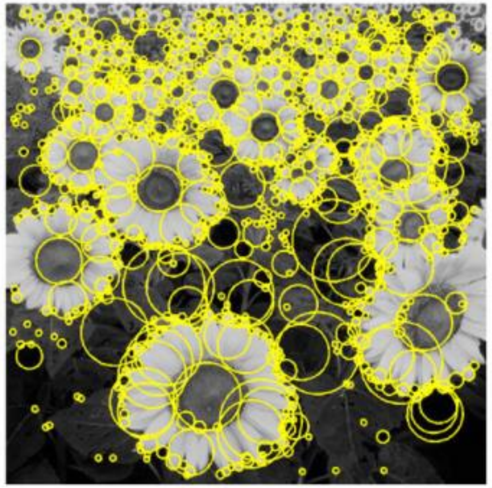
Appendix

- Gestalt psychology

Features

- What types can we use for features?
  - Edges
    - Distinguishability of edge is low
  - Region around a point are usually used
    - Region contains corners
    - Blob region

corners



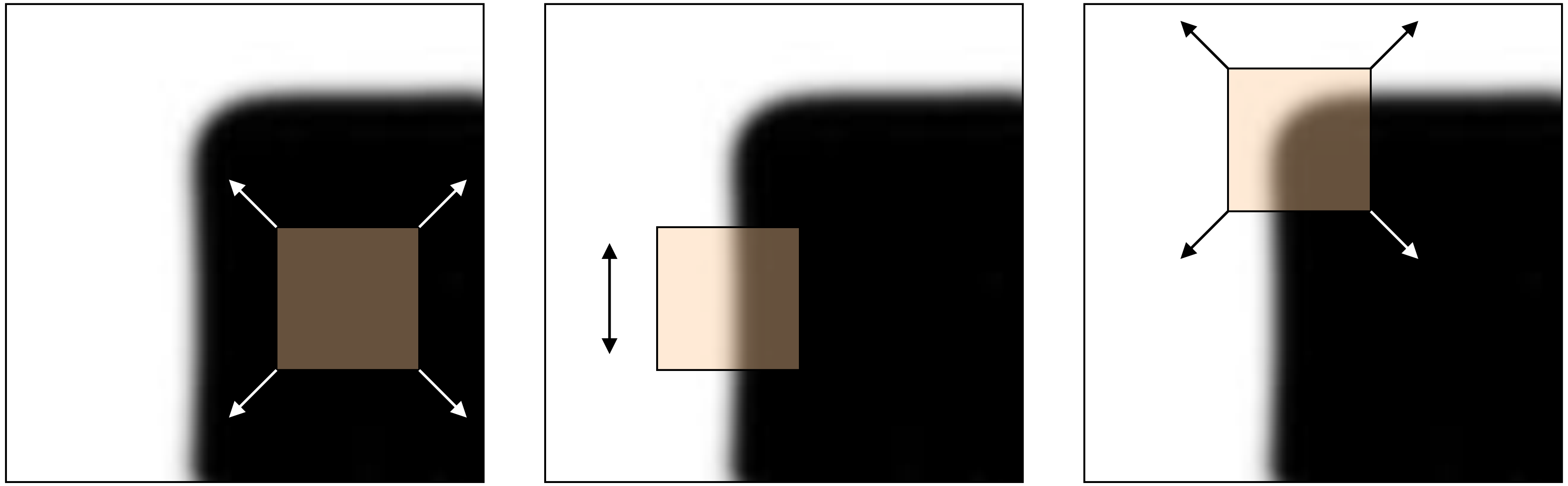
blobs

Corner detection

- What points would you choose?
- How can define them in mathematics?

Corner detection (Harris)

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



"flat" region:
no change in all directions

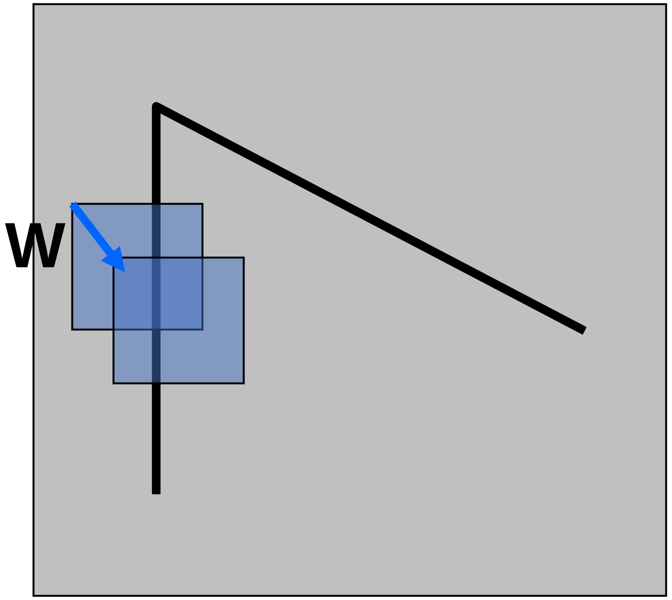
"edge":
no change along the edge direction

"corner":
significant change in all directions

Corner detection

- Consider shifting the window W by (u,v)
  - How do the pixels in W change?
  - Compare each pixel before and after by summing up the squared differences (SSD)
  - This defines an SSD "error" of E(u,v):

$$E(u,v) = \sum_{(x,y)\in W} [I(x+u, y+v) - I(x,y)]^2$$



SSD: Sum of Squared Difference

appendix: $SSD = \sum_{(u,v)\in I} (I_1[u,v] - I_2[u,v])^2$

Small motion assumption

- Taylor Series expansion of I:

$$I(x+u, y+v) = I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

- If the motion (u,v) is small, then first order approximation is good

$$\begin{aligned} I(x+u, y+v) &\approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x,y) + [I_x \;\; I_y] \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

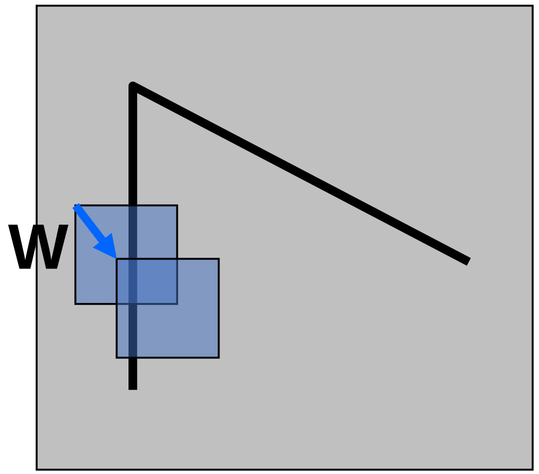
Corner detection

- Using the small motion assumption, replace I with a linear approximation

$$E(u,v) = \sum_{(x,y)\in W} \left(I(x+u,y+v) - I(x,y)\right)^2$$

$$\approx \sum_{(x,y)\in W} \left(I(x,y) + I_x(x,y)u + I_y(x,y)v - I(x,y)\right)^2$$

$$\approx \sum_{(x,y)\in W} \left(I_x(x,y)u + I_y(x,y)v\right)^2$$



(Shorthand: $I_x = \frac{\partial I}{\partial x}$)

Corner detection

- Using the small motion assumption, replace I with a linear approximation

$$
\begin{aligned}
E(u,v) &\approx \sum_{(x,y)\in W} \left(I_x(x,y)u + I_y(x,y)v\right)^2 \\
&\approx \sum_{(x,y)\in W} \left(I_x^2 u^2 + 2I_x I_y uv + I_y^2 v^2\right) \\
&\approx Au^2 + 2Buv + Cv^2
\end{aligned}
$$

$$
A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2
$$

- Thus, *E*(*u*,*v*) is locally approximated as a *quadratic form*

Quadratic form

- A quadratic form is a polynomial with terms all of degree two ("form" is another name for a homogeneous polynomial).
- A Simple example

$$4x^2 + 2xy - 3y^2$$

- This also be represented using matrix in the form of $Q(x) = x^T A x$

$$4x^2 + 2xy - 3y^2 \quad \rightarrow \quad \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Therefore,

$$Au^2 + 2Buv + Cv^2 \quad \rightarrow \quad \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Corner detection

- The second moment matrix
  - The surface *E*(*u*,*v*) is locally approximated by a quadratic form.

$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

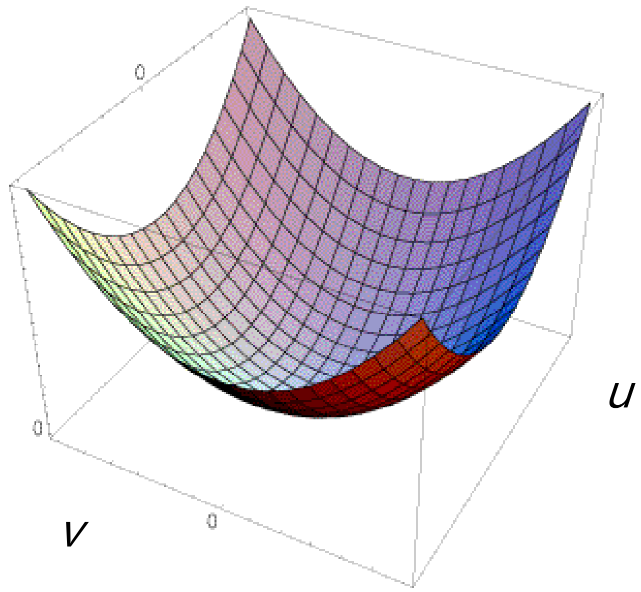
$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

Symmetric matrix

$$A = \sum_{(x,y)\in W} I_x^2$$

$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



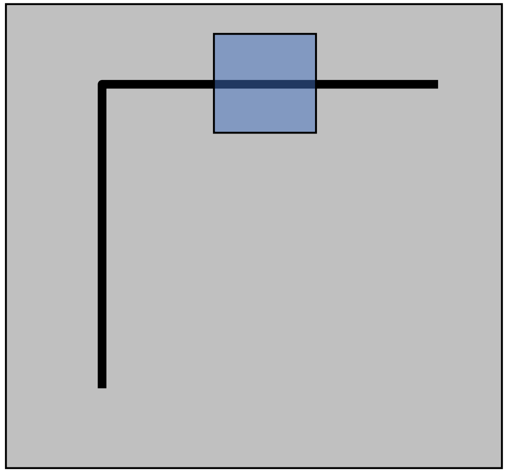
Corner detection

- The second moment matrix
  - The surface *E*(*u*,*v*) is locally approximated by a quadratic form.

$$A = \sum_{(x,y)\in W} I_x^2$$

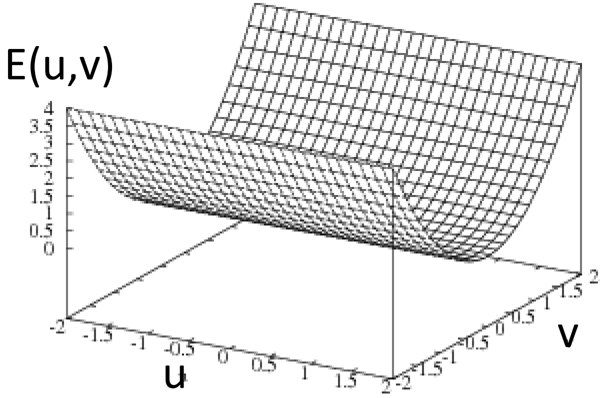
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$



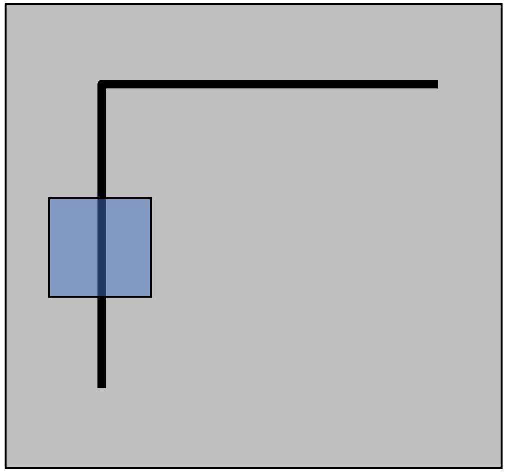
Corner detection

- The second moment matrix
  - The surface $E(u,v)$ is locally approximated by a quadratic form.

