CAP 4453 Robot Vision

Dr. Gonzalo Vaca-Castaño

gonzalo.vacacastano@ucf.edu

Administrative details

- Issues submitting homework

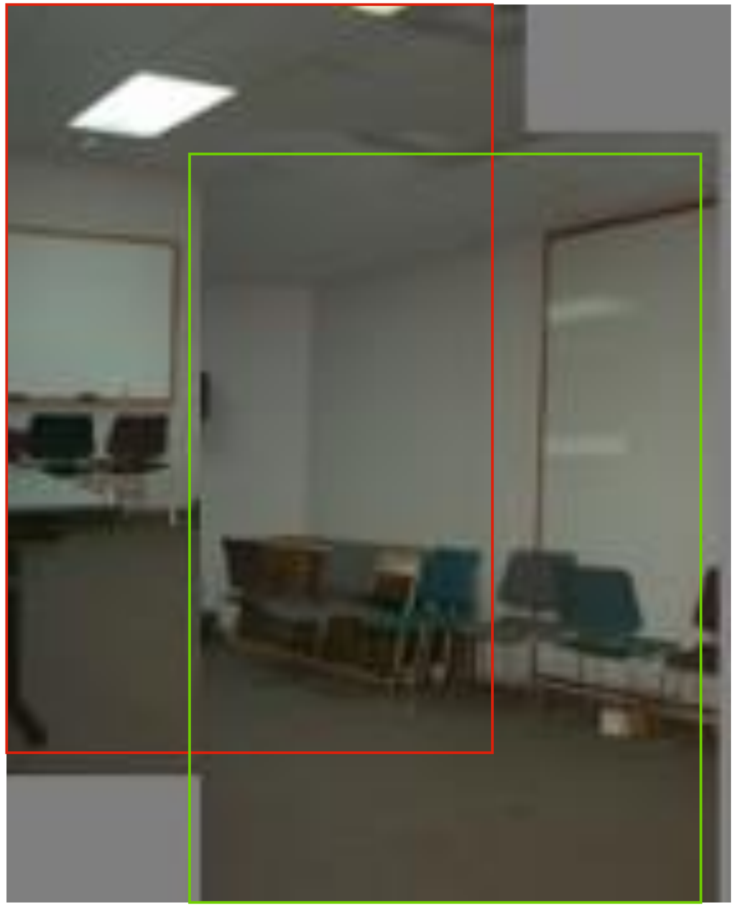
Credits

- Slides comes directly from:
  - Ioannis (Yannis) Gkioulekas (CMU)
  - Kris Kitani.
  - Fredo Durand (MIT).
  - James Hays (Georgia Tech).
  - Yogesh S Rawat (UCF)
  - Noah Snavely (Cornell)

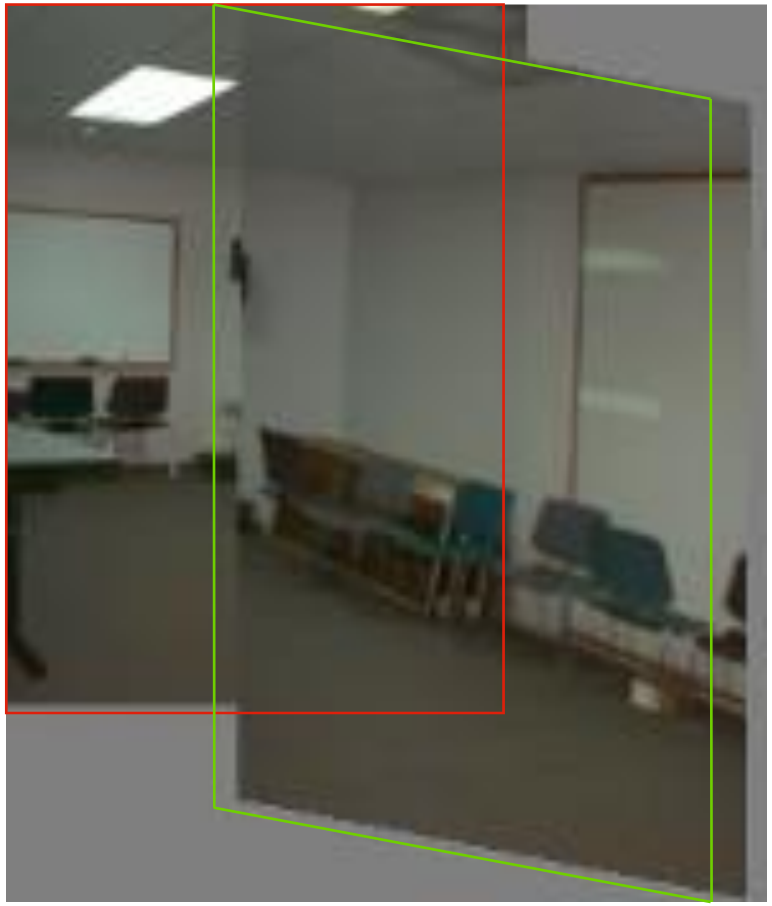
Short Review from last class

Warping with different transformations

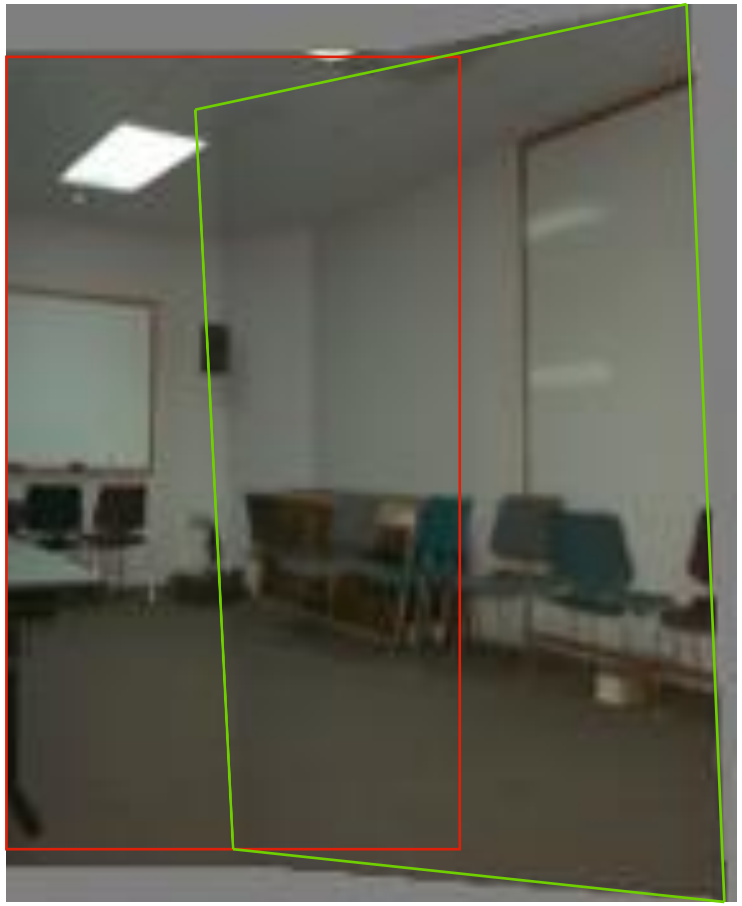
translation



affine



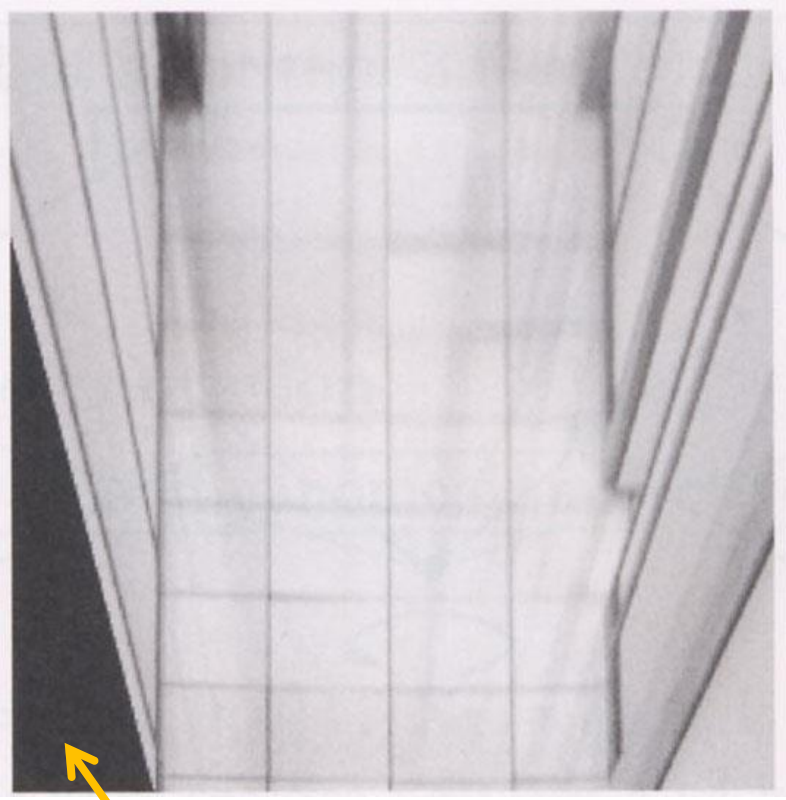
pProjective (homography)



View warping

original view

synthetic top view



synthetic side view

What are these black areas near the boundaries?

Virtual camera rotations

original view

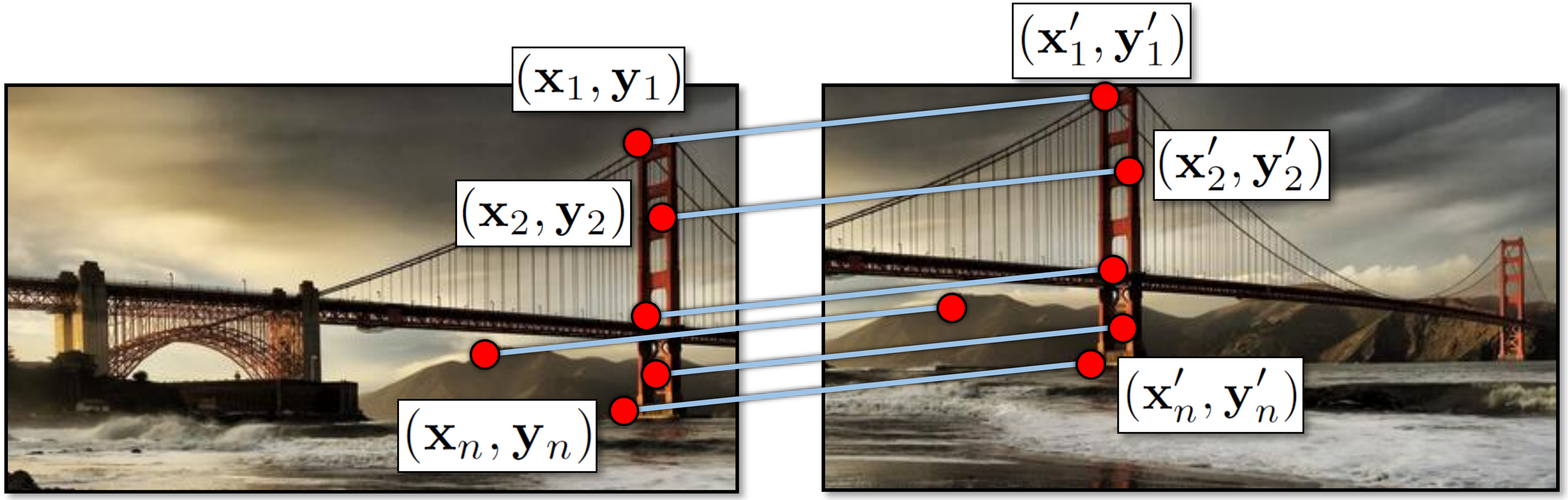
synthetic rotations

Image rectification

two original images

rectified and stitched

Image warping



Recap: Two Common Optimization Problems

Problem statement

$$\text{minimize} \quad \|\mathbf{Ax}-\mathbf{b}\|^2$$

$$\text{least squares solution to } \mathbf{Ax}=\mathbf{b}$$

Solution

$$\mathbf{x}=\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}$$

```
import numpy as np
x,resid,rank,s = np.linalg.lstsq(A,b)
```