$$A = \sum_{(x,y)\in W} I_x^2$$

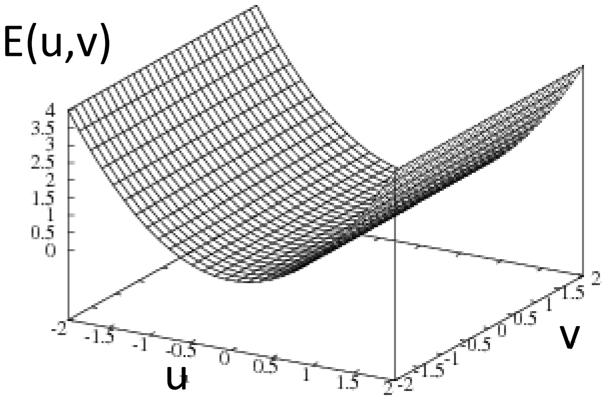
$$B = \sum_{(x,y)\in W} I_x I_y$$

$$C = \sum_{(x,y)\in W} I_y^2$$



Vertical edge: $I_y = 0$

$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$



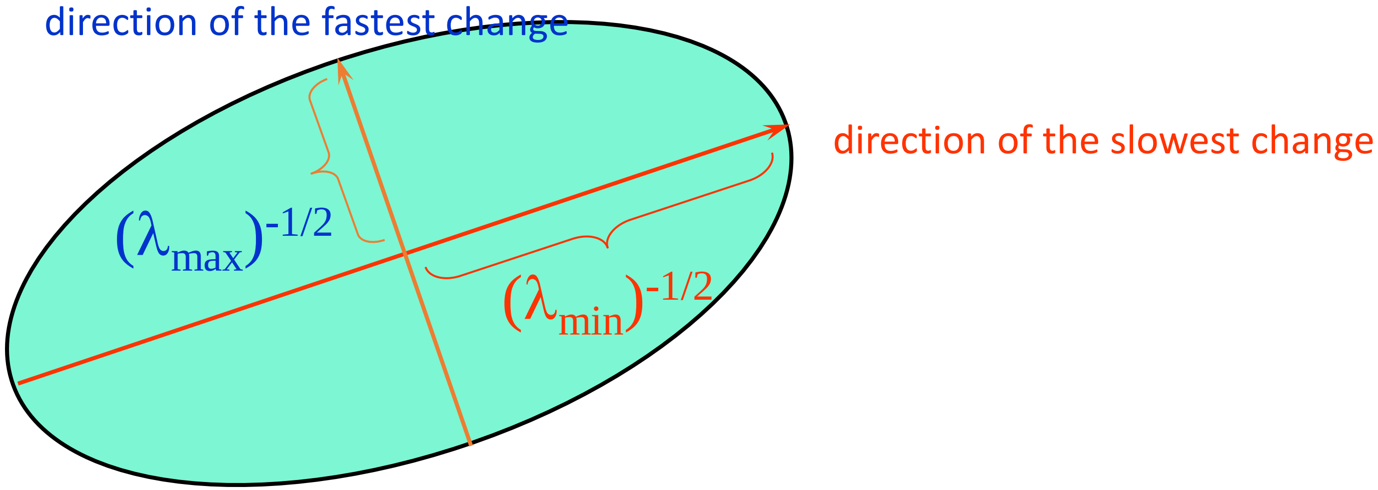
Corner detection

- General case
  - The shape of H tells us something about the distribution of gradients around a pixel
  - We can visualize H as an ellipse with axis lengths determined by the eigenvalues of H and orientation determined by the eigenvectors of H

Ellipse equation:

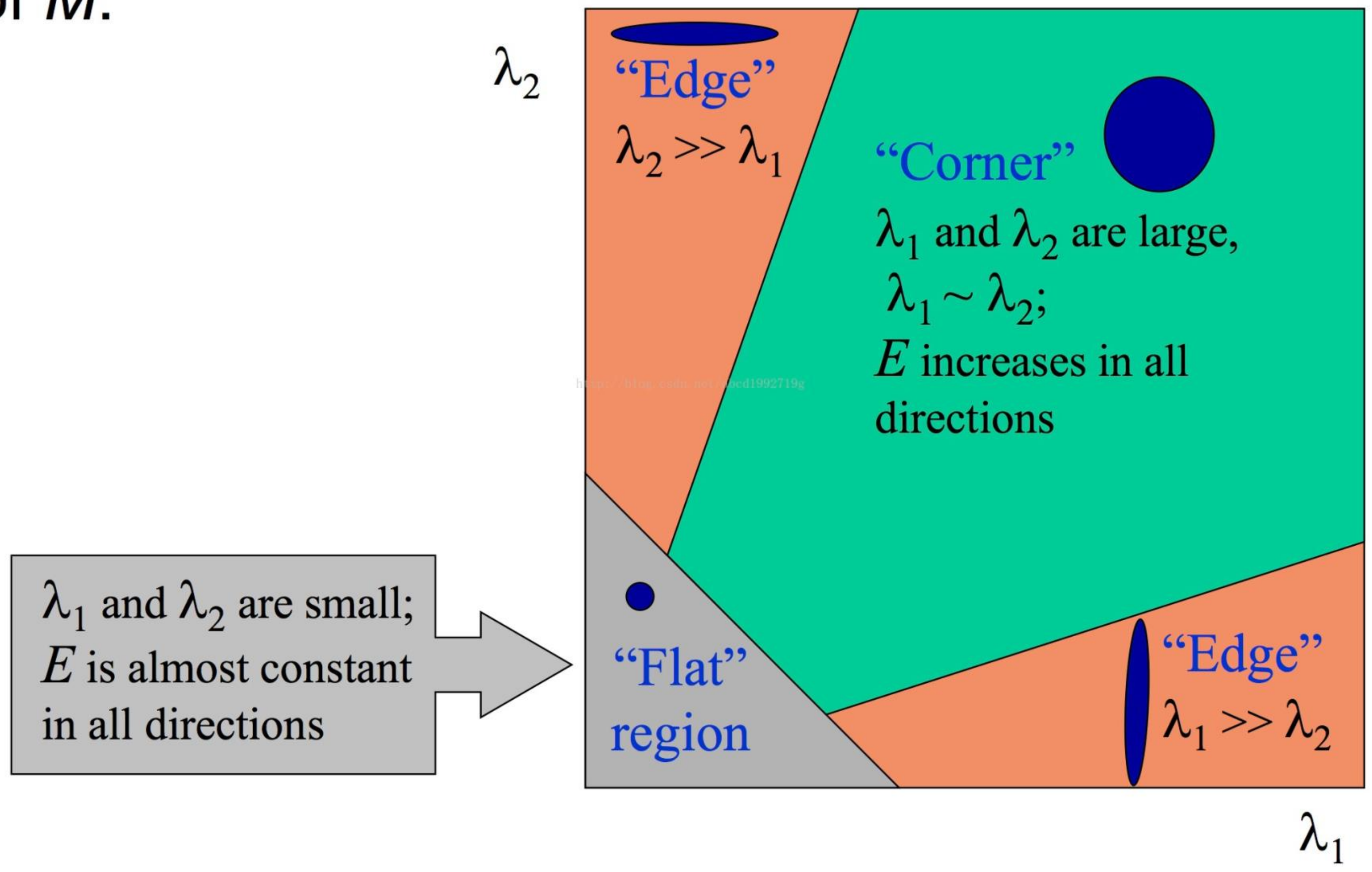
$$[u \;\; v] \;\; H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$

λ_{max}, λ_{min} : eigenvalues of *H*



Corner detection

Classification of image points using eigenvalues of *M*:



Corner detection

- Corner definition
  - How can we define that both eigenvalues are "large enough?"
  - There are many suggestions, but the following two equations are mainly used for the response function.

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

$$f = \frac{\lambda_1\lambda_2}{\lambda_1 + \lambda_2}$$

$$\begin{bmatrix} A & B \\ B & C \end{bmatrix}$$

Determinant of H: $\lambda_1\lambda_2$
= AC - BB

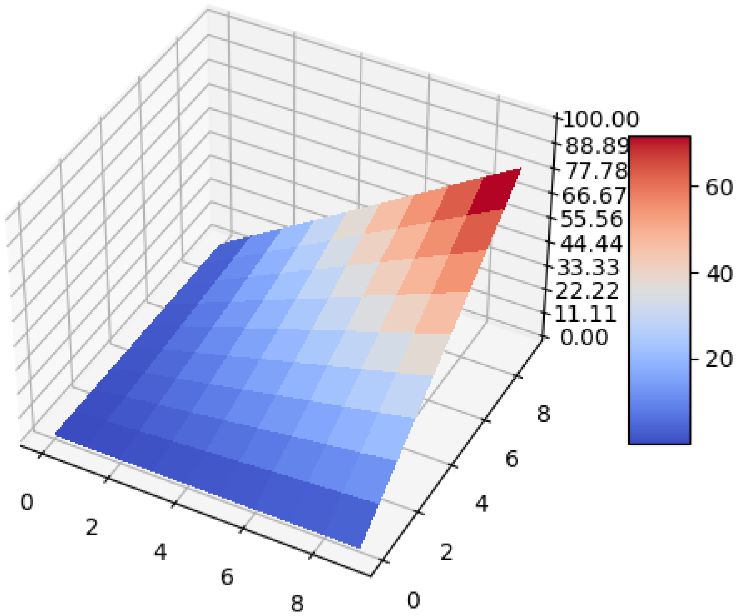
Trace of H: $\lambda_1 + \lambda_2$
= A+C

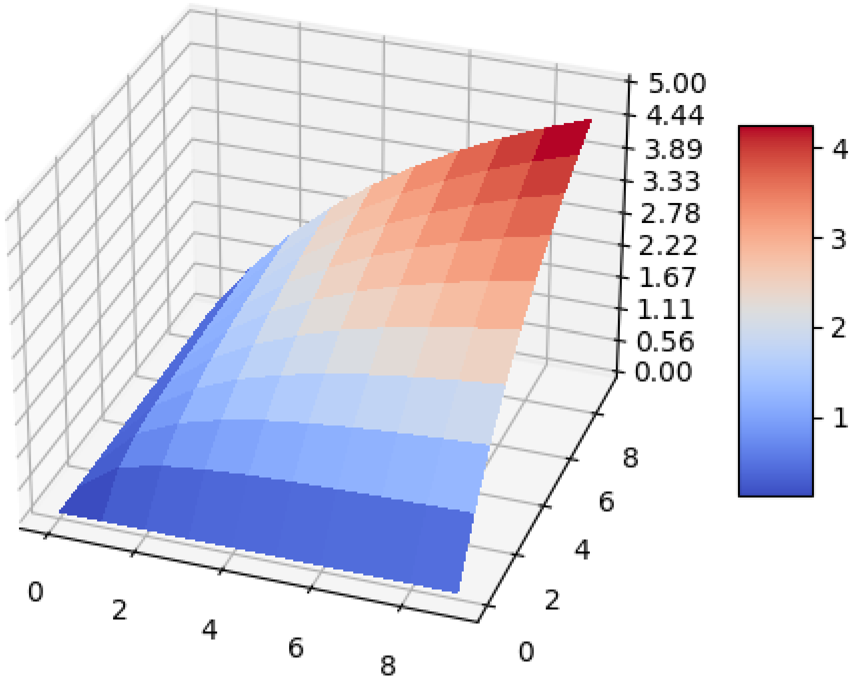
Corner detection

- Corner definition

$$\lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

$$\frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$



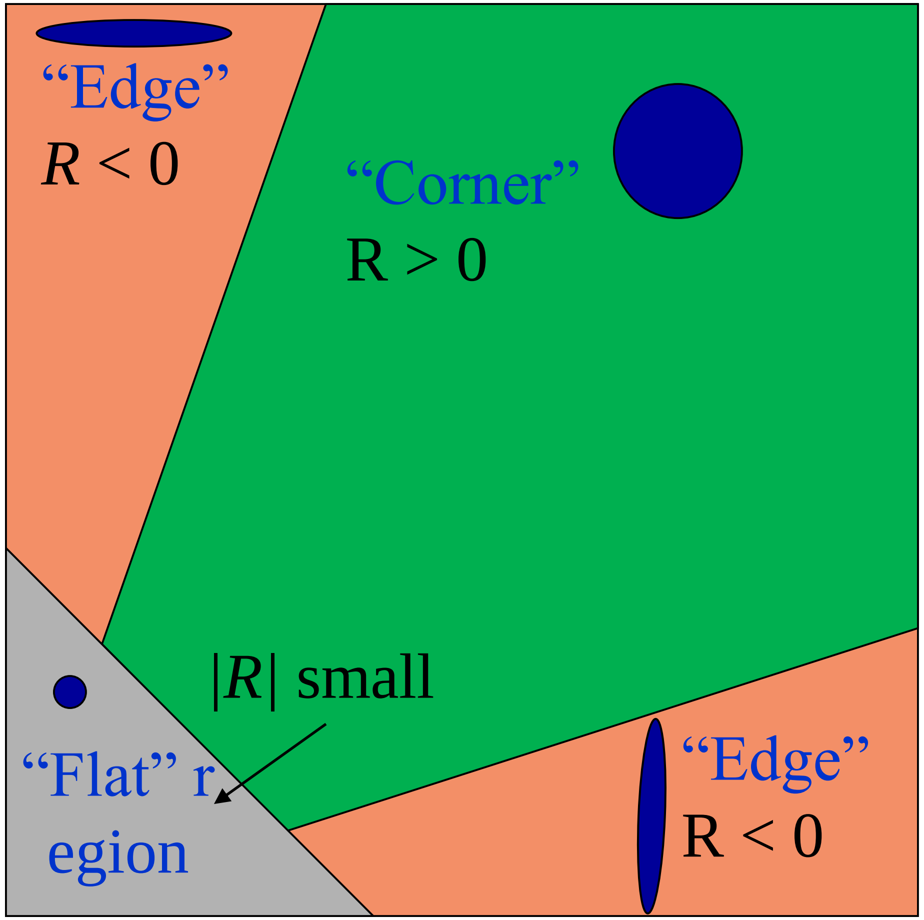


Harris corner

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

α: constant (0.04 to 0.06)

- Should we calculate the eigenvalues for every window every time?
- NO. Instead, we can only compute determinant and trace of H

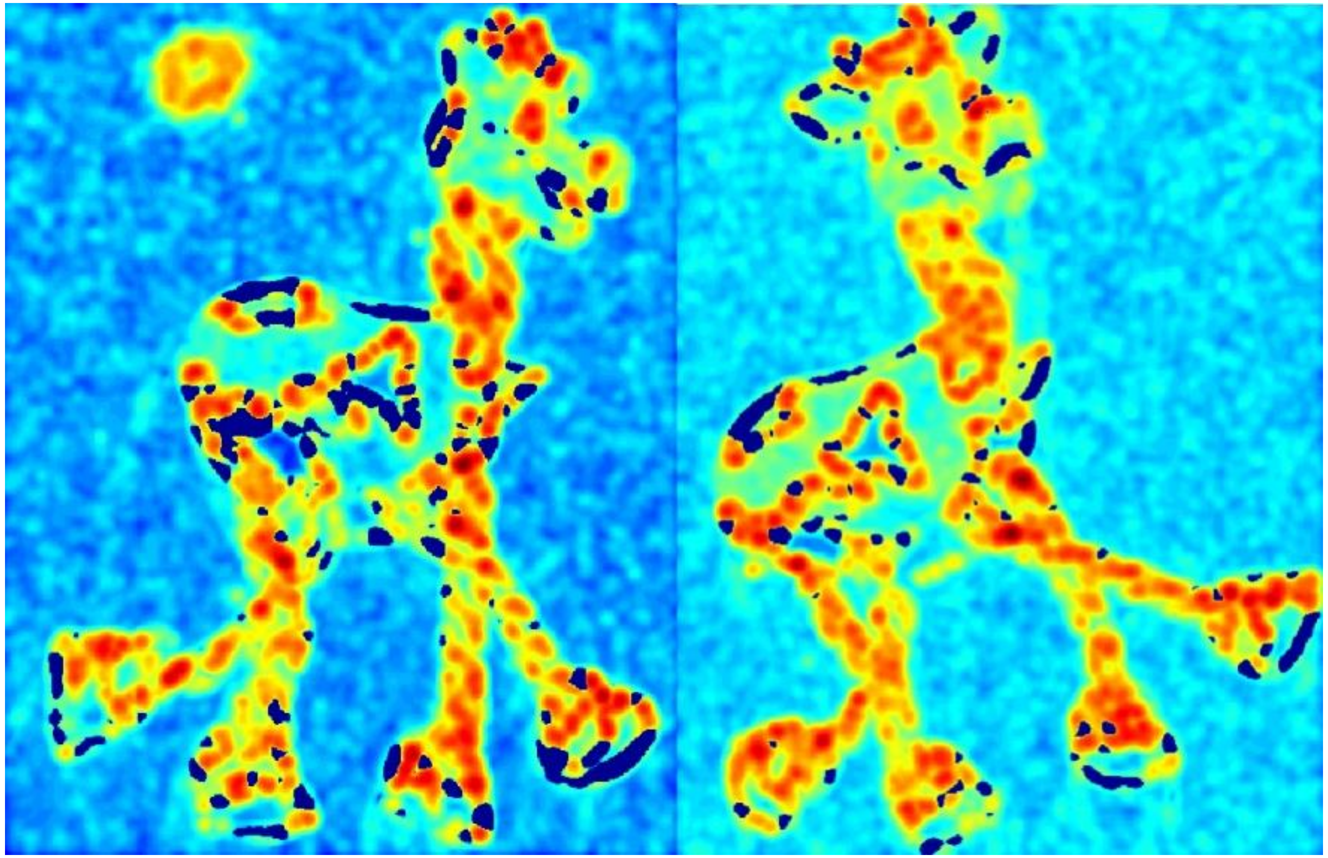


Harris corner

- Example

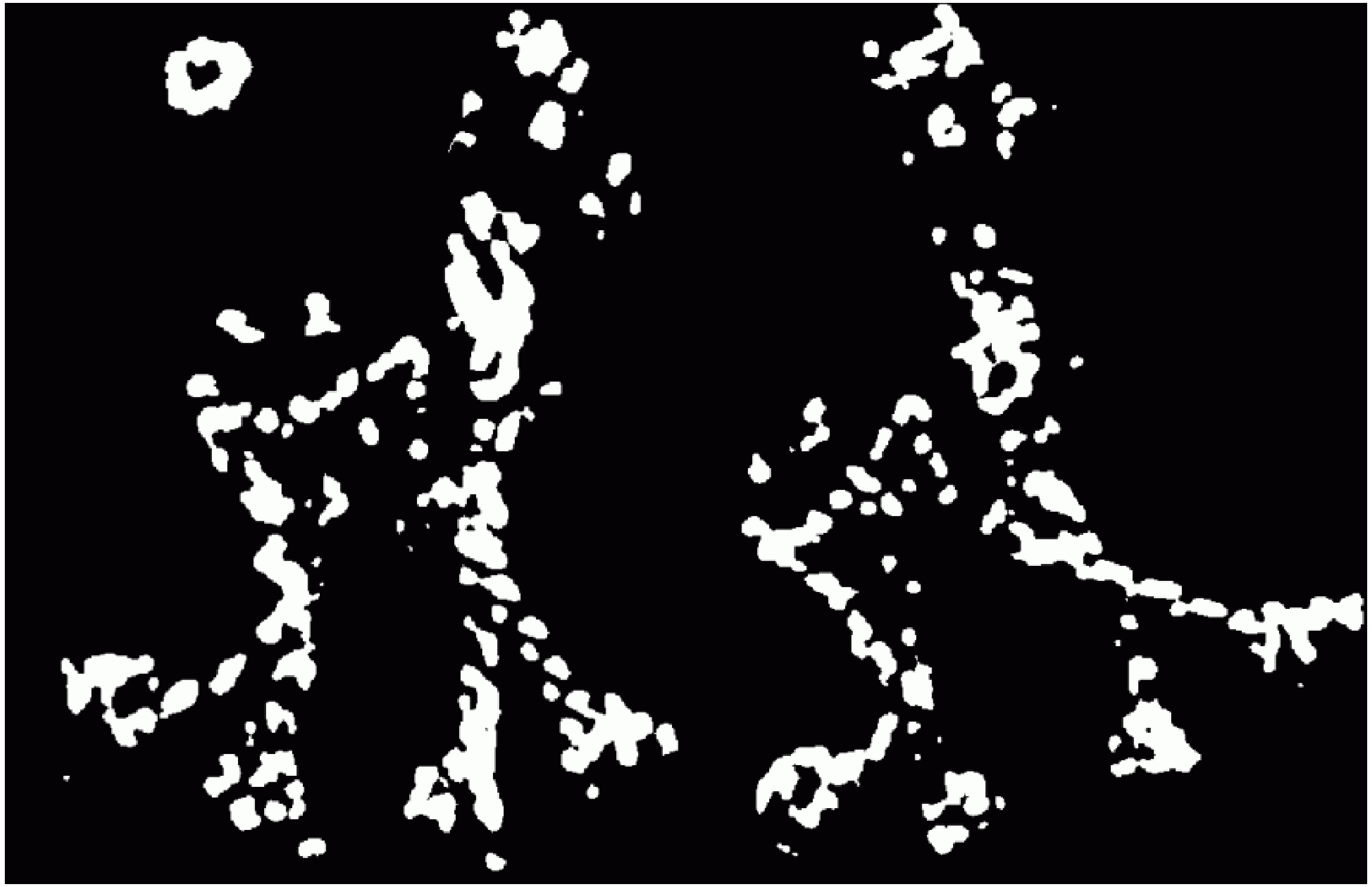
Harris corner

- Get R



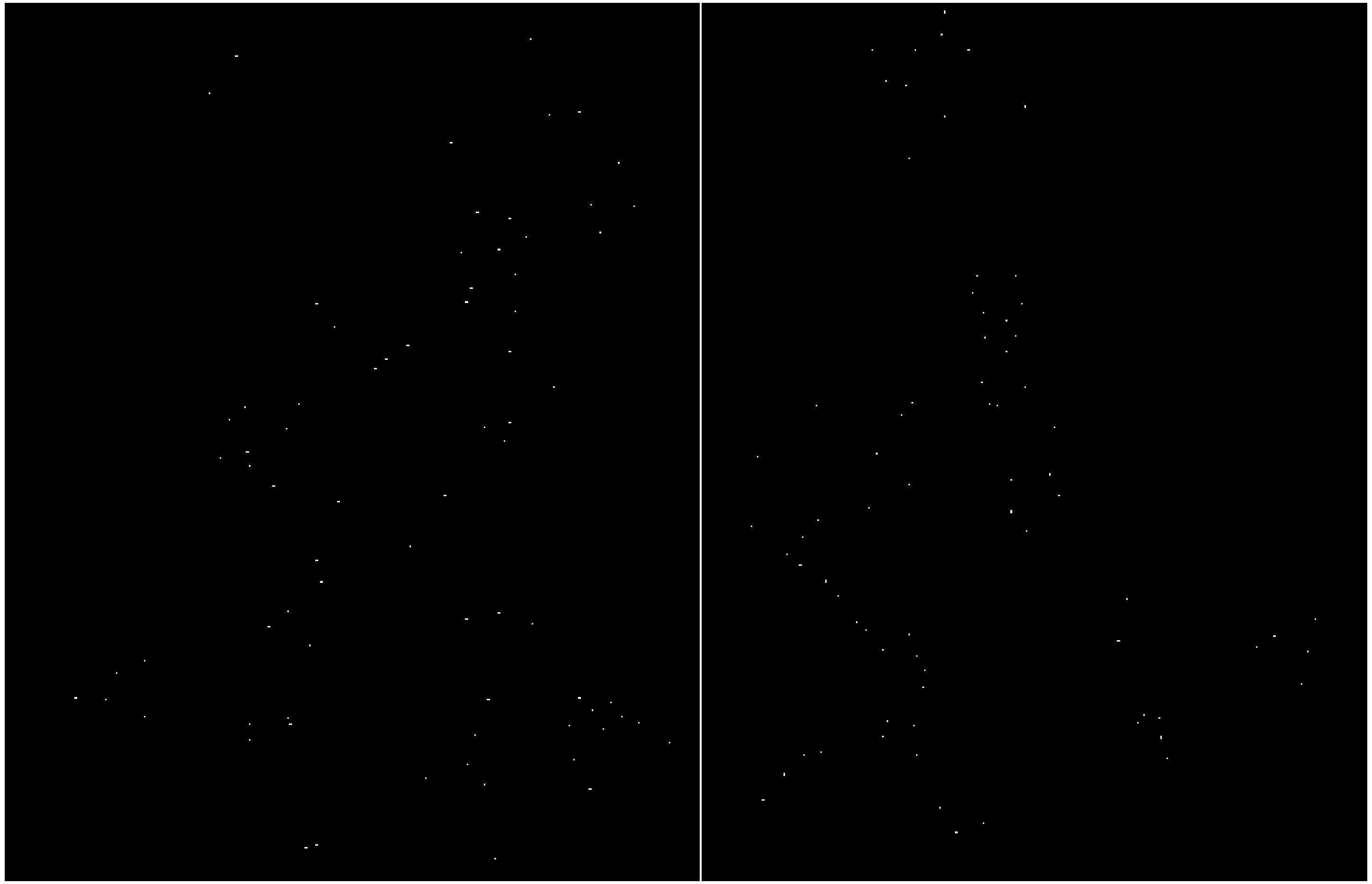
Harris corner

- Threshold (R > value)



Harris corner

- Find local maxima (Maximum values within defined area)



Harris corner

- Overlap on original image (corners: red points)