Problem statement

$$\text{minimize} \quad \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} \quad \text{s.t. } \mathbf{x}^T\mathbf{x}=1$$

$$\text{non-trivial lsq solution to } \mathbf{Ax}=0$$

Solution

$$[\mathbf{v},\lambda]=\text{eig}(\mathbf{A}^T\mathbf{A})$$

$$\lambda_1<\lambda_{2..n}:\mathbf{x}=\mathbf{v}_1$$

Affine transformations

- Matrix form

$$
\underbrace{\begin{bmatrix}
x_1 & y_1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_1 & y_1 & 1 \\
x_2 & y_2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2 & y_2 & 1 \\
 & & & \vdots & & \\
x_n & y_n & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_n & y_n & 1
\end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 6}}
\underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\substack{\mathbf{t} \\ 6 \times 1}}
=
\underbrace{\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}}_{\substack{\mathbf{b} \\ 2n \times 1}}
$$

Solving for homographies

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\underset{2n \times 9}{\mathbf{A}} \qquad \underset{9}{\mathbf{h}} \qquad \underset{2n}{\mathbf{0}}$$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points

Recap: Two Common Optimization Problems

Problem statement

$$\text{minimize} \quad \left\|\mathbf{Ax}-\mathbf{b}\right\|^2$$

$$\text{least squares solution to } \mathbf{Ax}=\mathbf{b}$$

Solution

$$\mathbf{x}=\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}$$

```
import numpy as np
x,resid,rank,s = np.linalg.lstsq(A,b)
```

Problem statement

$$\text{minimize} \quad \mathbf{x}^T\mathbf{A}^T\mathbf{Ax} \quad \text{s.t. } \mathbf{x}^T\mathbf{x}=1$$

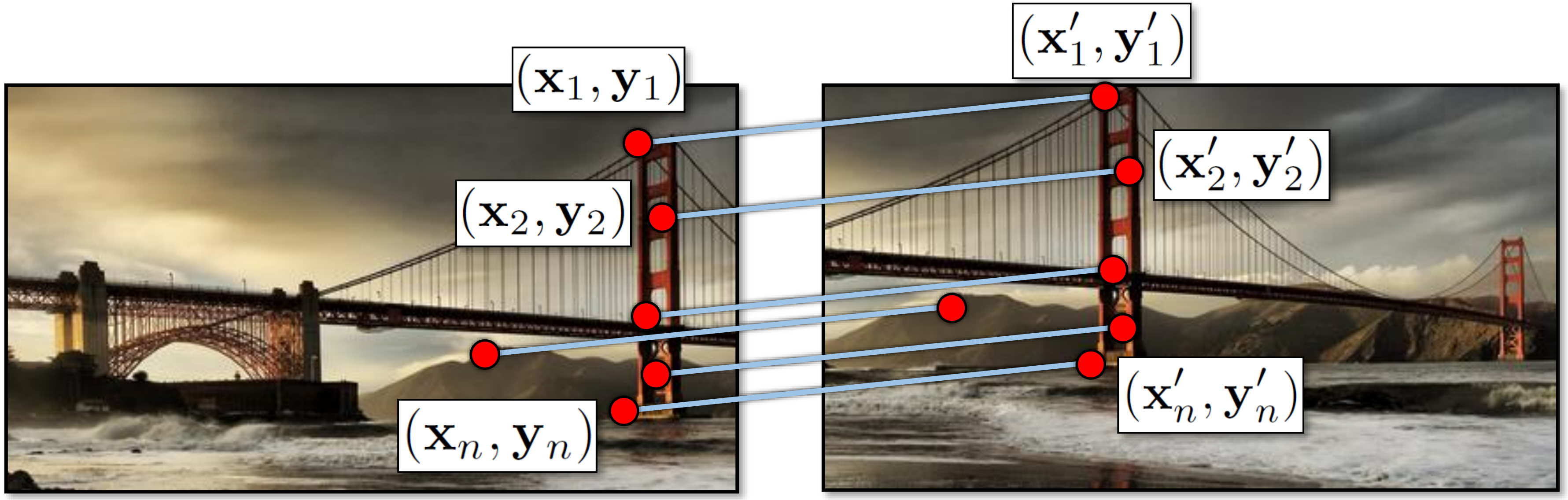
$$\text{non-trivial lsq solution to } \mathbf{Ax}=0$$

Solution

$$[\mathbf{v},\lambda]=\operatorname{eig}(\mathbf{A}^T\mathbf{A})$$

$$\lambda_1<\lambda_{2..n}:\mathbf{x}=\mathbf{v}_1$$

Image warping



How do we find point correspondences automatically?

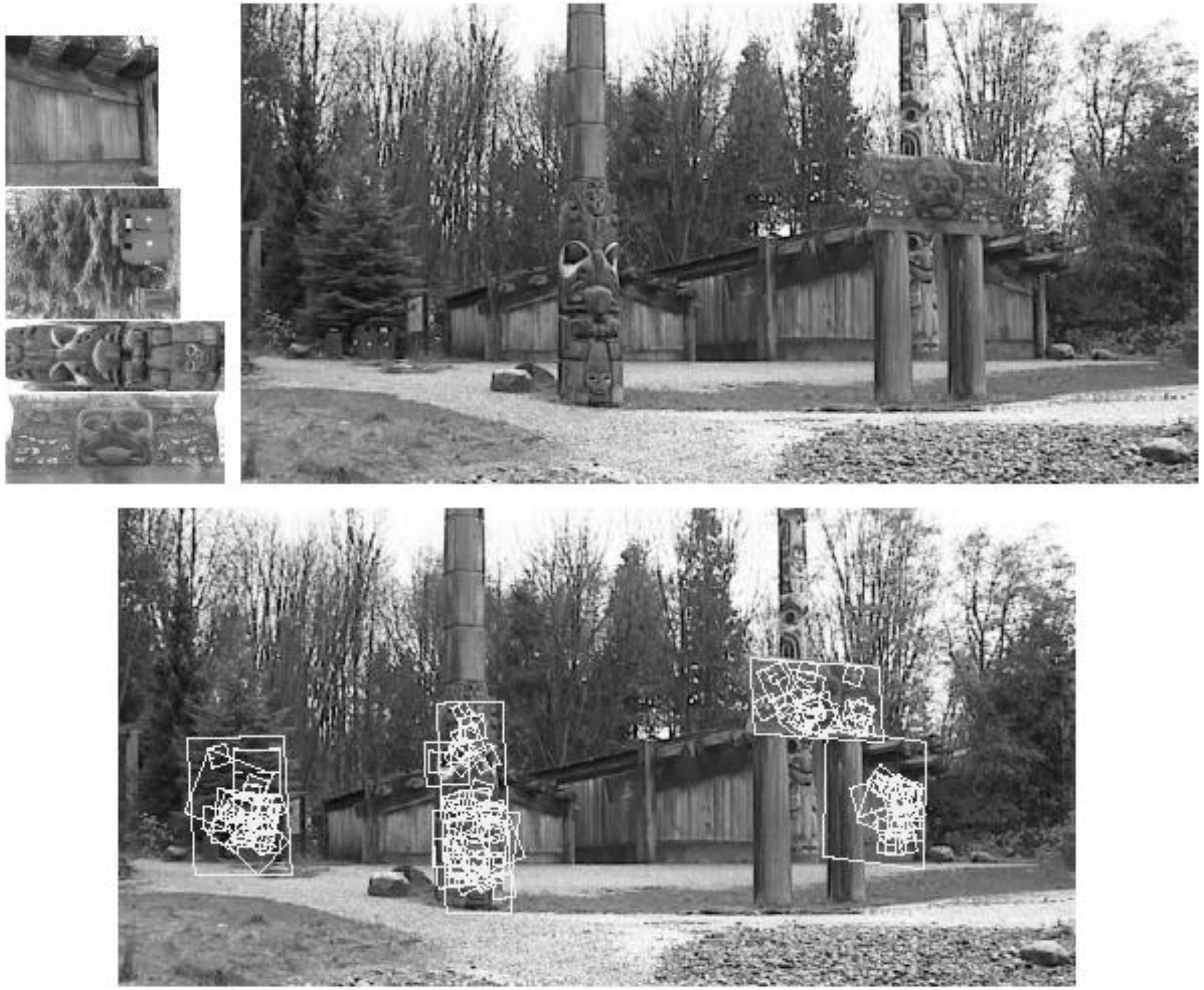
Robot Vision

11. Feature points detection

Outline

- Motivation
- Detecting key points
  - Harris corner detector
  - Blob detection

Location Recognition



Robot Localization

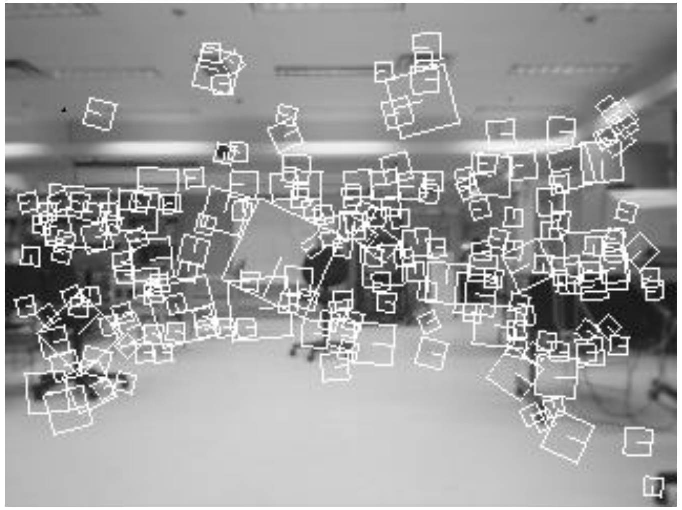
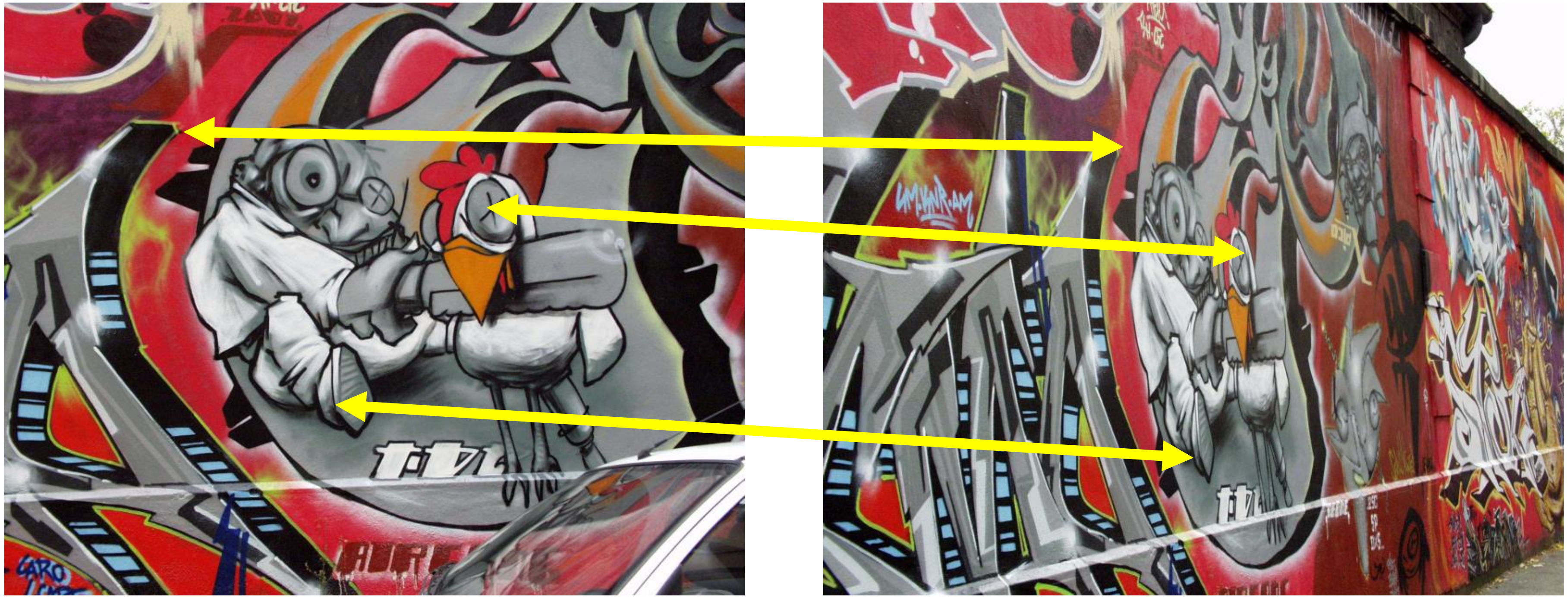


Image matching

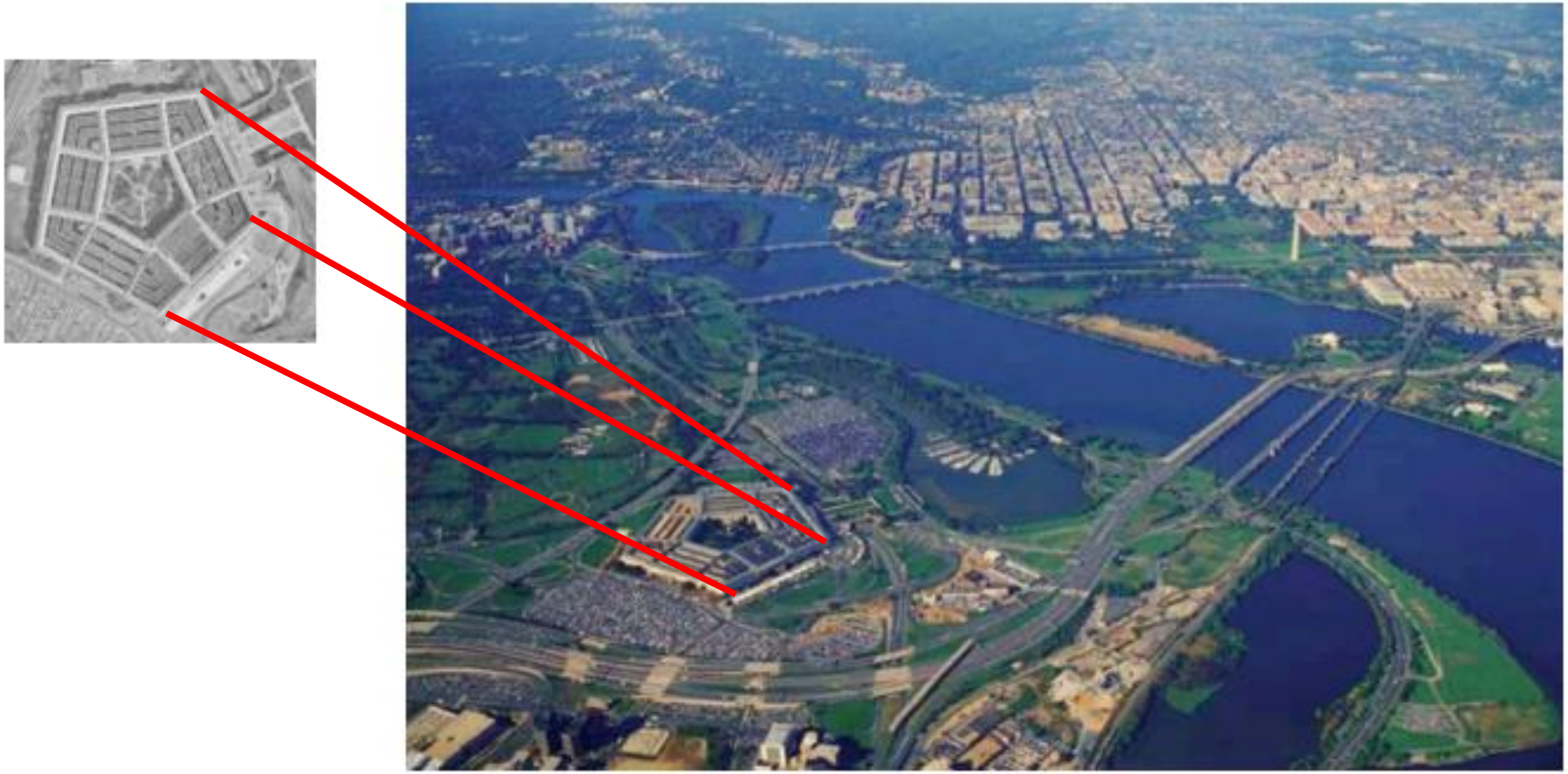
Structure from motion

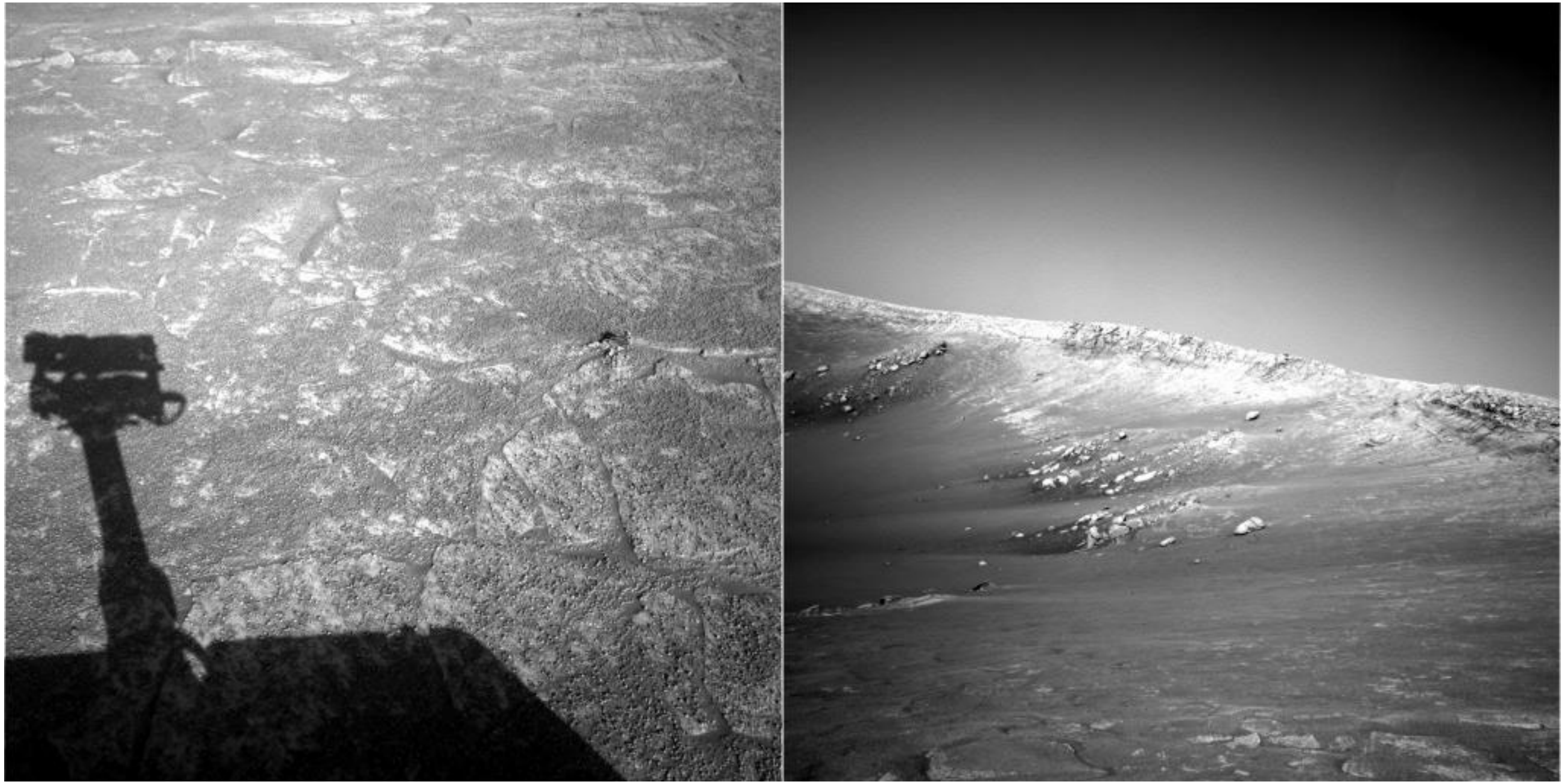
3D photosynth

Image matching



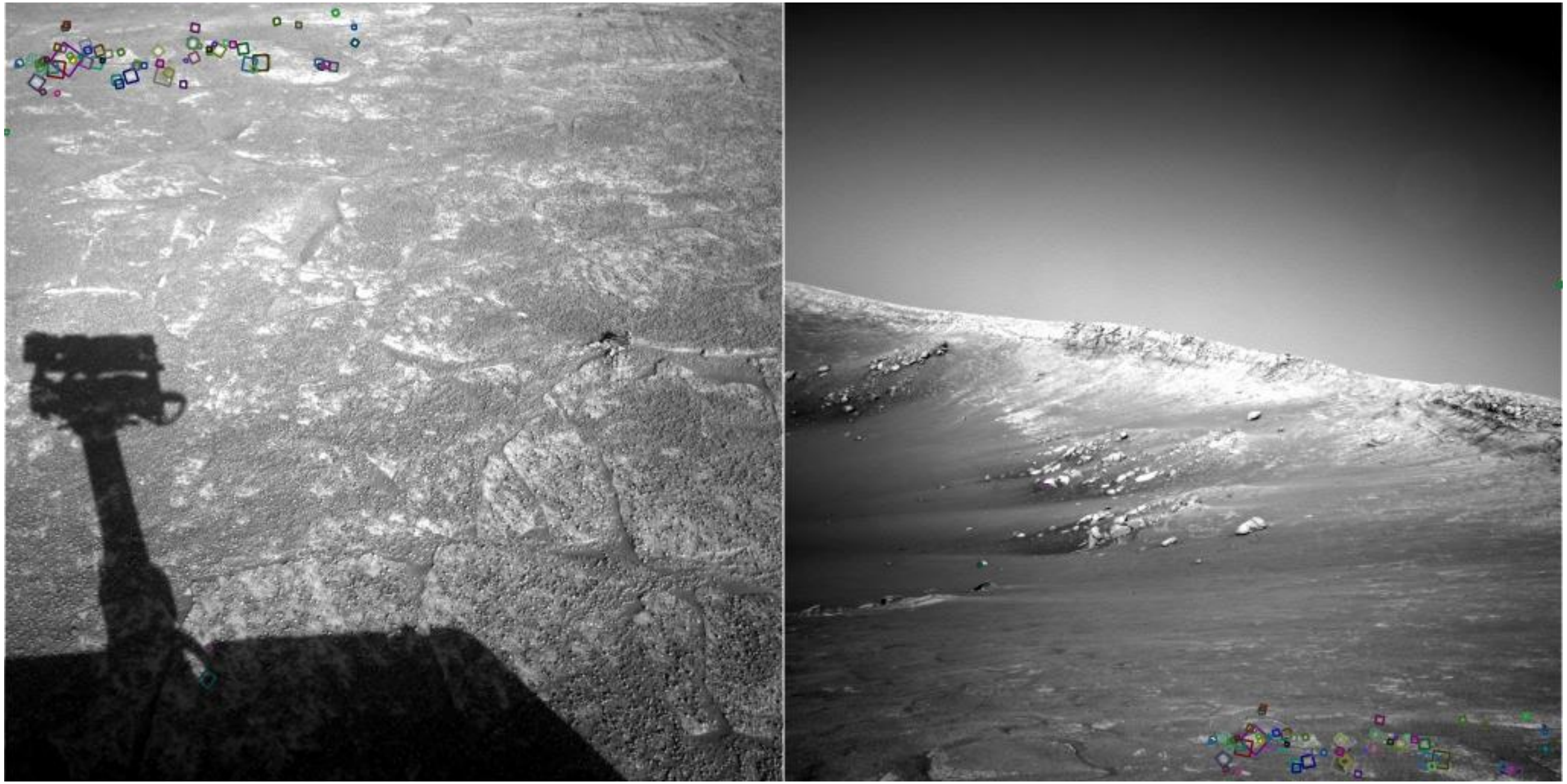
Matching



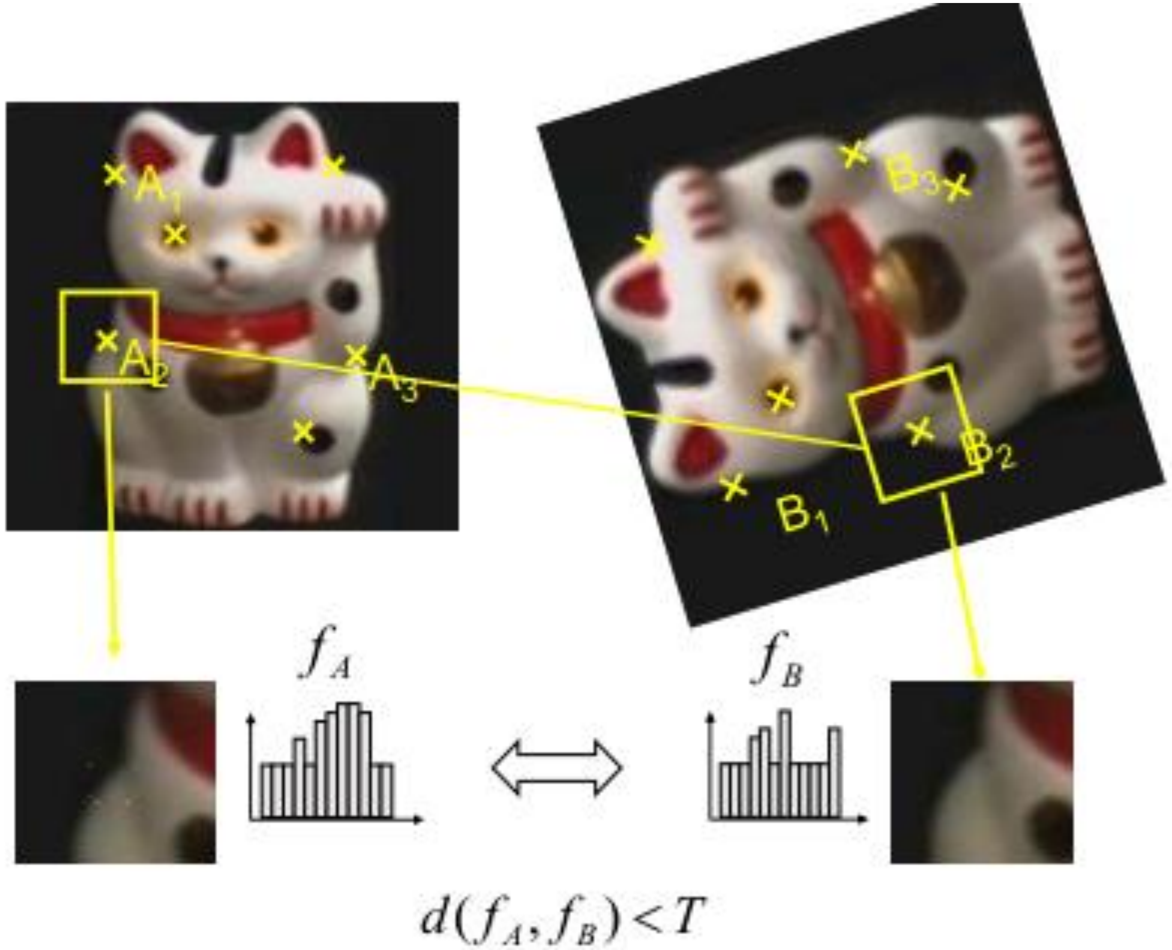


NASA Mars Rover images

Where are the corresponding points?



Application: KeyPoint Matching



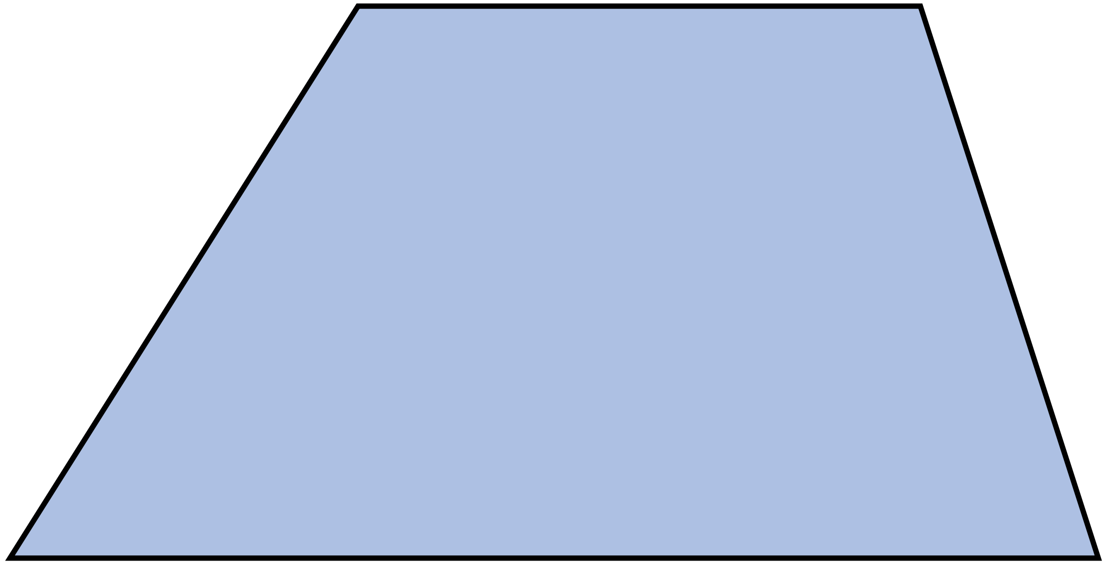
1. Find a set of distinctive key-points
2. Define a region around each key-point
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

K. Grauman, B. Leibe

Finding interest points

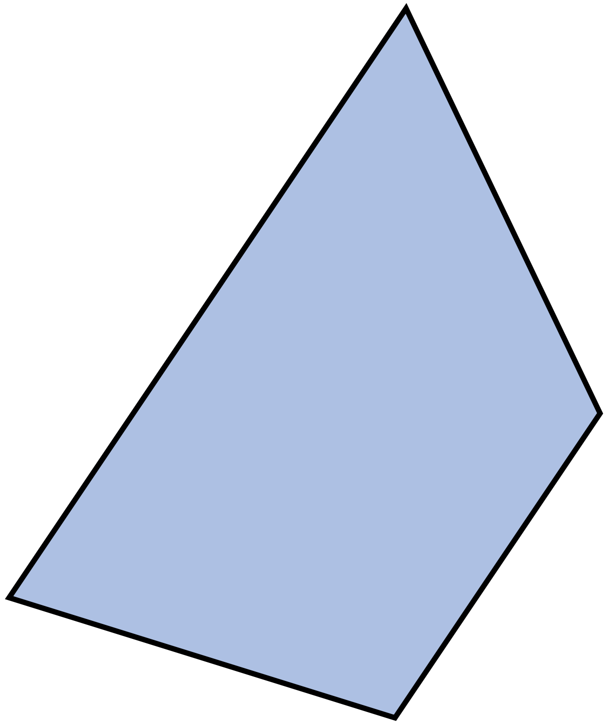
The aperture problem

- Individual pixels are ambiguous
- Idea: Look at whole patches!



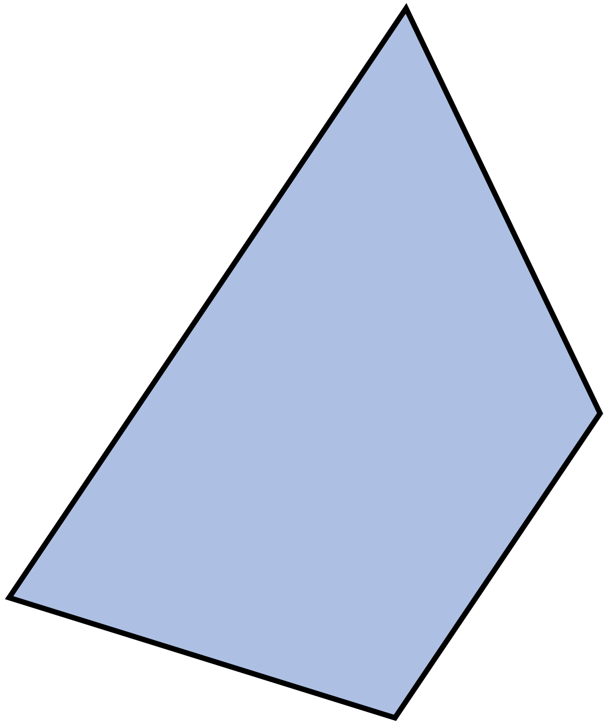
Pick a point in the image.
Find it again in the next image.

What type of feature would you select?



Pick a point in the image.
Find it again in the next image.

What type of feature would you select?

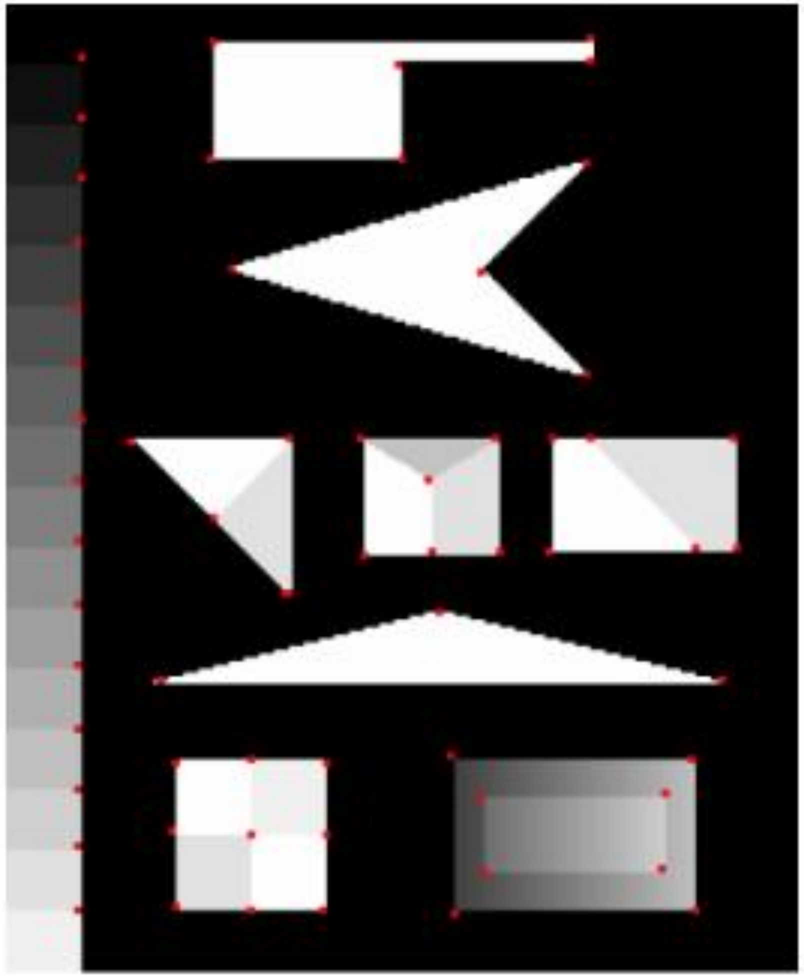


Pick a point in the image.
Find it again in the next image.

What type of feature would you select?

a corner

What is an interest point?



Properties of interest points algorithm