Harris corner

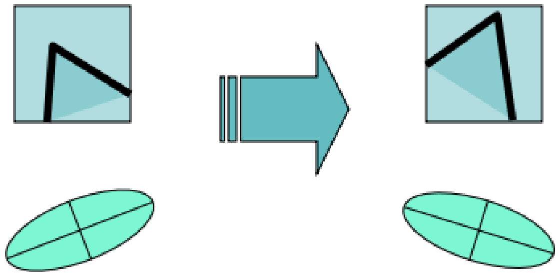
- Exercise

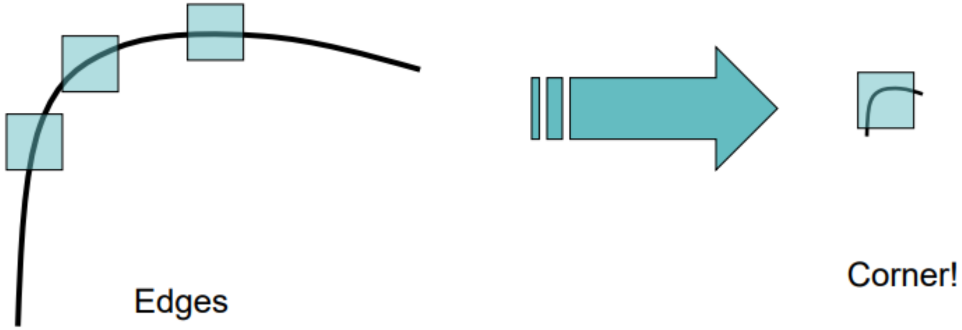
```
import cv2
import numpy as np

filename = 'zip.png'
img = cv2.imread(filename)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)

# #result is dilated for marking the corners, not important
# dst = cv2.dilate(dst,None)

# for display
for r in range(dst.shape[0]):
    for c in range(dst.shape[1]):
        if dst[r,c] > 0.02*dst.max():
            cv2.circle(img, (c, r), 3, (0, 255, 255), -1)

cv2.imshow('dst',img)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Corner detection

- Properties of Harris Corners
  - Rotation Invariance
    - Ellipse rotates but the shape (i.e. eigenvalues) remain the same
    - Corner response R is invariant to image rotation.
  - Partial invariance to affine intensity change

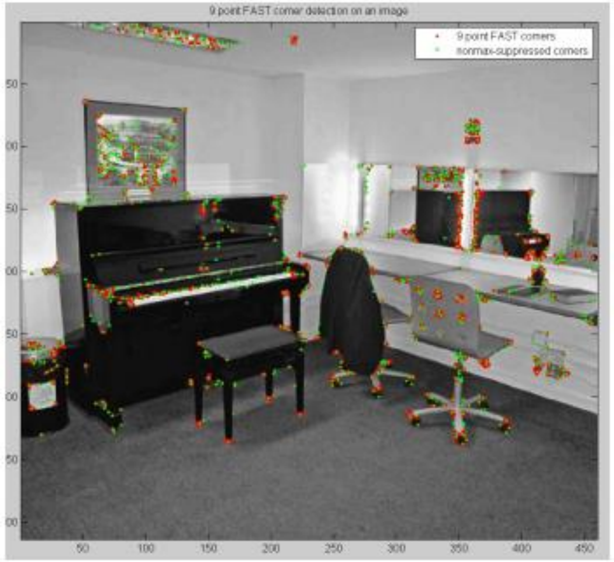
  - **NOT invariant to image scale**
  - Corner at one scale may not be a corner at another
  - Scale is user specified parameter



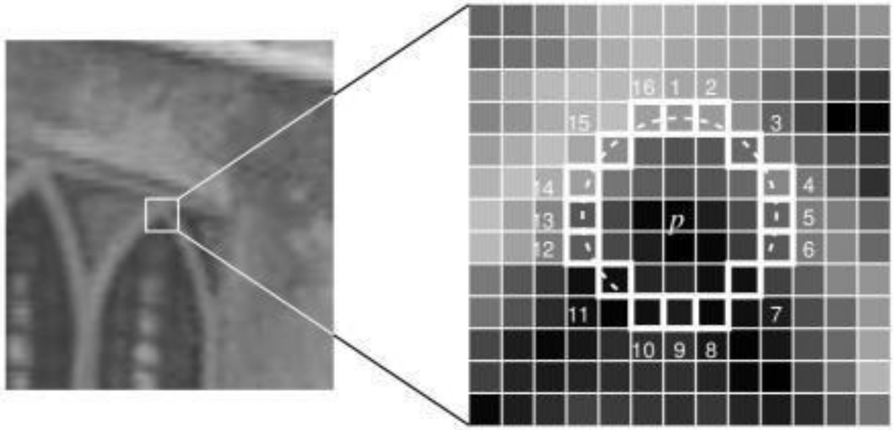


FAST

- Features from Accelerated Segment Test
  - Each pixel in the circle is labeled from integer number 1 to 16 clockwise.
  - A pixel p is a corner point if the pixel satisfies conditions below:
  - Condition 1