- Detect all (or most) true interest points
- No false interest points
- Well localized
- Robust with respect to noise
- Efficient detection
- **Detect points that are repeatable and distinctive**

Outline

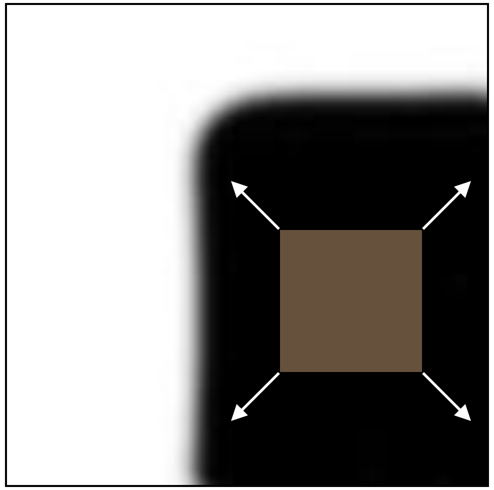
- Motivation
- Detecting key points
  - **Harris corner detector**
  - Blob detection

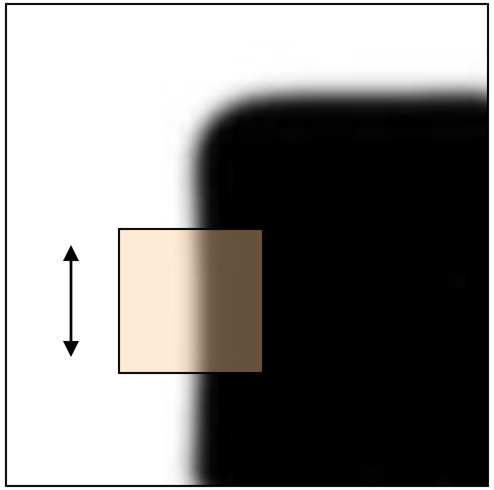
Corner detection: Possible approaches

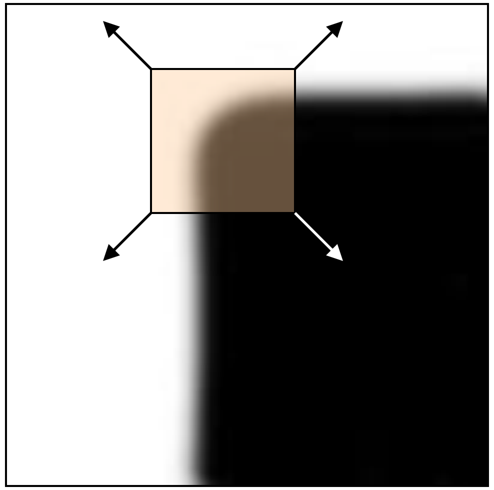
- Based on brightness of images
  - Usually image derivatives
- Based on boundary extraction
  - First step edge detection
  - Curvature analysis of edges

Corner Detection: 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

Source: A. Efros

Harris corner detector

A COMBINED CORNER AND EDGE DETECTOR

Chris Harris & Mike Stephens

Plessey Research Roke Manor, United Kingdom

Consistency of image edge filtering is of prime importance for 3D interpretation of image sequences using feature tracking algorithms. To cater for image regions containing texture and isolated features, a combined corner and edge detector based on the local auto-correlation function is utilised, and it is shown to perform with good consistency on natural imagery.

INTRODUCTION

The problem we are addressing in Alvey Project MMI149 is that of using computer vision to understand the unconstrained 3D world, in which the viewed scenes will in general contain too wide a diversity of objects for top-down recognition techniques to work. For example, we desire to obtain an understanding of natural scenes, containing roads, buildings, trees, bushes, etc., as typified by the two frames from a sequence illustrated in Figure 1. The solution to this problem that we are pursuing is to use a computer vision system based upon motion analysis of a monocular image sequence from a mobile camera. By extraction and tracking of image features, representations of the 3D analogues of these features can be constructed.

To enable explicit tracking of image features to be performed, the image features must be discrete, and not form a continuum like texture, or edge pixels (edgels). For this reason, our earlier work[1] has concentrated on the extraction and tracking of feature-points or corners, since they are discrete, reliable and meaningful[2]. However, the lack of connectivity of feature-points is a major limitation in our obtaining higher level descriptions, such as surfaces and objects. We need the richer information that is available from edges[3].

THE EDGE TRACKING PROBLEM

Matching between edge images on a pixel-by-pixel basis works for stereo, because of the known epi-polar camera geometry. However for the motion problem, where the camera motion is unknown, the aperture problem prevents us from undertaking explicit edgel matching. This could be overcome by solving for the motion beforehand, but we are still faced with the task of tracking each individual edge pixel and estimating its 3D location from, for example, Kalman Filtering. This approach is unattractive in comparison with assembling the edgels into edge segments, and tracking those segments as the features.

Now, the unconstrained imagery we shall be considering will contain both curved edges and texture of various scales. Representing edges as a set of straight line fragments[4], and using these as our discrete features will be inappropriate, since curved lines and texture edges can be expected to fragment differently on each image of the sequence, and so be untrackable. Because of ill-conditioning, the use of parametrised curves (eg. circular arcs) cannot be expected to provide the solution, especially with real imagery.

Figure 1. Pair of images from an outdoor sequence.

Harris Detector

C.Harris and M.Stephens. “A Combined Corner and Edge Detector.”1988.

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \qquad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x^2} = I_x \cdot I_x \qquad I_{y^2} = I_y \cdot I_y \qquad I_{xy} = I_x \cdot I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x^2} = G_{\sigma'} * I_{x^2} \qquad S_{y^2} = G_{\sigma'} * I_{y^2} \qquad S_{xy} = G_{\sigma'} * I_{xy}$$

Harris Detector

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

4. Define the matrix at each pixel

$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

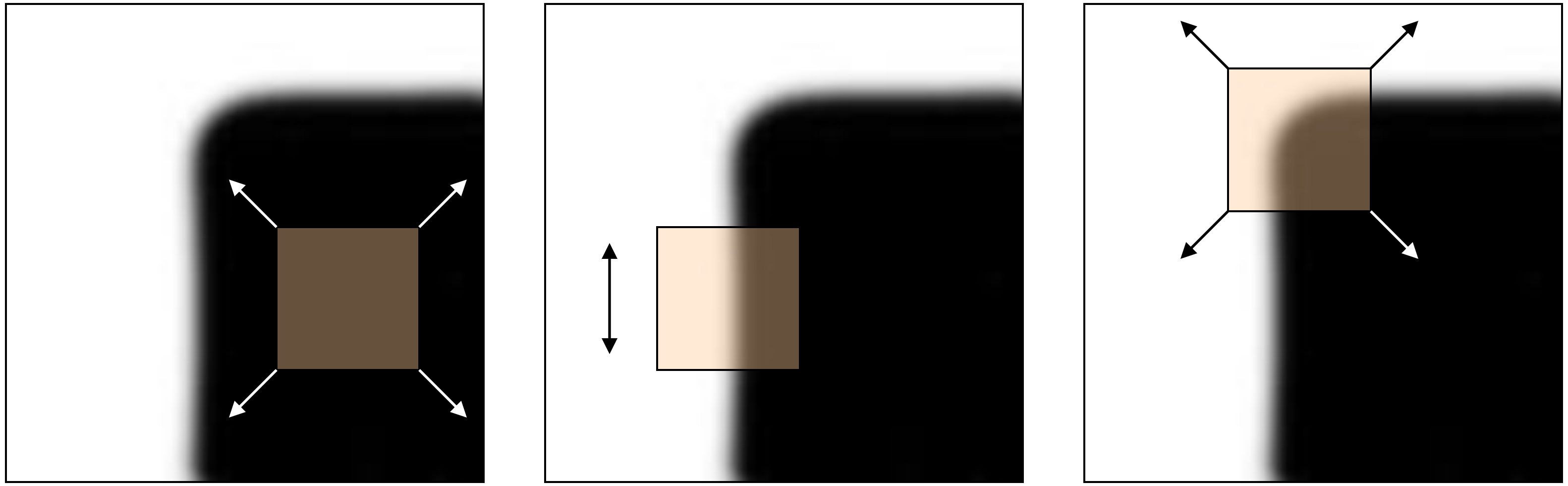
5. Compute the response of the detector at each pixel

$$R = \det M - k(\mathrm{trace} M)^2$$

6. Threshold on value of R; compute non-max suppression.

Corner Detection: 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

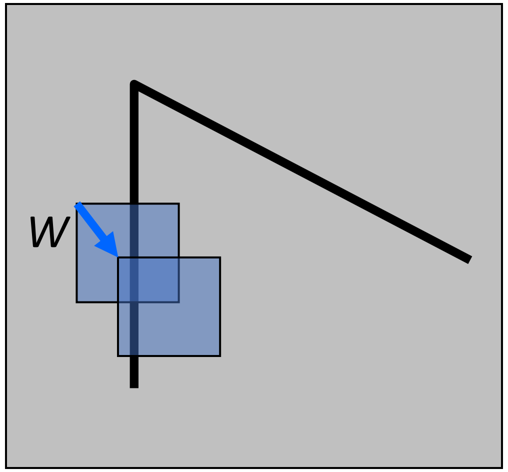
"edge":
no change along the edge direction

"corner":
significant change in all directions

Source: A. Efros

Corner detection the math

- Consider shifting the window W_{nn} by (*u*,*v*)
  - how do the pixels in W change?
- Write pixels in window as a vector:



$$\phi_0 = [I(0,0), I(0,1), \ldots, I(n,n)]$$

$$\phi_1 = [I(0+u, 0+v), I(0+u, 1+v), \ldots, I(n+u, n+v)]$$

$$E(u,v) = \|\phi_0 - \phi_1\|_2^2$$

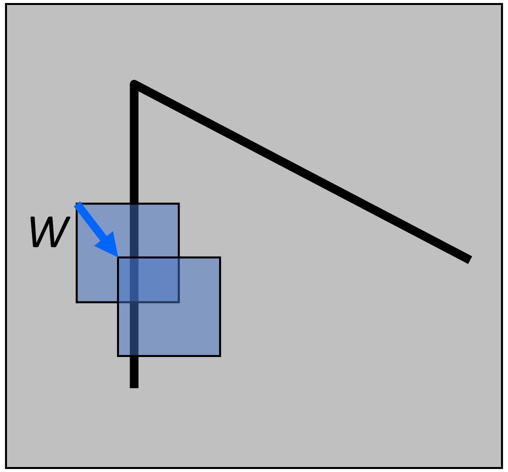
Corner detection: the math

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" *E*(*u*,*v*):

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

- We want E(u,v) to be *as high as possible for all u, v!*



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

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

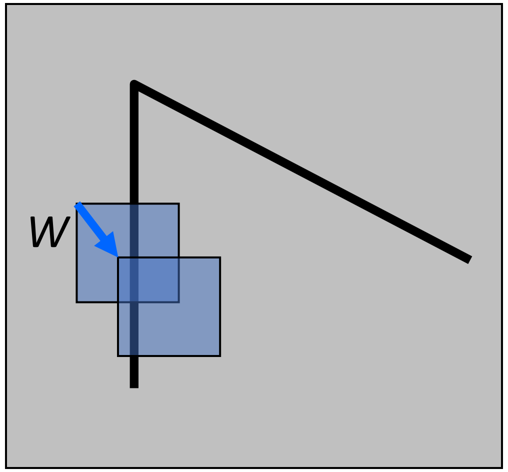
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- define an SSD "error" *E(u,v)*:

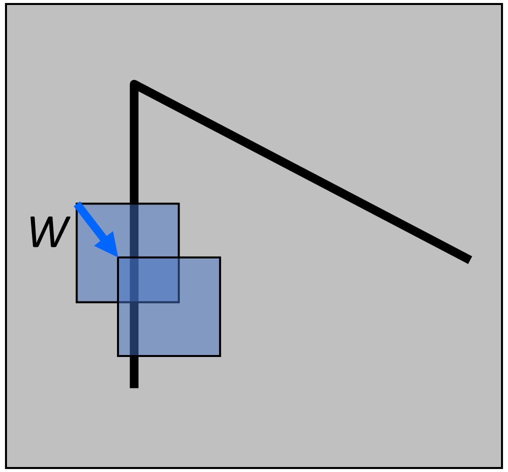


$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} [I(x+u,y+v) - I(x,y)]^2 \\
&\approx \sum_{(x,y)\in W} [I(x,y) + I_x u + I_y v - I(x,y)]^2 \\
&\approx \sum_{(x,y)\in W} [I_x u + I_y v]^2
\end{aligned}
$$

Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- define an "error" *E(u,v)*:



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

$$\approx Au^2 + 2Buv + Cv^2$$

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

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

A more general formulation

- Maybe all pixels in the patch are not equally important
- Consider a "window function" $w(x, y)$ that acts as weights
- $E(u, v) = \sum_{(x,y)\in W} w(x, y)[I(x + u, y + v) - I(x, y)]^2$
- Case till now:
  - w(x,y) = 1 inside the window, 0 otherwise

Using a window function

- Change in appearance of window w(x,y) for the shift [u,v]:

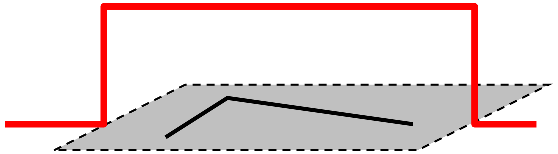
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

Shifted intensity

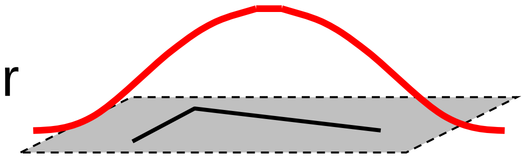
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Source: R. Szeliski

Redoing the derivation using a window function

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

$$\approx \sum_{x,y \in W} w(x,y)\big[I(x,y) + uI_x(x,y) + vI_y(x,y) - I(x,y)\big]^2$$

$$= \sum_{x,y \in W} w(x,y)\big[uI_x(x,y) + vI_y(x,y)\big]^2$$

$$= \sum_{x,y \in W} w(x,y)[u^2 I_x(x,y)^2 + v^2 I_y(x,y)^2 + 2uvI_x(x,y)I_y(x,y)]$$

Redoing the derivation using a window function

-

$$E(u,v) \approx \sum_{x,y \in W} w(x,y)[u^2 I_x(x,y)^2 + v^2 I_y(x,y)^2 + 2uvI_x(x,y)I_y(x,y)]$$

$$= Au^2 + 2Buv + Cv^2$$

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

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

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

The second moment matrix

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\underbrace{M = \sum_{x,y \in W} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}}_{\text{Second moment matrix}}$$

The second moment matrix

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\underbrace{M = \sum_{x,y \in W} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}}_{\text{Second moment matrix}}$$

Recall that we want E(u,v) to be as large as possible for all u,v

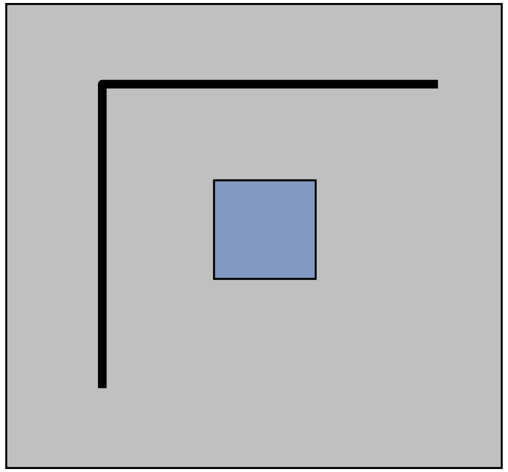
What does this mean in terms of M?