    - A set of N contiguous pixels S, $\forall x \in S$, the intensity of $x > I_p + \mathrm{t}$
  - Condition 2
    - A set of N contiguous pixels S, $\forall x \in S$, the intensity of $x < I_p - \mathrm{t}$



- Rotation invariant?
- Scale invariant?



Scale invariant feature

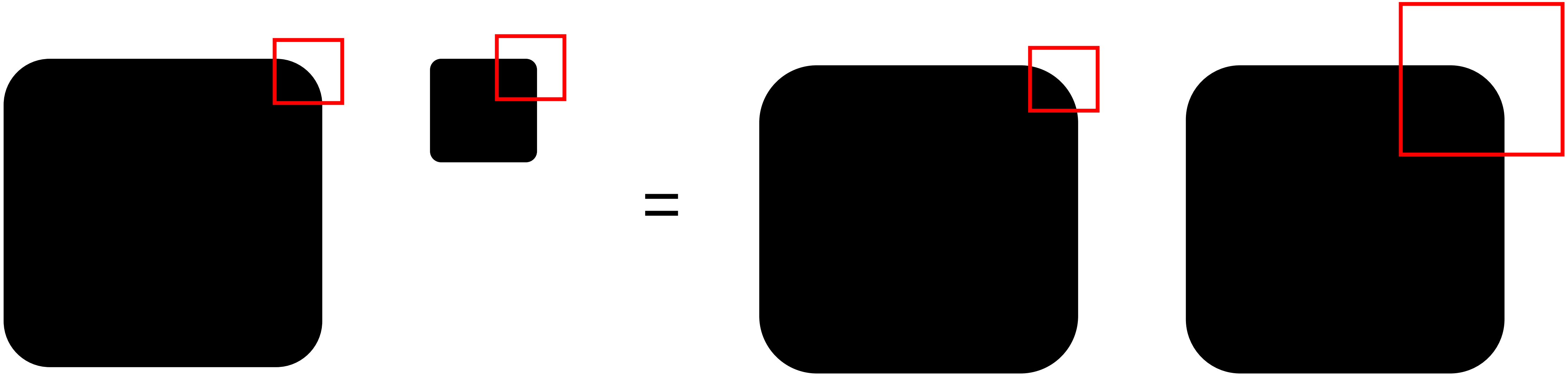
- Automatic scale selection by increasing the size of windows

$$f(I_{i_1 \ldots i_m}(x,\sigma)) = f(I_{i_1 \ldots i_m}(x',\sigma'))$$

How to find corresponding patch sizes?

Scale invariant feature

- Basic Idea
  - Response values of cornerness are determined by
    - the scale of image with the fixed window size=
    - the scale of window with the fixed image size
  - Solution: various window size or various image size

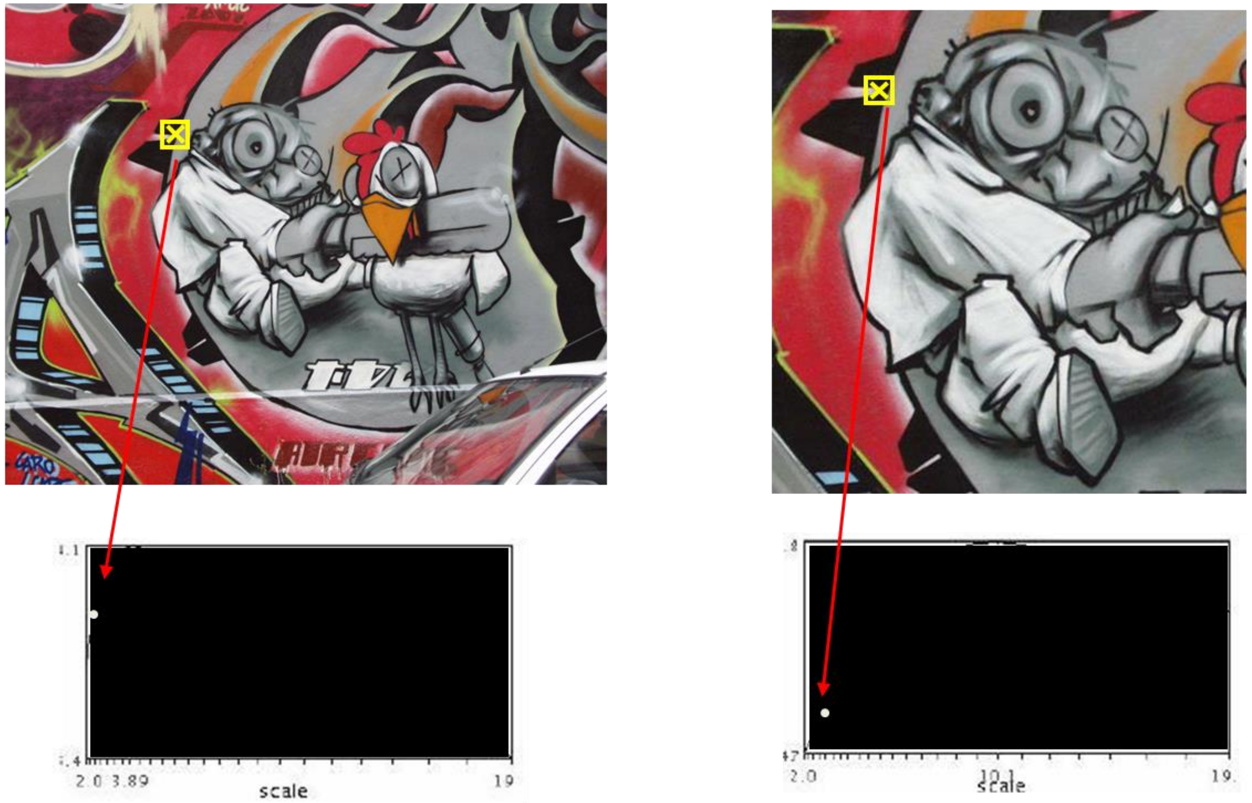


fixed window size

fixed image size

Scale invariant feature

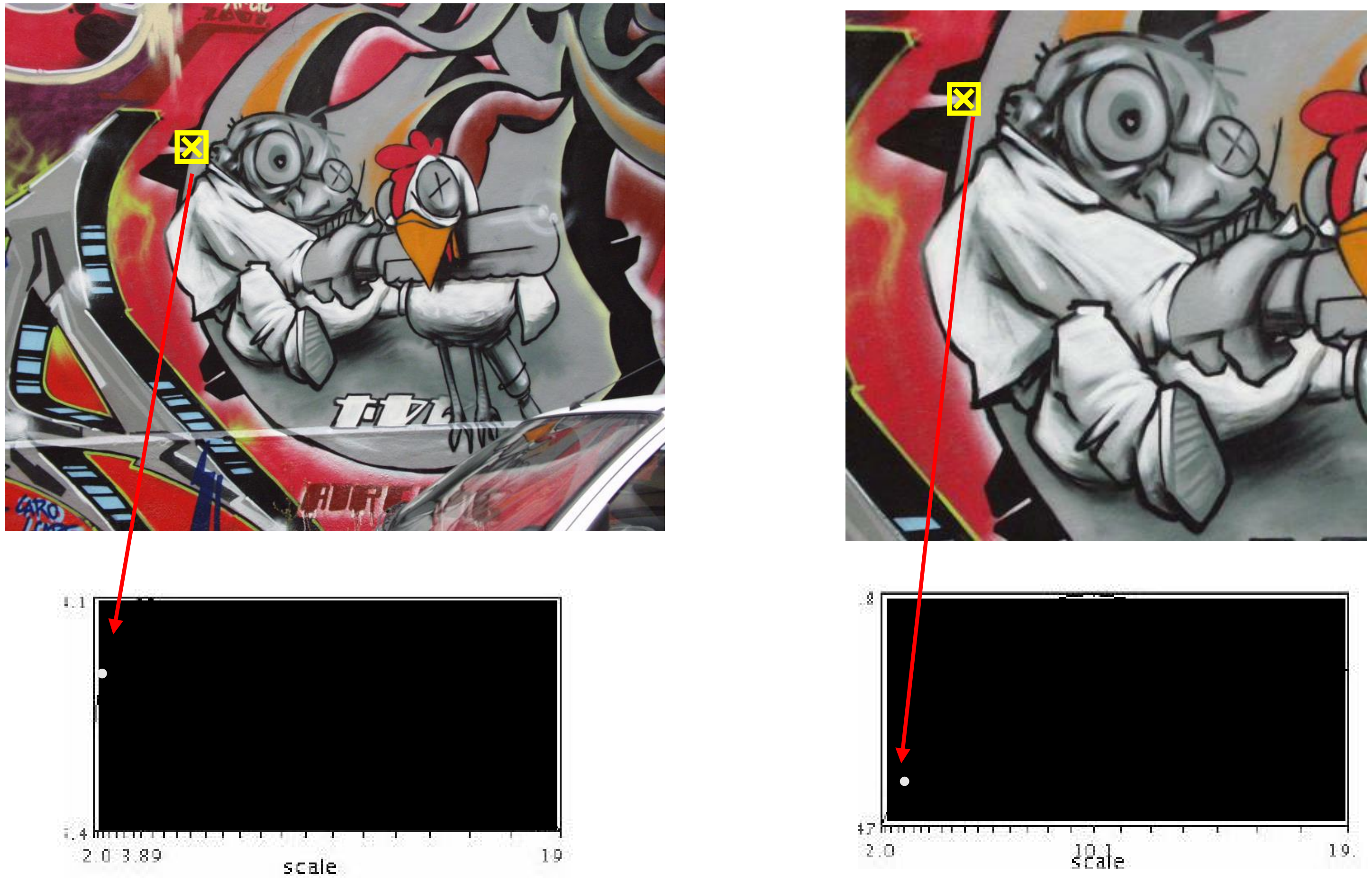
- Function responses for increasing scale (scale signature)



$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

Scale invariant feature

- Function responses for increasing scale (scale signature)

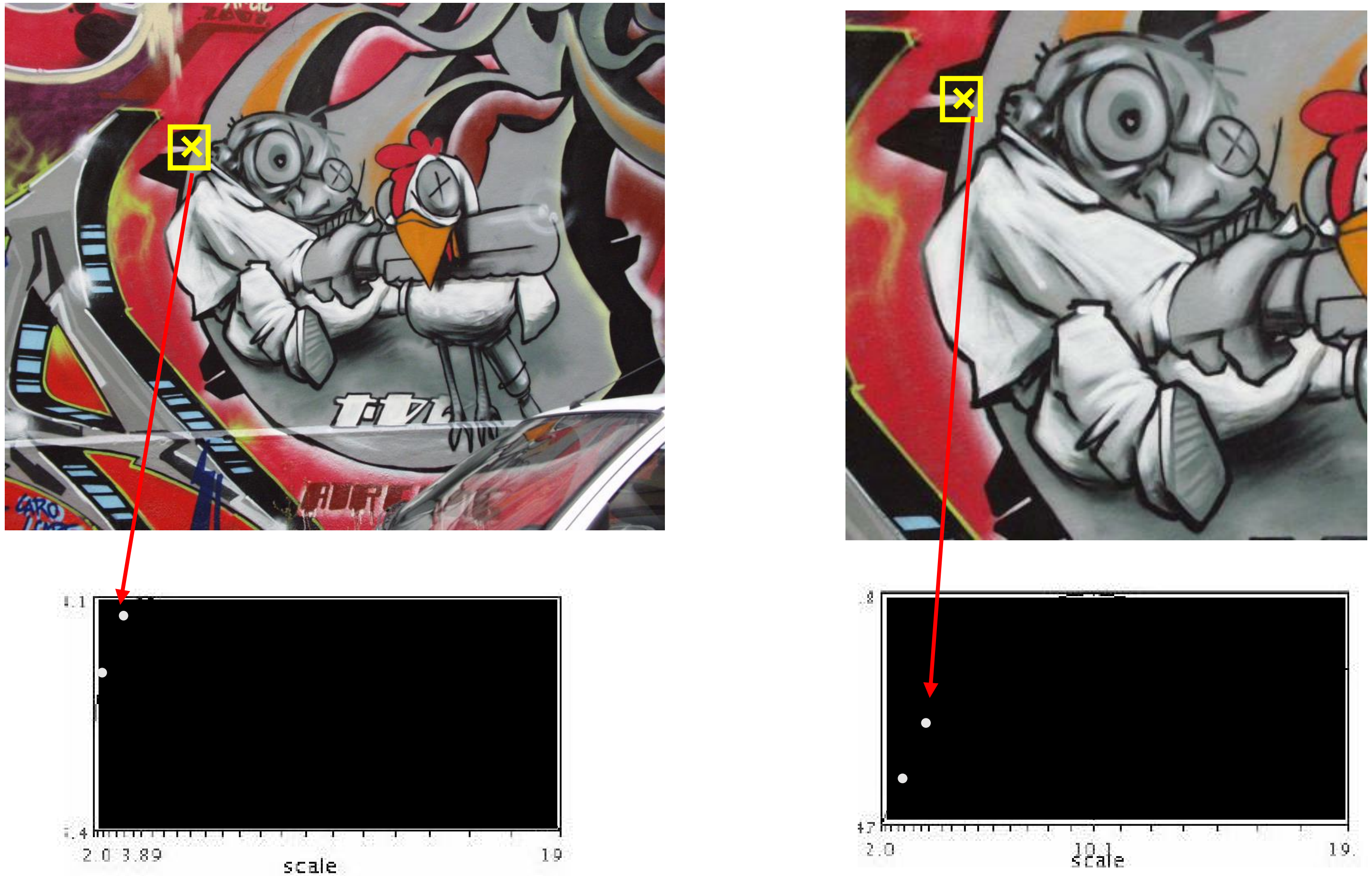


$f(I_{i_1 \dots i_m}(x, \sigma))$ $f(I_{i_1 \dots i_m}(x', \sigma))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)

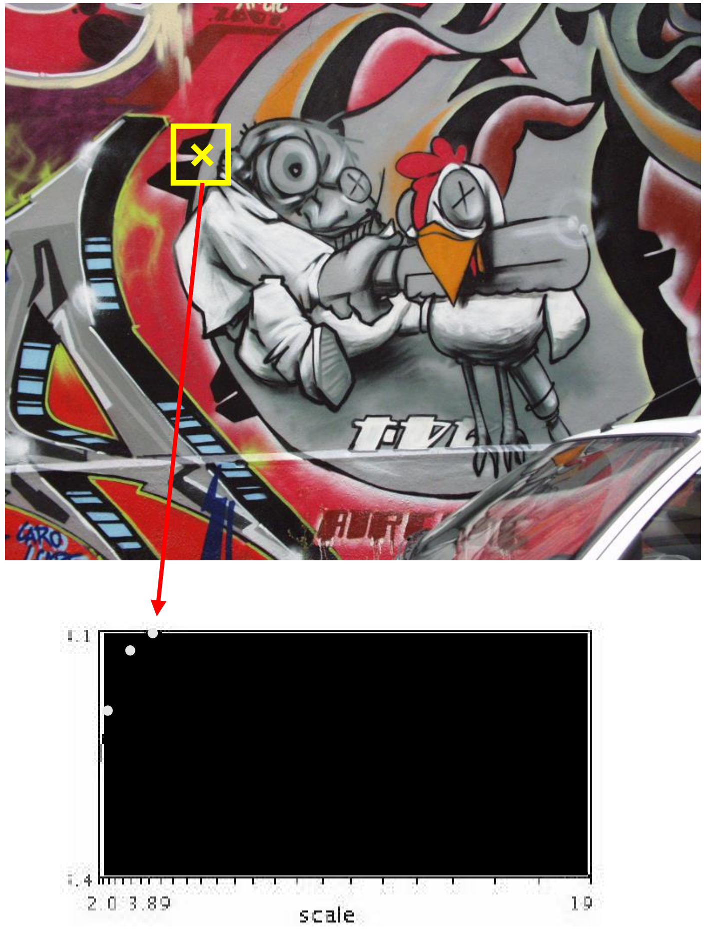


$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)



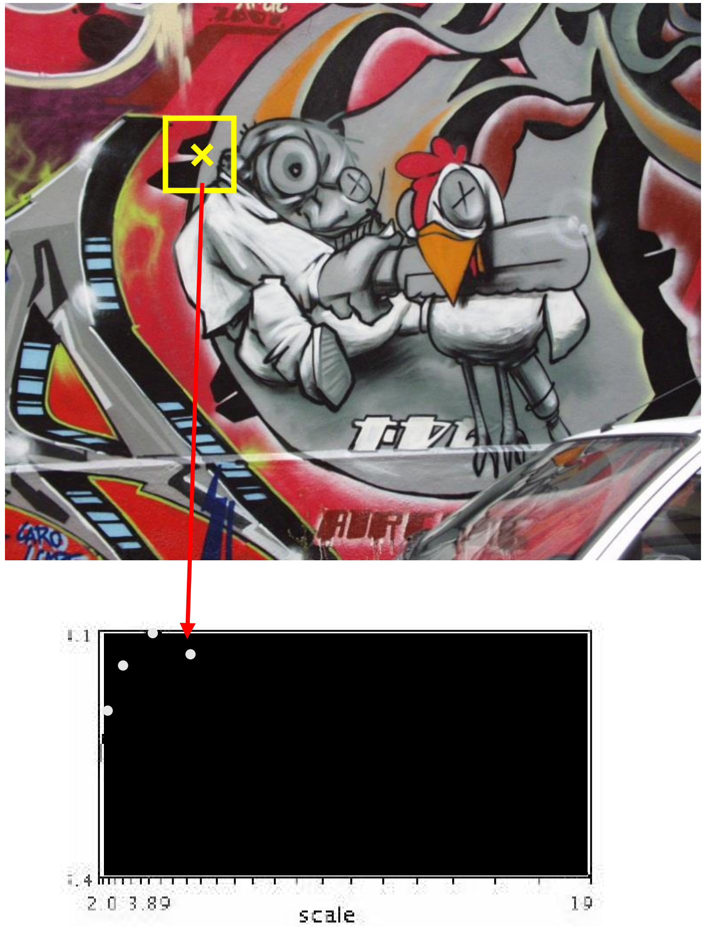
$f(I_{i_1 \ldots i_m}(x, \sigma))$

$f(I_{i_1 \ldots i_m}(x', \sigma))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)



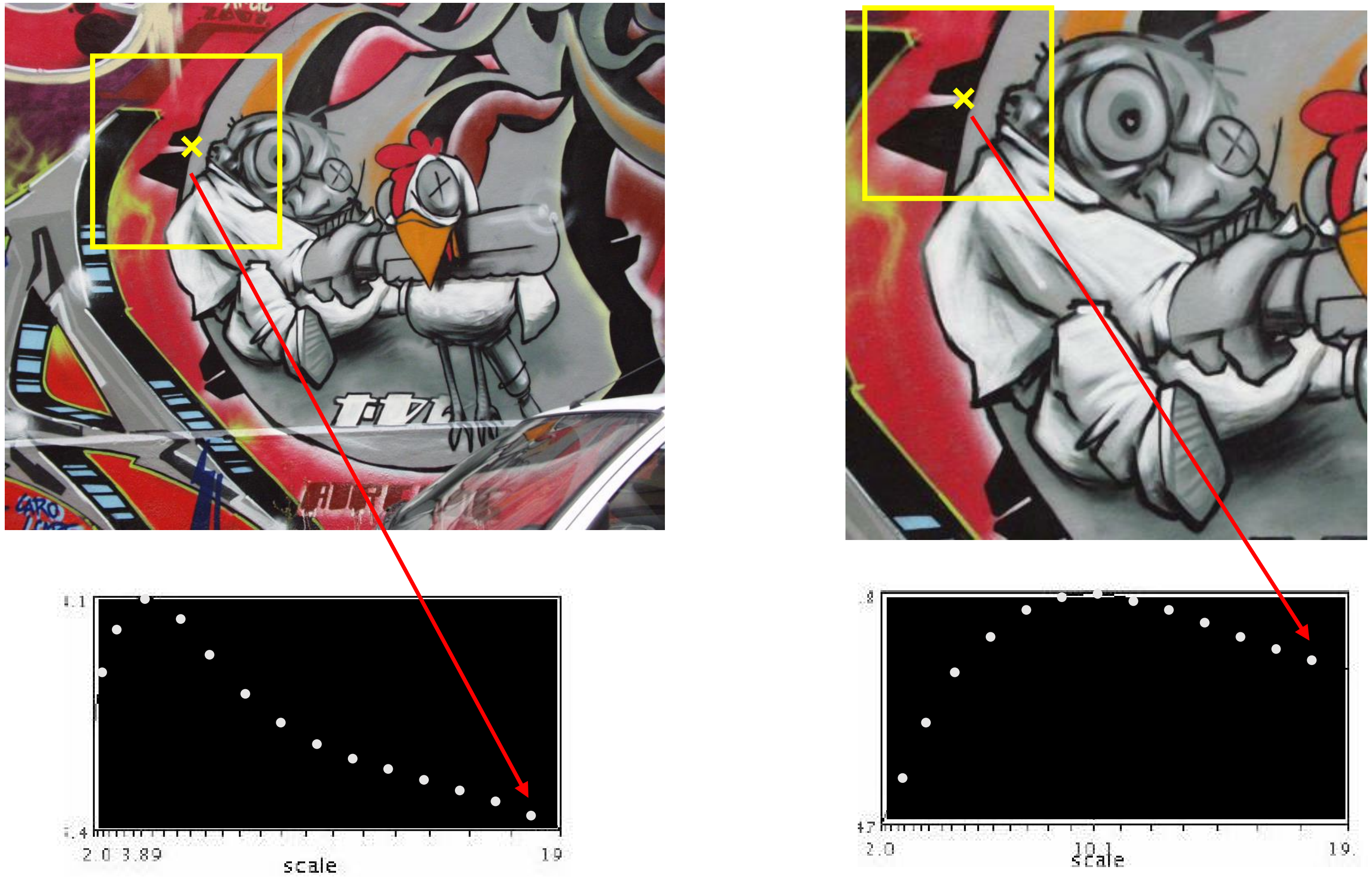
$f(I_{i_1 \ldots i_m}(x,\sigma))$

$f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)

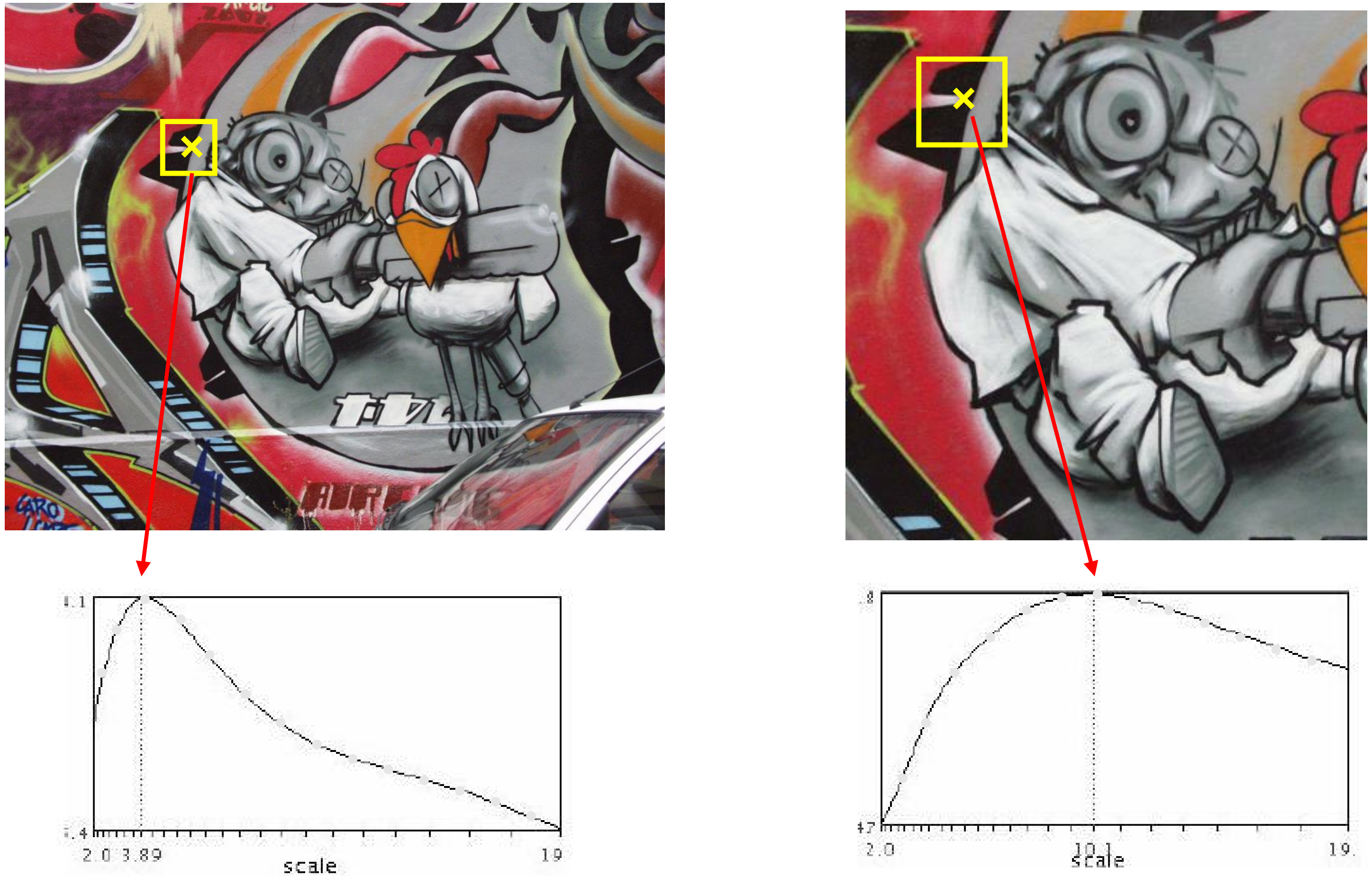


$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)

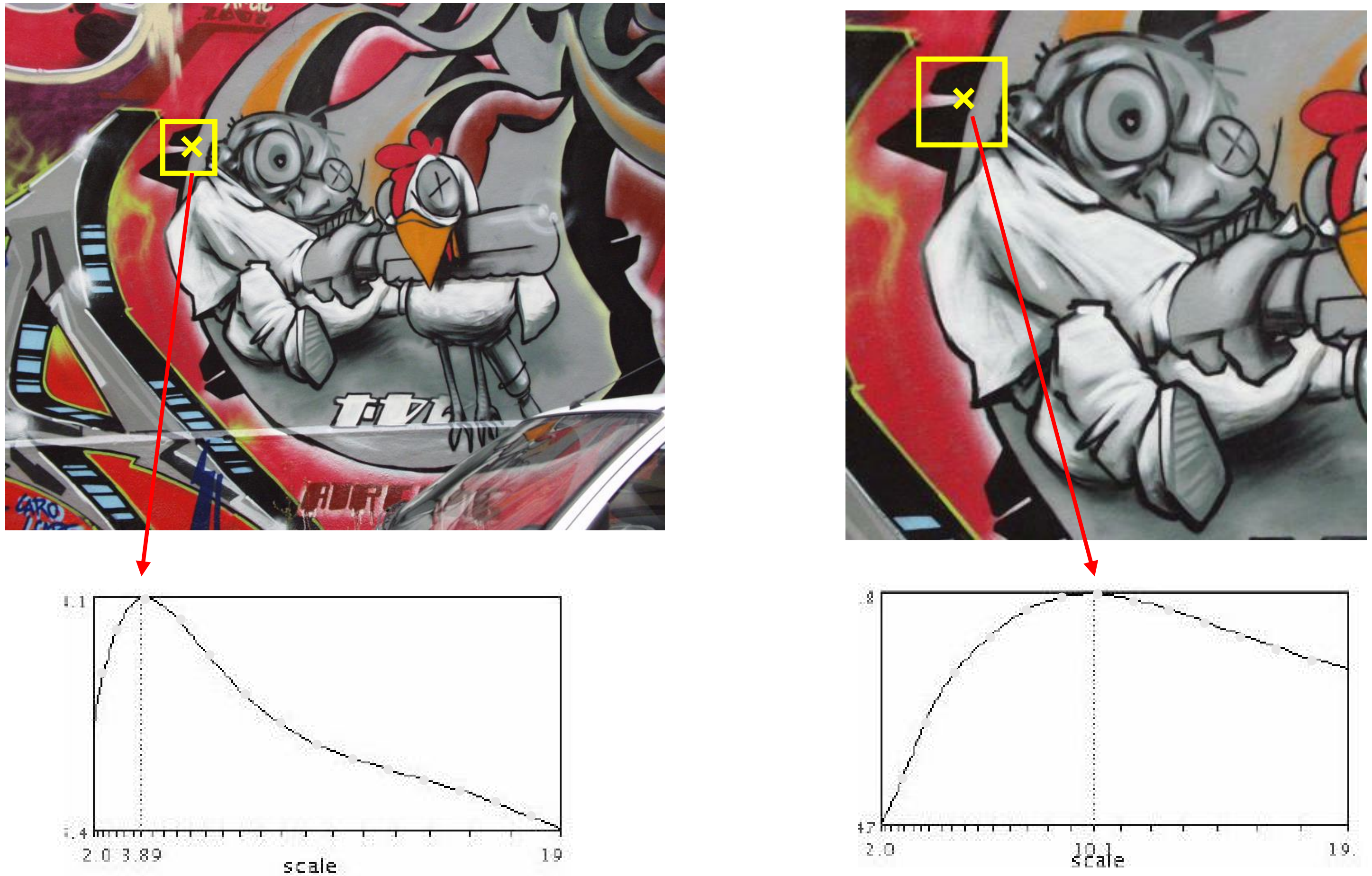


$f(I_{i_1 \ldots i_m}(x, \sigma))$ $f(I_{i_1 \ldots i_m}(x', \sigma'))$

K. Grauman, B. Leibe

Scale invariant feature

- Function responses for increasing scale (scale signature)

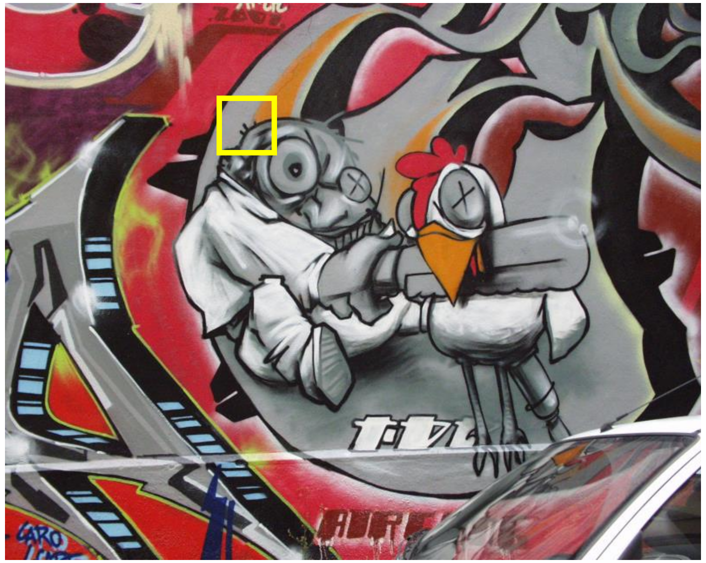


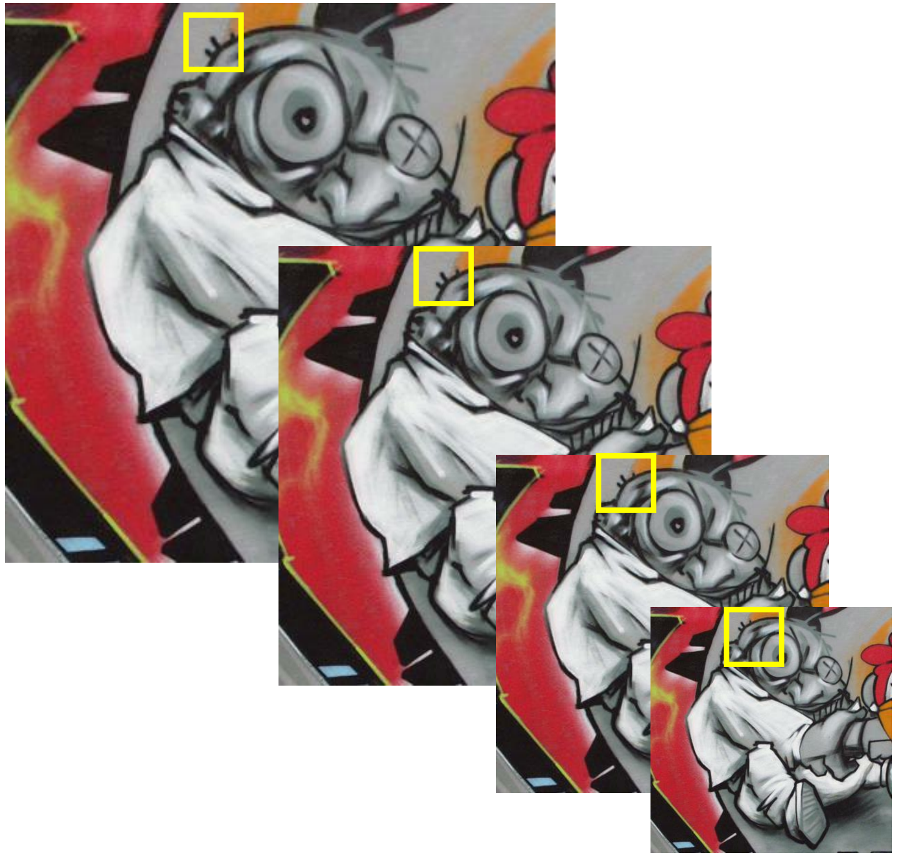
$f(I_{i_1 \dots i_m}(x, \sigma))$ $\quad$ $f(I_{i_1 \dots i_m}(x', \sigma'))$

K. Grauman, B. Leibe

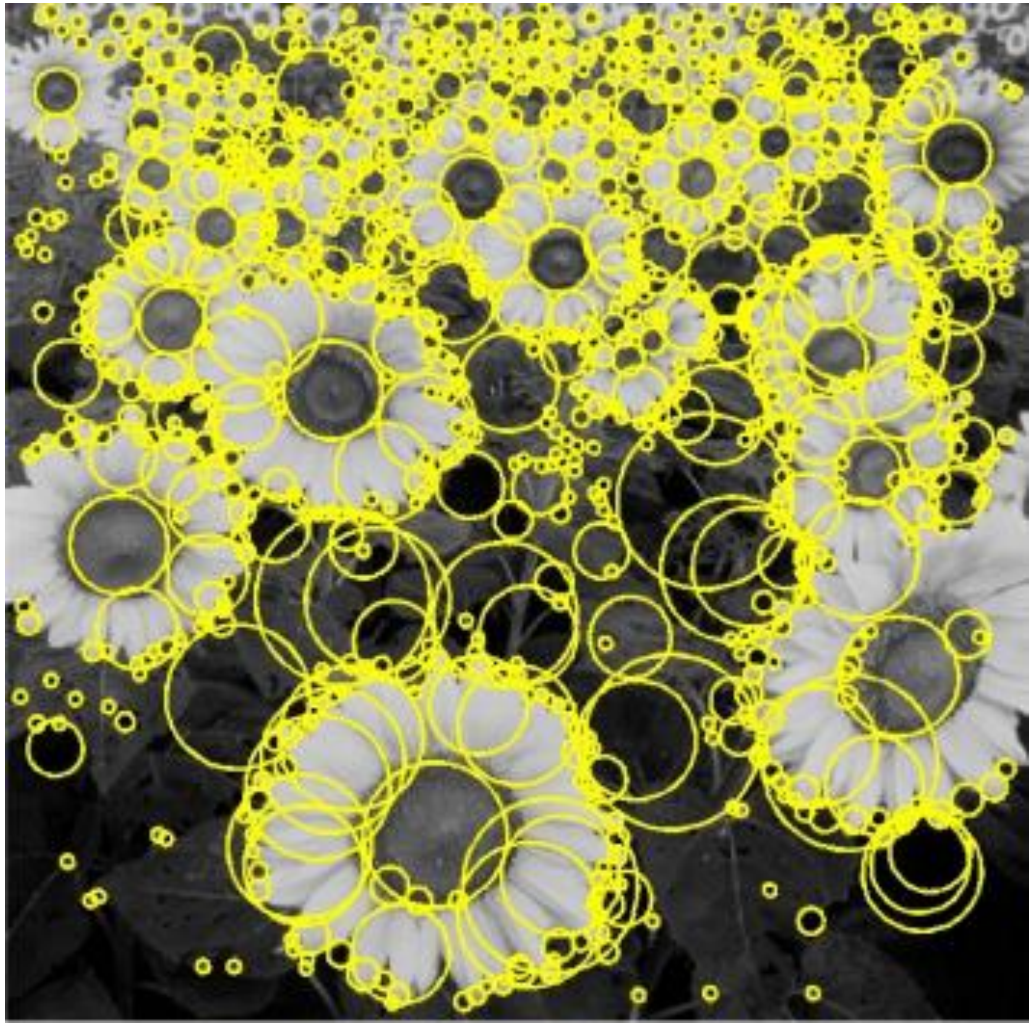
Image Pyramid

- Reducing the size of the image = increasing the size of the window.



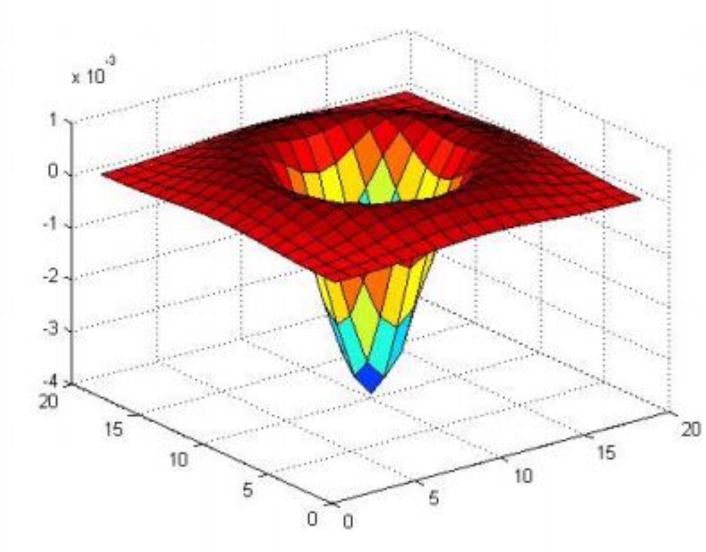


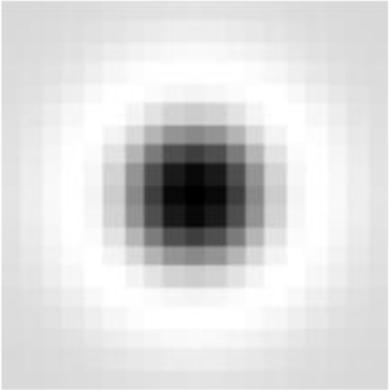
Blob detector



Blob detector

- Laplacian of Gaussian(LOG): Circularly symmetric operator for blob detection in 2D

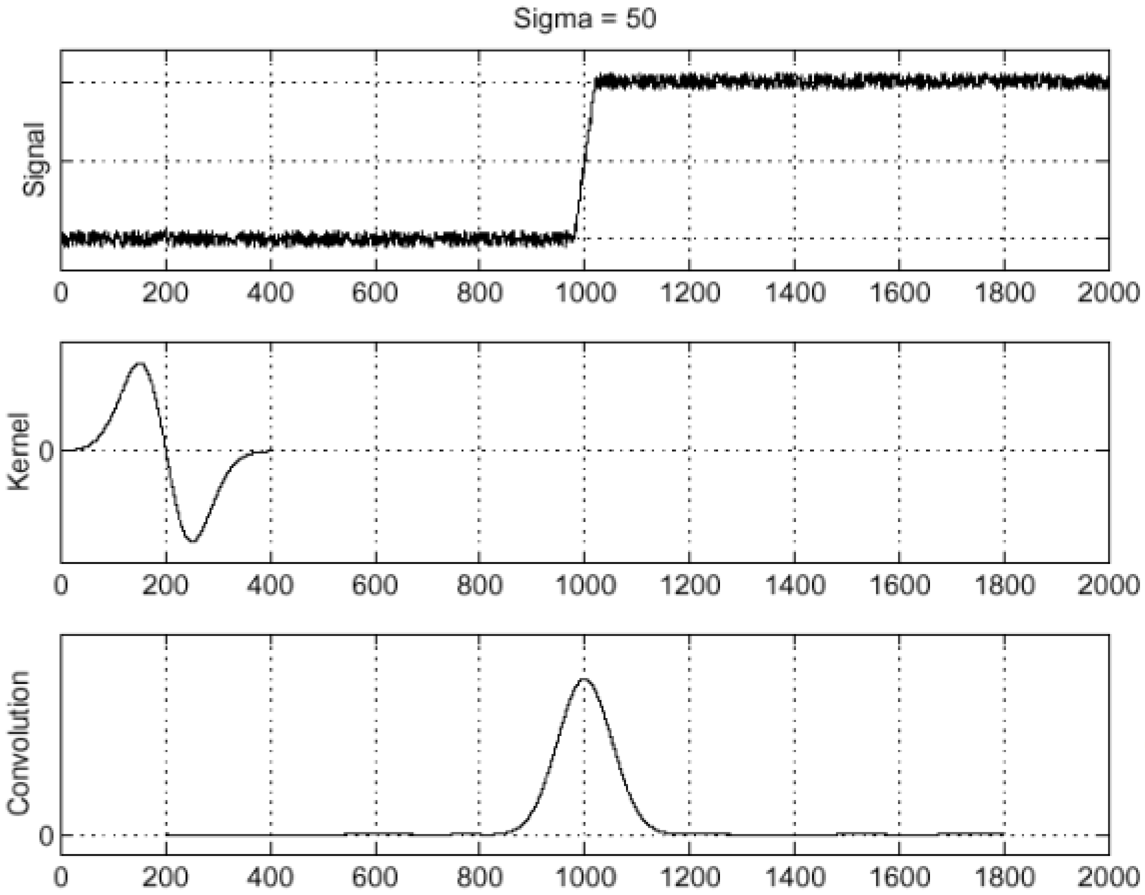




$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Blob detector