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

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

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



Flat patch: $I_x = 0$

$I_y = 0$

$$M = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

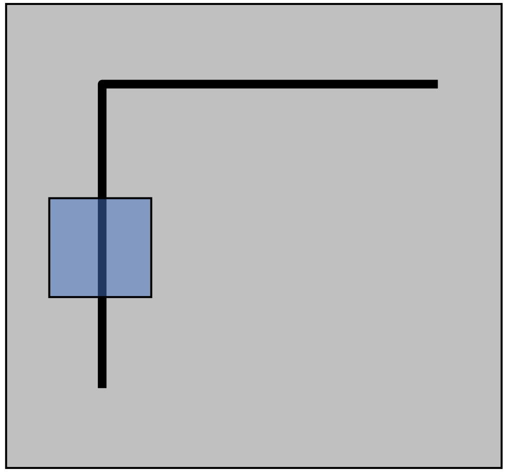
$$E(u,v) = 0 \quad \forall u, v$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

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

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



Vertical edge: $I_y = 0$

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

$$M \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

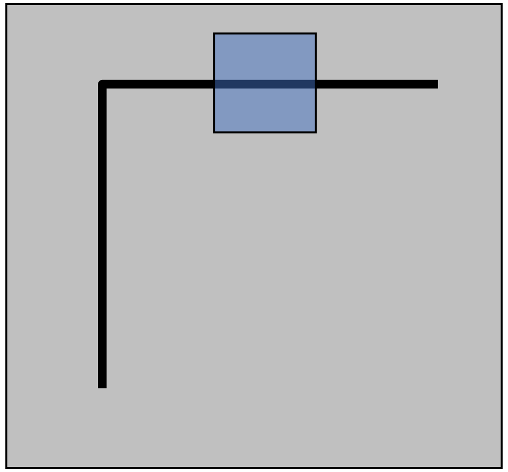
$$E(0,v) = 0 \ \ \forall v$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{M} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

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

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



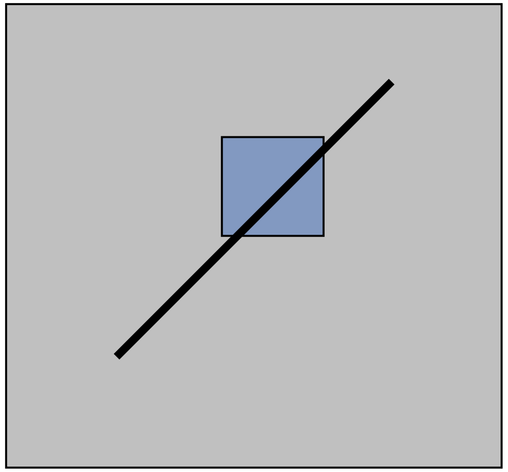
Horizontal edge: $I_x = 0$

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

$$M \begin{bmatrix} u \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$E(u,0) = 0 \; \forall u$$

What about edges in arbitrary orientation?



$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

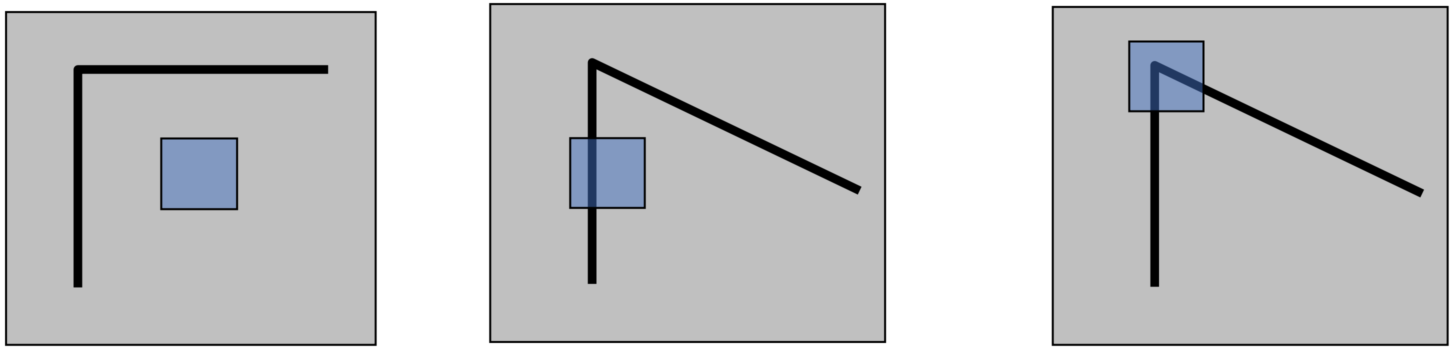
$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Leftrightarrow E(u,v) = 0$$

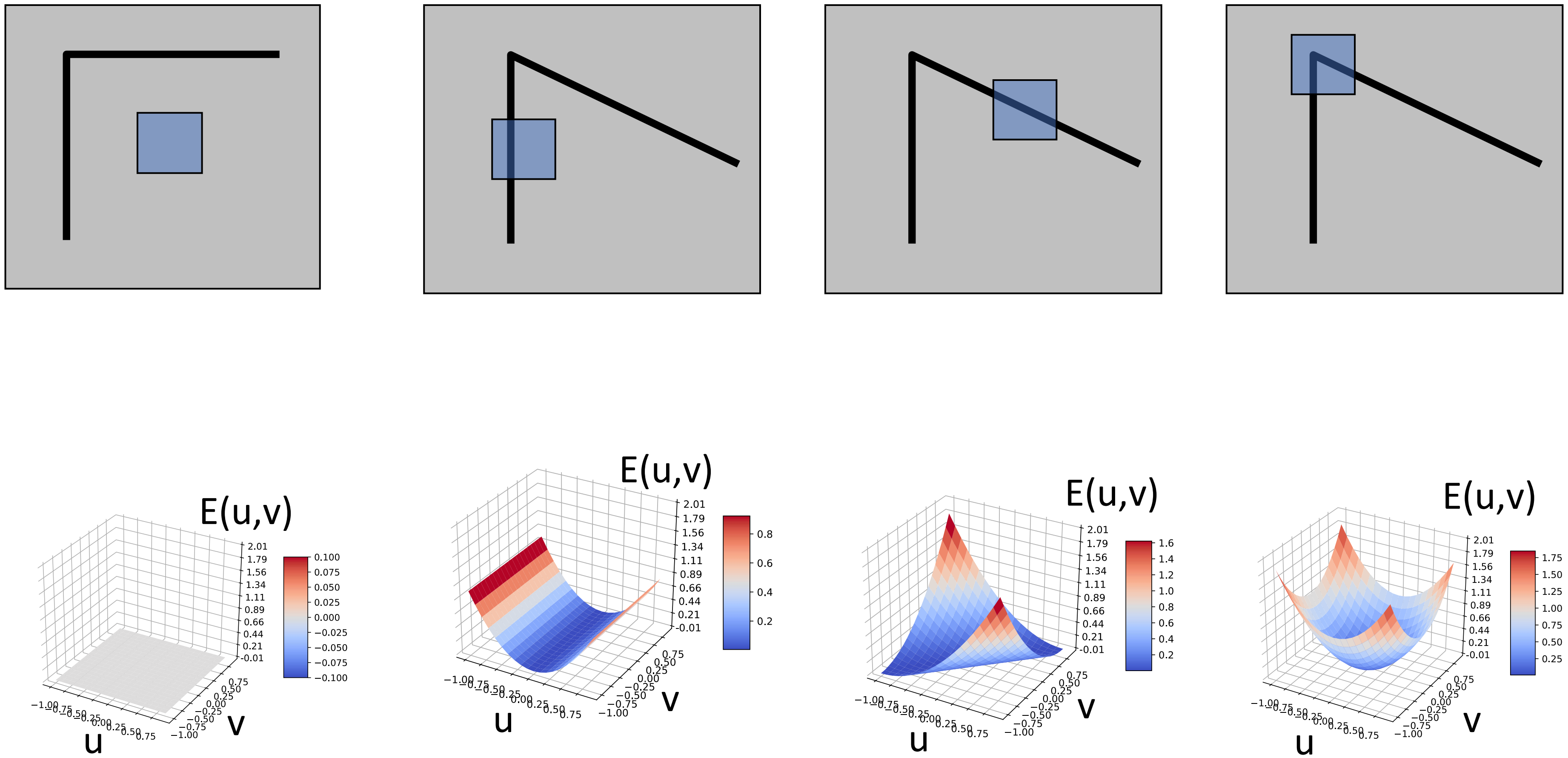
Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

For corners, we want no such directions to exist




E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v

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

Harris Detector

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \qquad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x^2} = I_x \cdot I_x \qquad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{xy} = G_{\sigma'} * I_{xy} \qquad S_{y^2} = G_{\sigma'} * I_{y^2}$$

$$S_{x^2} = G_{\sigma'} * I_{x^2}$$

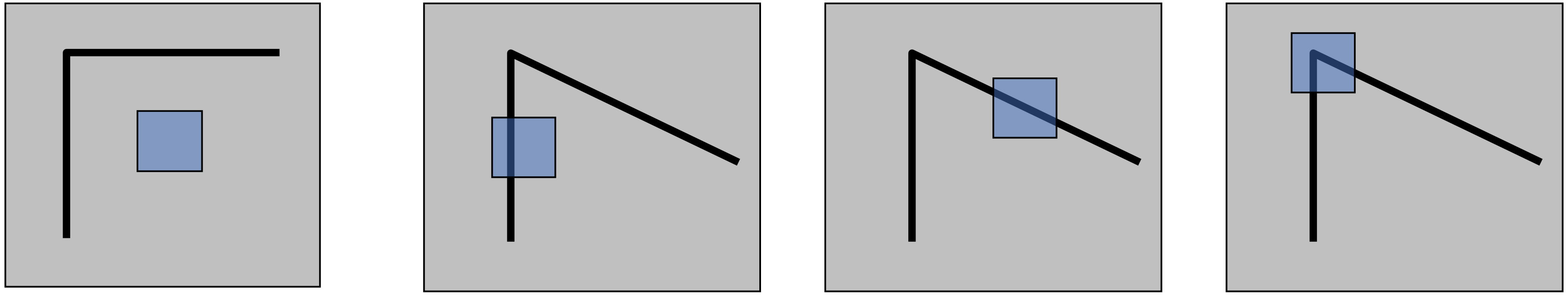
4. Define the matrix at each pixel

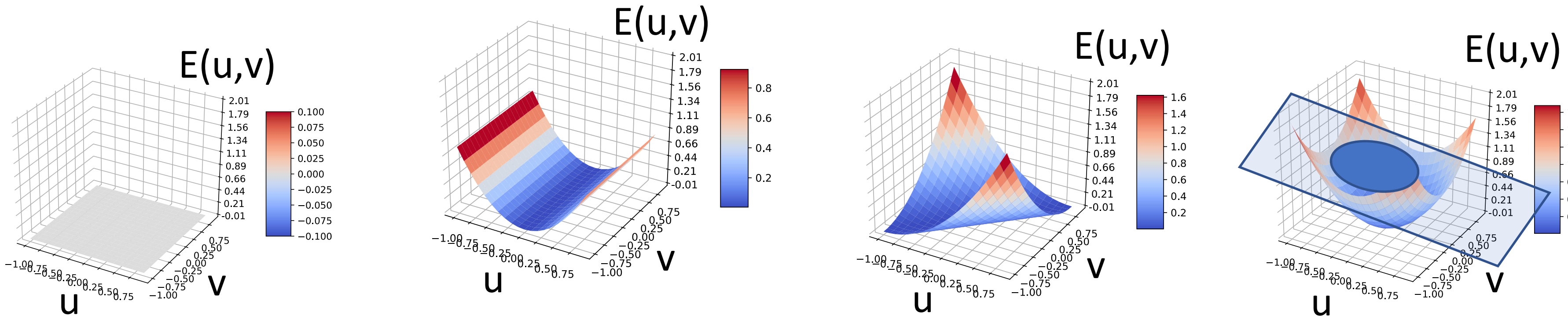
$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