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D



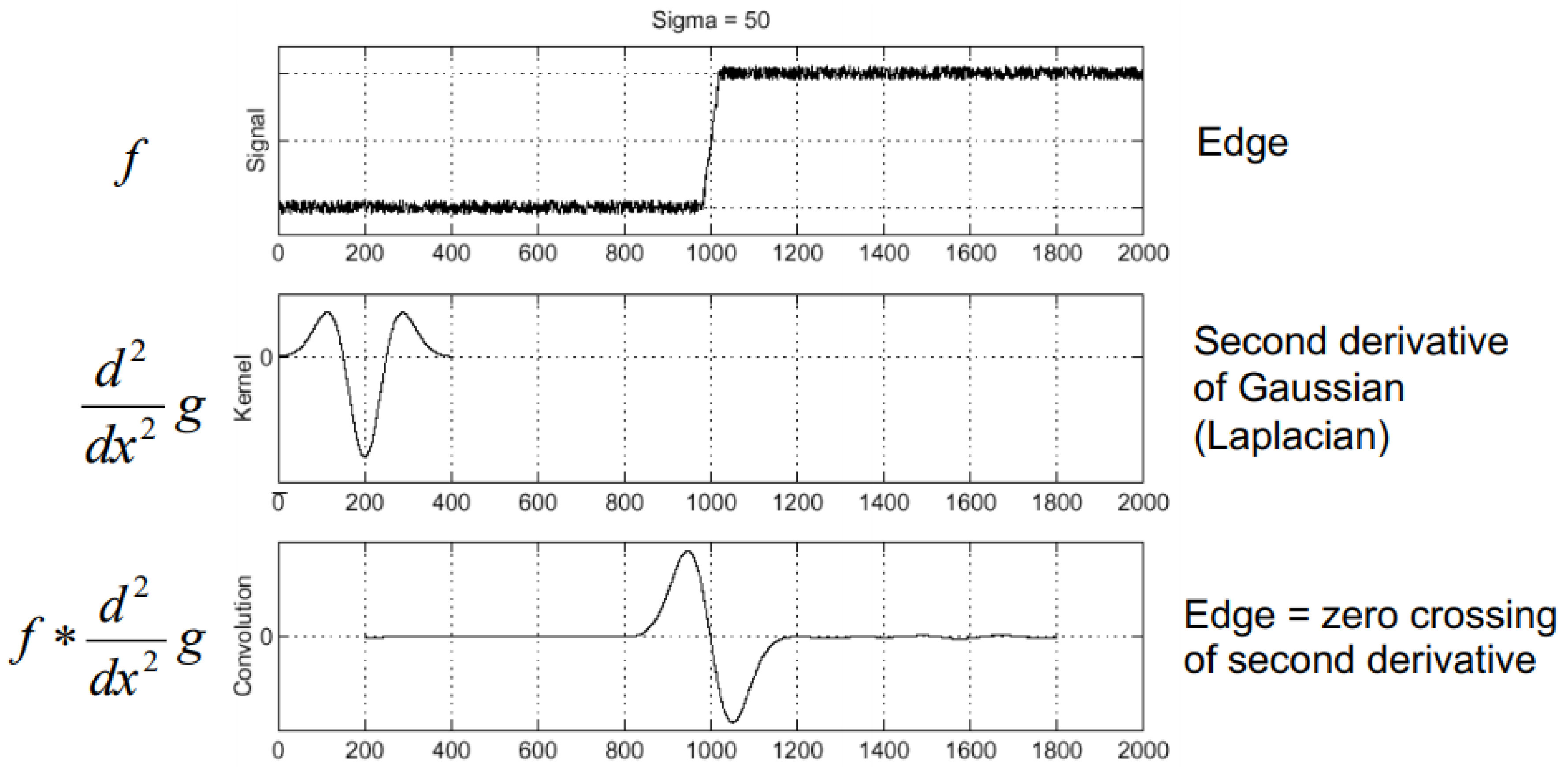
f — Edge

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

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

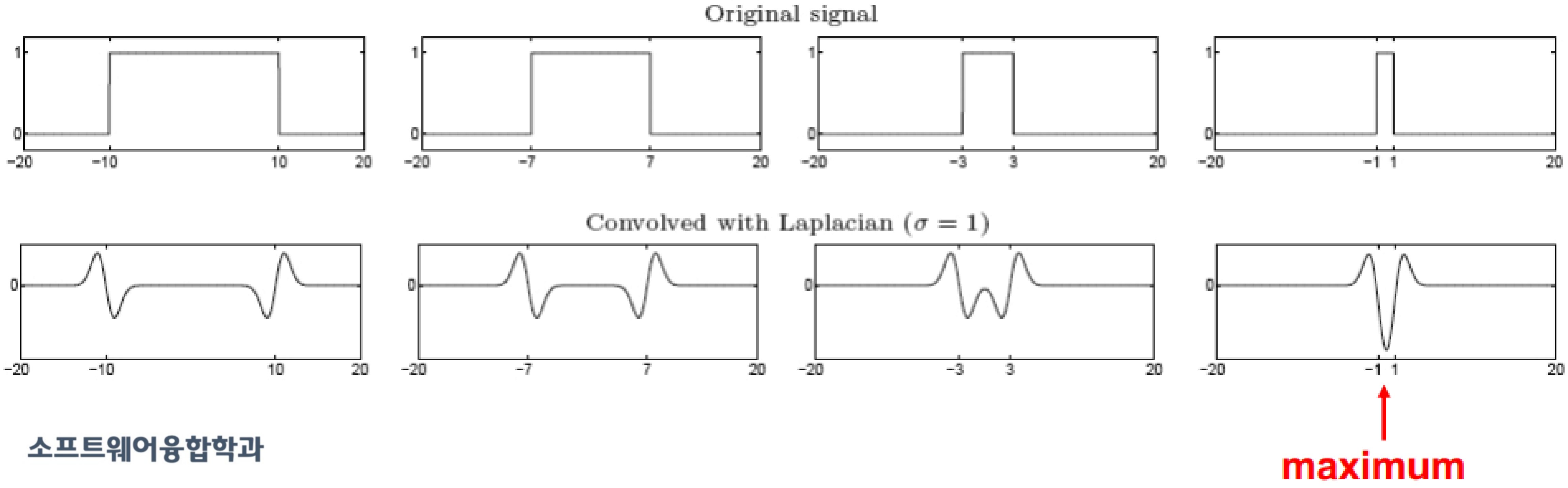
Blob detector

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D



Blob detector

- 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 is "matched" to the scale of the blob



Difference of Gaussian

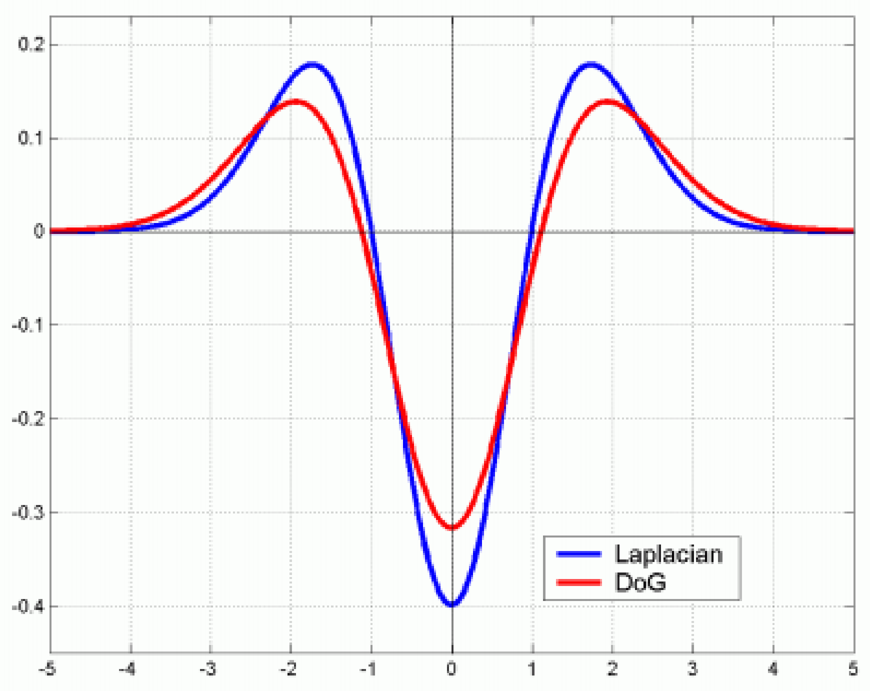
- Approximating the Laplacian with a difference of Gaussians:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

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

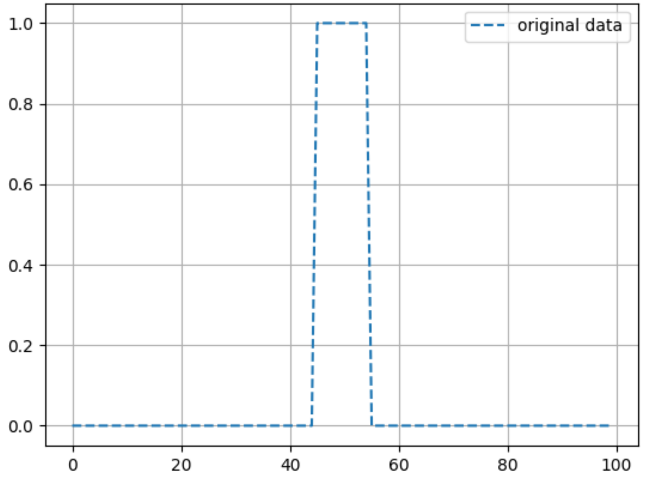
(Difference of Gaussians)



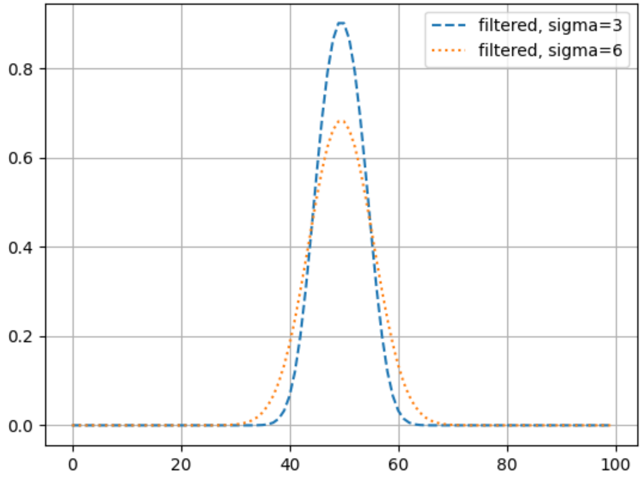
Blob detector

- Difference of Gaussian example

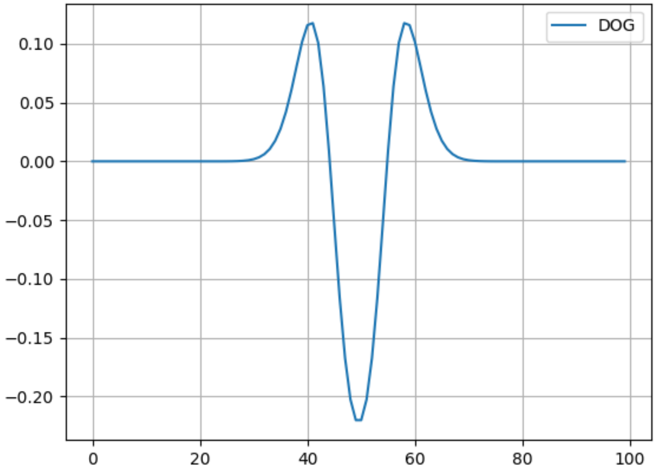
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import gaussian_filter1d

x = np.arange(100)
y = np.zeros(x.shape, np.float32)
y[45:55] = 1

y3 = gaussian_filter1d(y, 3)
y6 = gaussian_filter1d(y, 5)

plt.plot(y, '--', label='original data')
plt.plot(y3, '--', label='filtered, sigma=3')
plt.plot(y6, ':', label='filtered, sigma=6')
plt.plot(y6-y3, '-', label='DOG')

plt.grid()
plt.legend()
plt.show()
```