$$R = \det M - k(\mathrm{trace} M)^2$$

6. Threshold on value of R; compute non-max suppression.





$$E(u, v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{Constant}$$

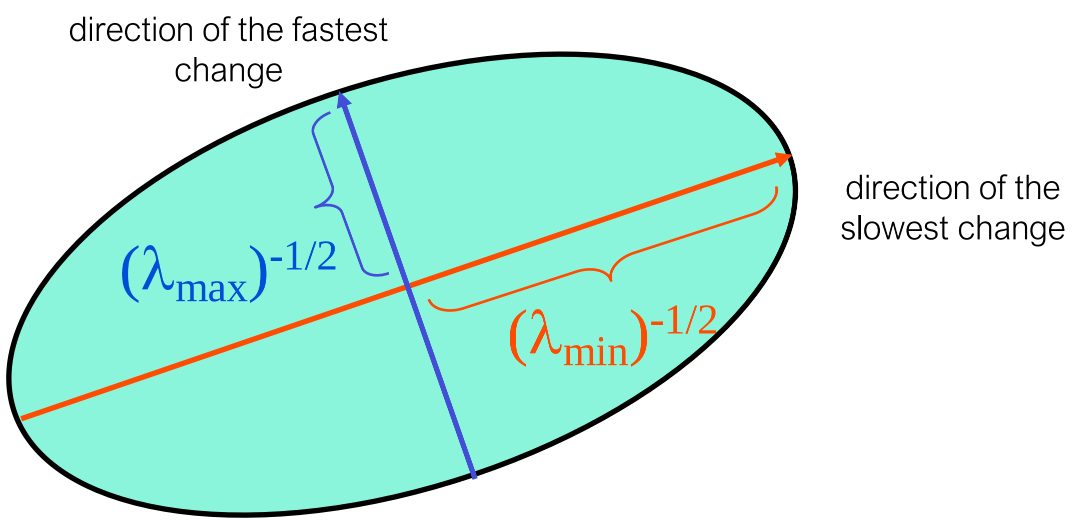
Visualization as an ellipse

Since M is symmetric, we have $M = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$

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

Ellipse equation:

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



SVD

$$A = U \Sigma V^{-1} \qquad \Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \cdot & \\ & & & \sigma_N \end{bmatrix}$$

U, V = orthogonal matrix $\longrightarrow$ $U^{-1} = U^T$

$$\sigma_i = \sqrt{\lambda_i}$$

σ = singular value
λ = eigenvalue of $A^t A$

<u>For a square symmetric matrix</u>

- U,V becomes Rotation Matrix R
- Diagonal matrix has eigenvalues of A

Compute eigenvalues and eigenvectors

eigenvalue

$$\boldsymbol{M e} = \lambda \boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

Compute eigenvalues and eigenvectors

eigenvalue

$$M\boldsymbol{e} = \lambda \boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

1. Compute the determinant of $M - \lambda I$
(returns a polynomial)

Compute eigenvalues and eigenvectors

eigenvalue

$$Me = \lambda e$$

eigenvector

$$(M - \lambda I)e = 0$$

1. Compute the determinant of $M - \lambda I$
(returns a polynomial)

2. Find the roots of polynomial $\det(M - \lambda I) = 0$
(returns eigenvalues)

Compute eigenvalues and eigenvectors

eigenvalue

$$Me = \lambda e$$

eigenvector

$$(M - \lambda I)e = 0$$

1. Compute the determinant of $M - \lambda I$
(returns a polynomial)

2. Find the roots of polynomial $\det(M - \lambda I) = 0$
(returns eigenvalues)

3. For each eigenvalue, solve $(M - \lambda I)e = 0$
(returns eigenvectors)

Eigenvalues & Eigenvector computation example

- Compute eigenvalues, eigenvectors of $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$

- determinant of the matrix (A − λI) equals zero are the eigenvalues

$$|A - \lambda I| = \left| \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = \begin{vmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{vmatrix}$$
$$= 3 - 4\lambda + \lambda^2.$$

- Setting the characteristic polynomial equal to zero, it has roots at <u>λ=1</u> and <u>λ=3</u>, which are the two eigenvalues of A.

Eigenvalues & Eigenvector computation example

- Compute eigenvalues, eigenvectors of $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$

- For λ=1,

$$(A - I)\mathbf{v}_{\lambda=1} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$1v_1 + 1v_2 = 0$$

  - Any nonzero vector with v1 = −v2 solves this equation.

$$\mathbf{v}_{\lambda=1} = \begin{bmatrix} v_1 \\ -v_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

For λ=3,

$$(A - 3I)\mathbf{v}_{\lambda=3} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

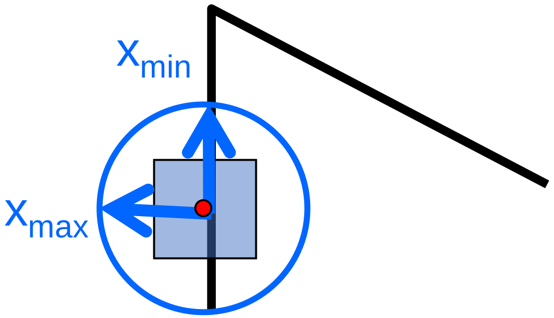
$$-1v_1 + 1v_2 = 0;$$
$$1v_1 - 1v_2 = 0$$

Any nonzero vector with v1 = v2 solves this equation. Therefore,

$$\mathbf{v}_{\lambda=3} = \begin{bmatrix} v_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Eigenvalues and eigenvectors of M

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$



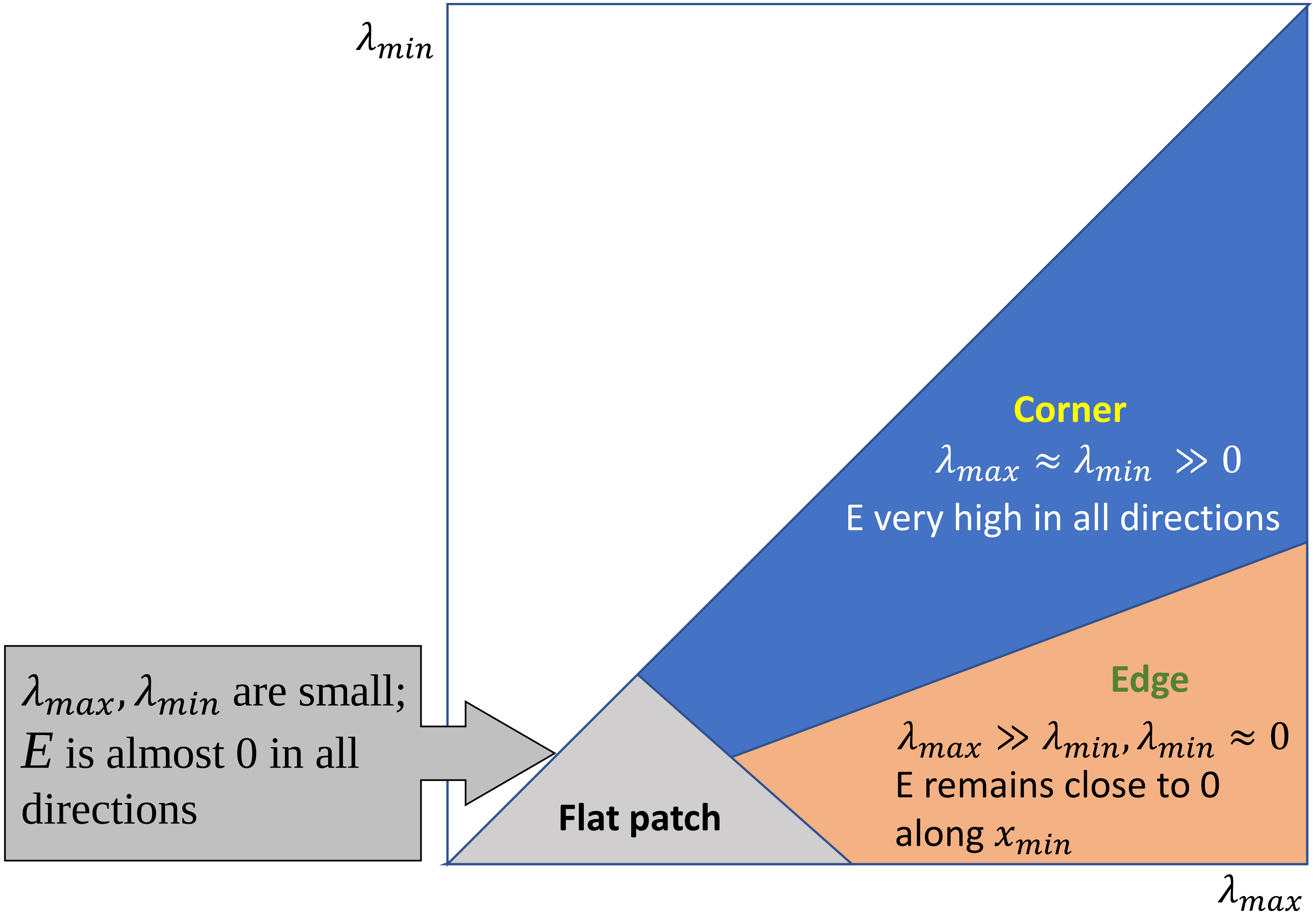
$$M\, x_{\max} = \lambda_{\max} x_{\max}$$