Input signal



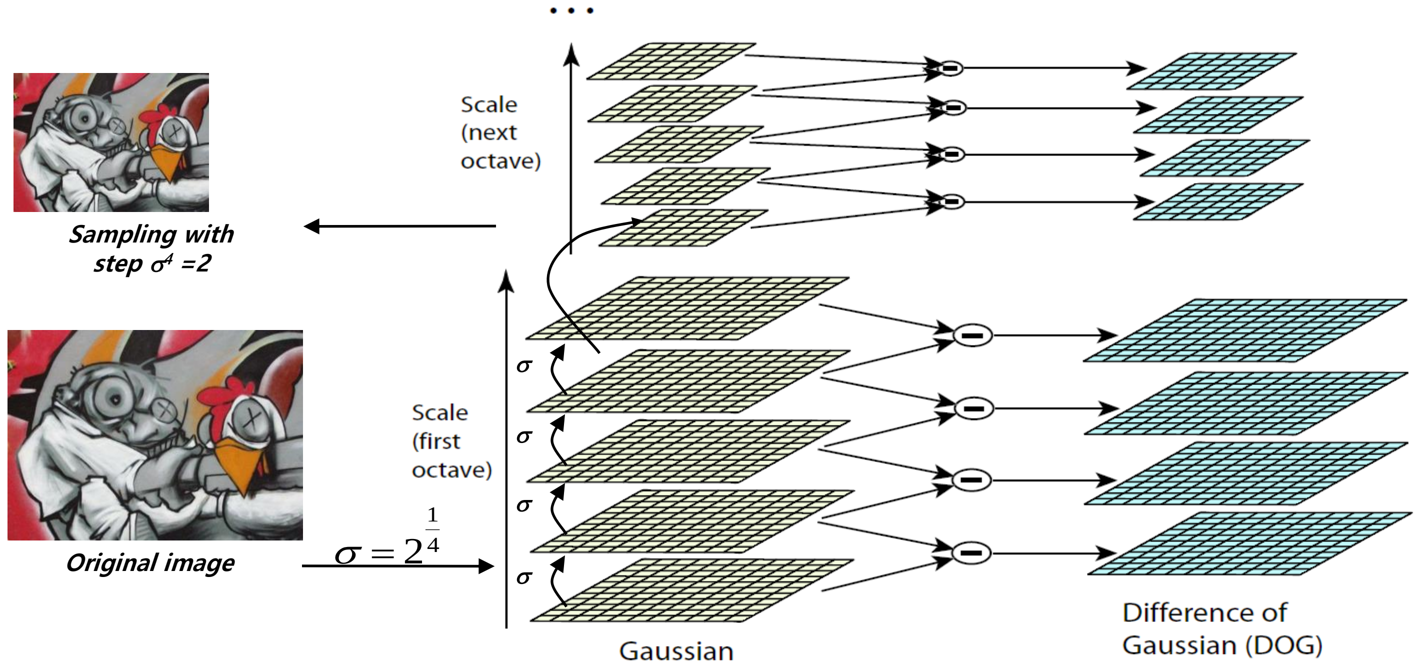
Gaussian filtering
Sigma = 3, 5



Difference of Gaussian

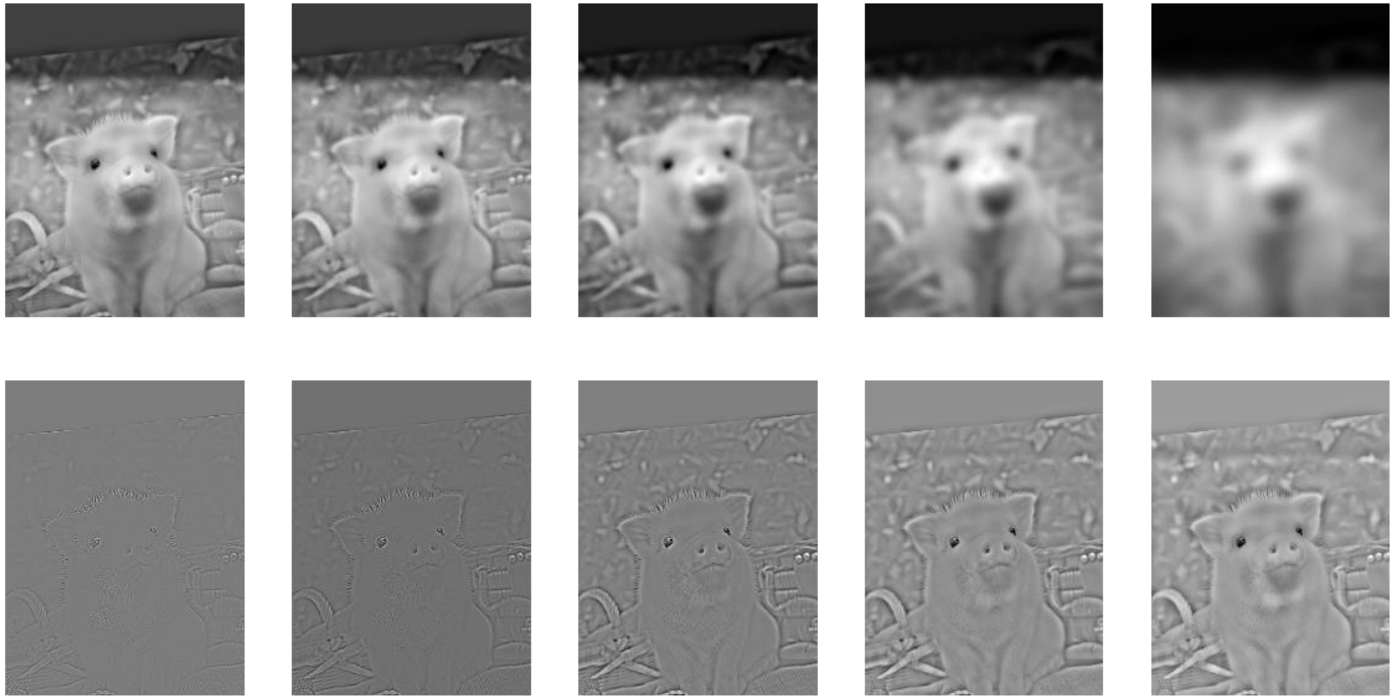
DoG – Efficient Computation

- Computation in Gaussian scale pyramid
  - Same region of different scale will be extracted by blob detection in image pyramid



Blob detector

- Difference of Gaussian
  - Difference of Gaussians is a feature enhancement algorithm that involves the subtraction of one Gaussian blurred version of an original image from another, less blurred version of the original.



Thank you