$$M\, x_{\min} = \lambda_{\min} x_{\min}$$

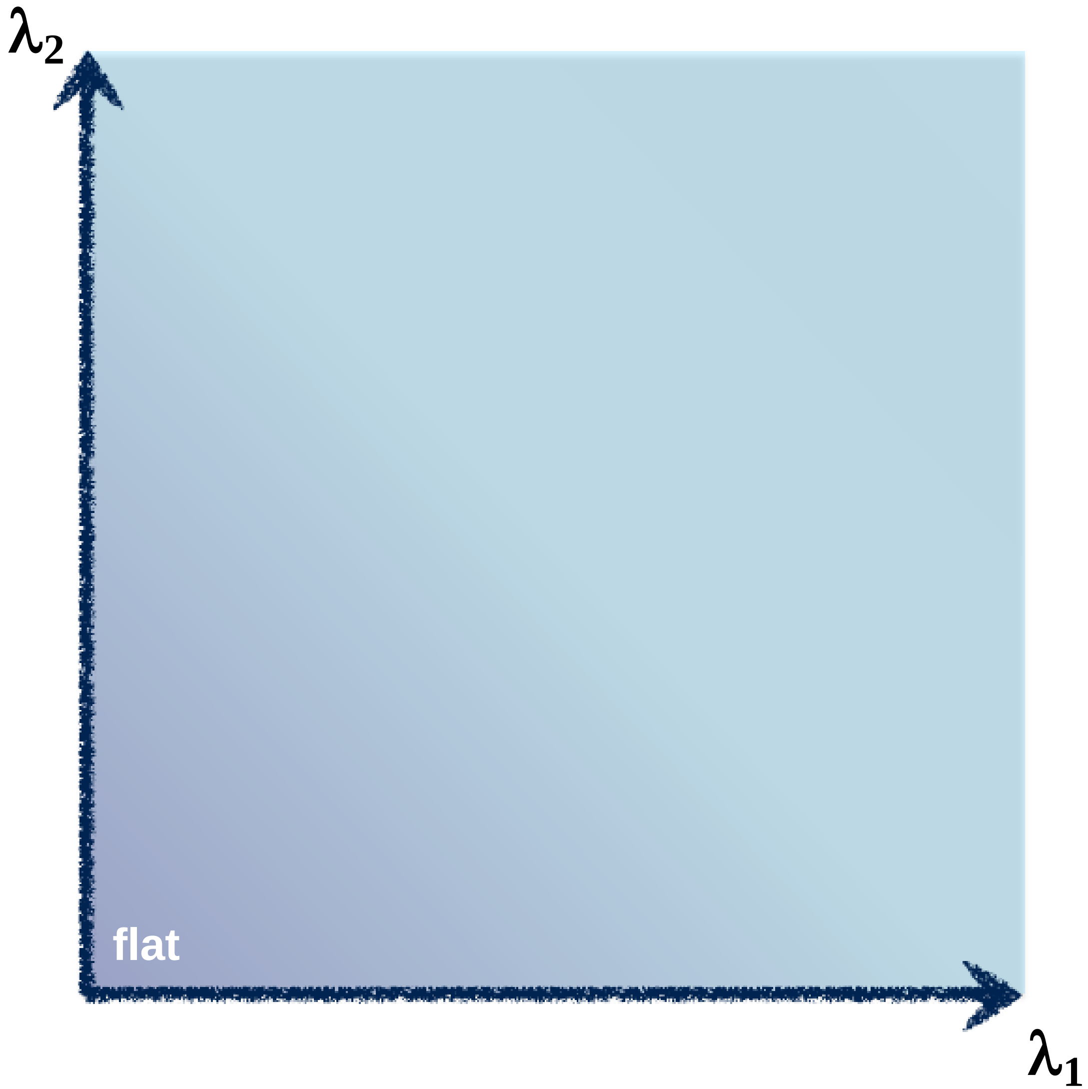
Eigenvalues and eigenvectors of M

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

Interpreting the eigenvalues

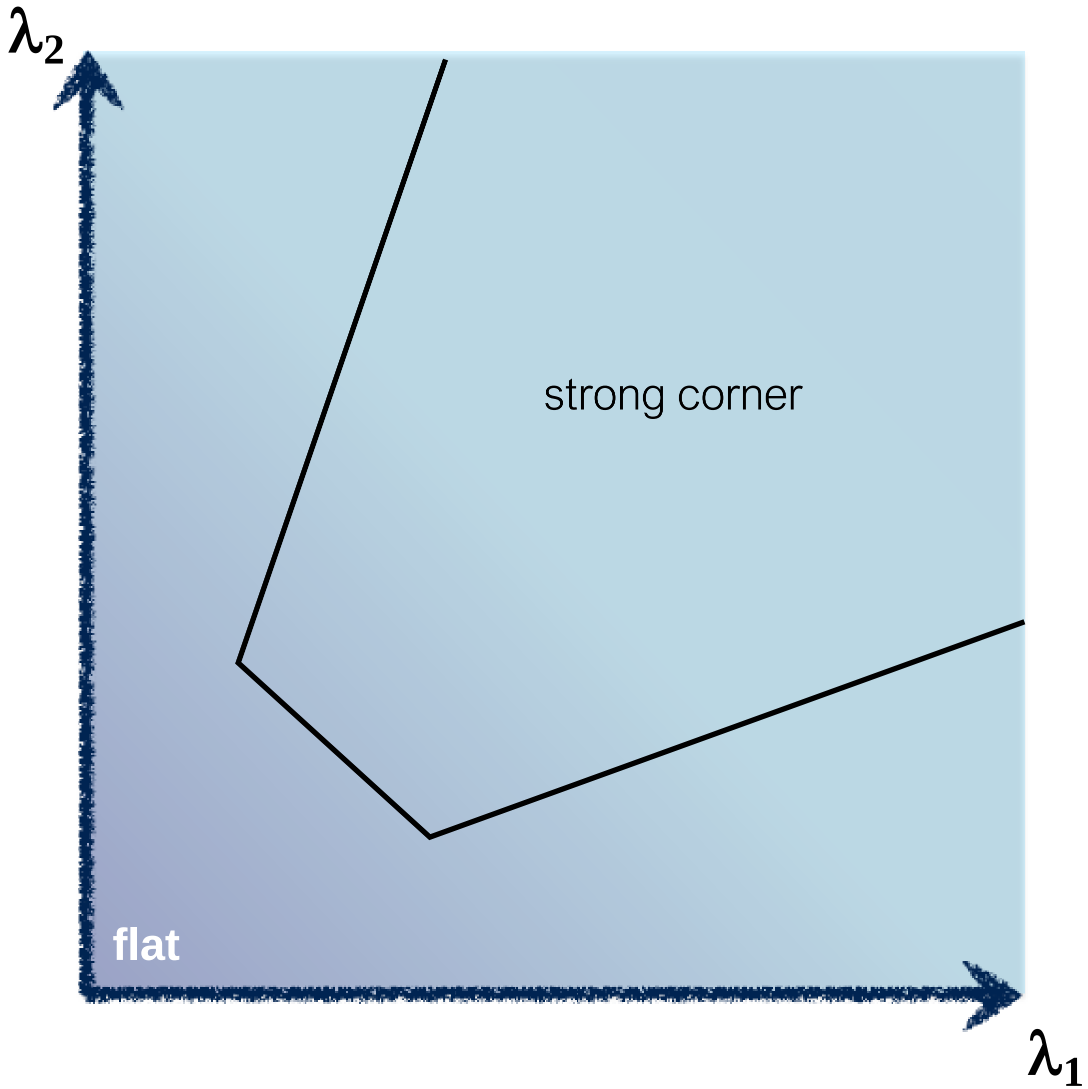


Use threshold on eigenvalues to detect corners



Think of a function to score 'cornerness'

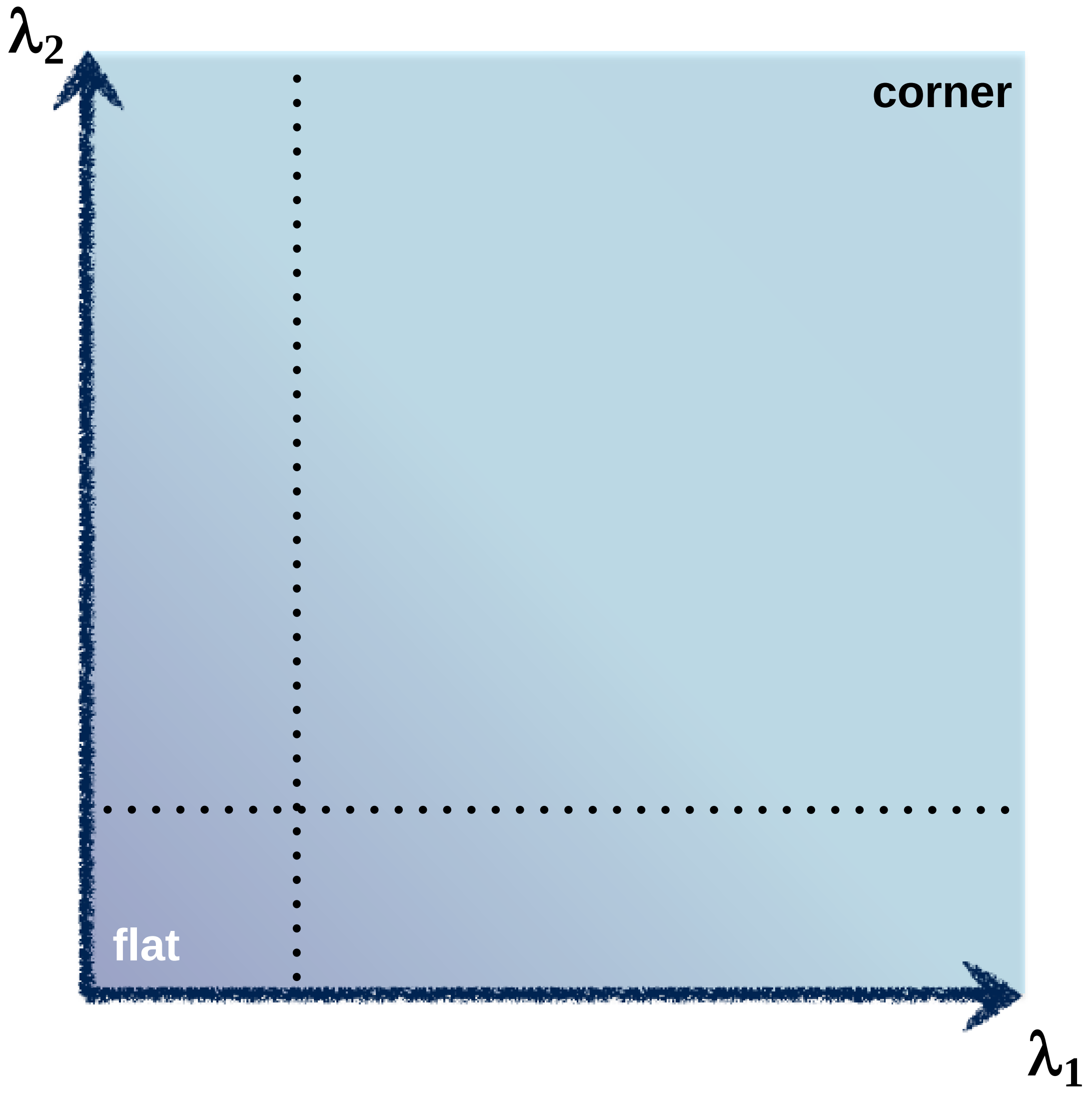
Use threshold on eigenvalues to detect corners



Think of a function to score 'cornerness'

Use threshold on eigenvalues to detect corners

(a function of)

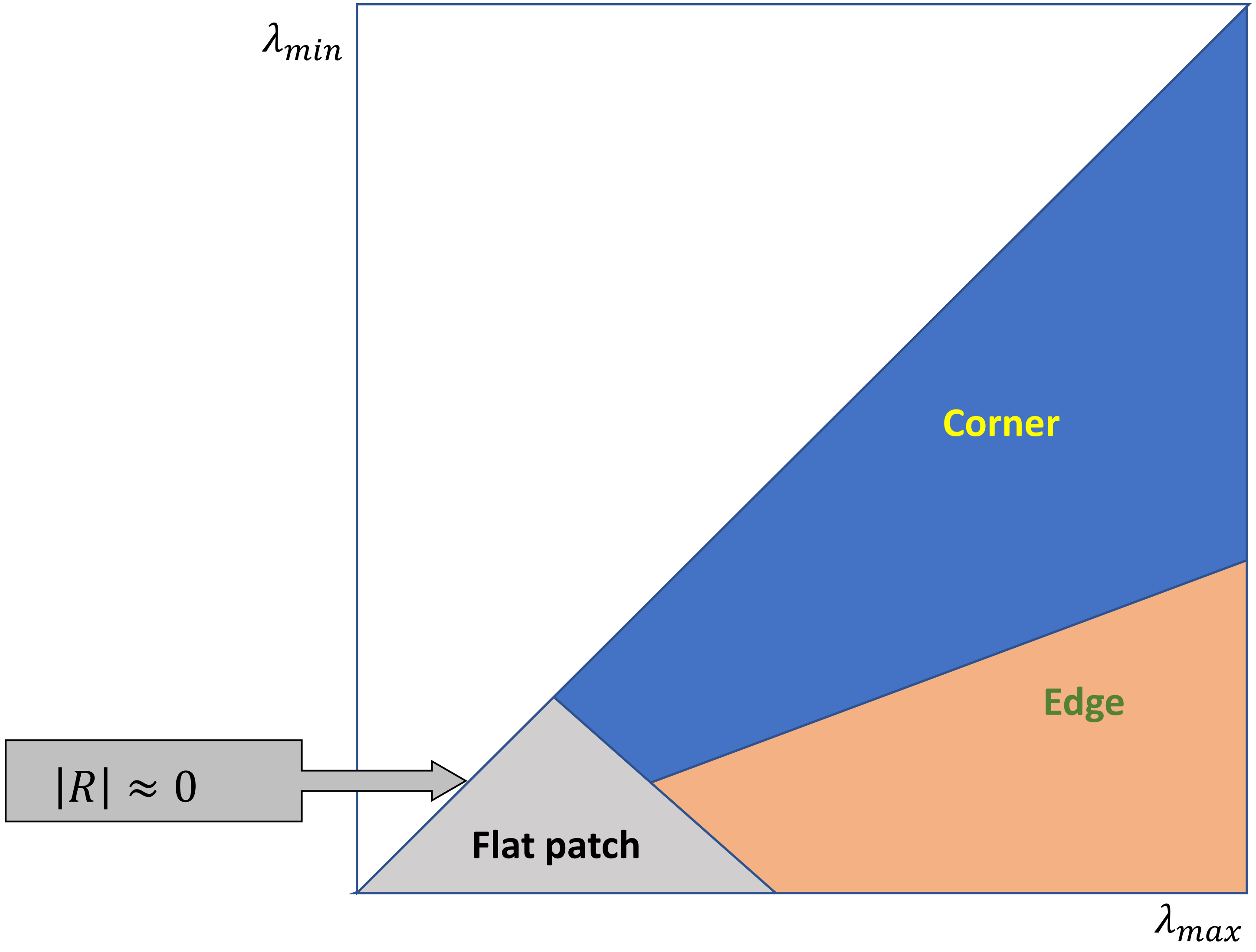


Use the smallest eigenvalue as the response function

$$R = \min(\lambda_1, \lambda_2)$$

Corner response function

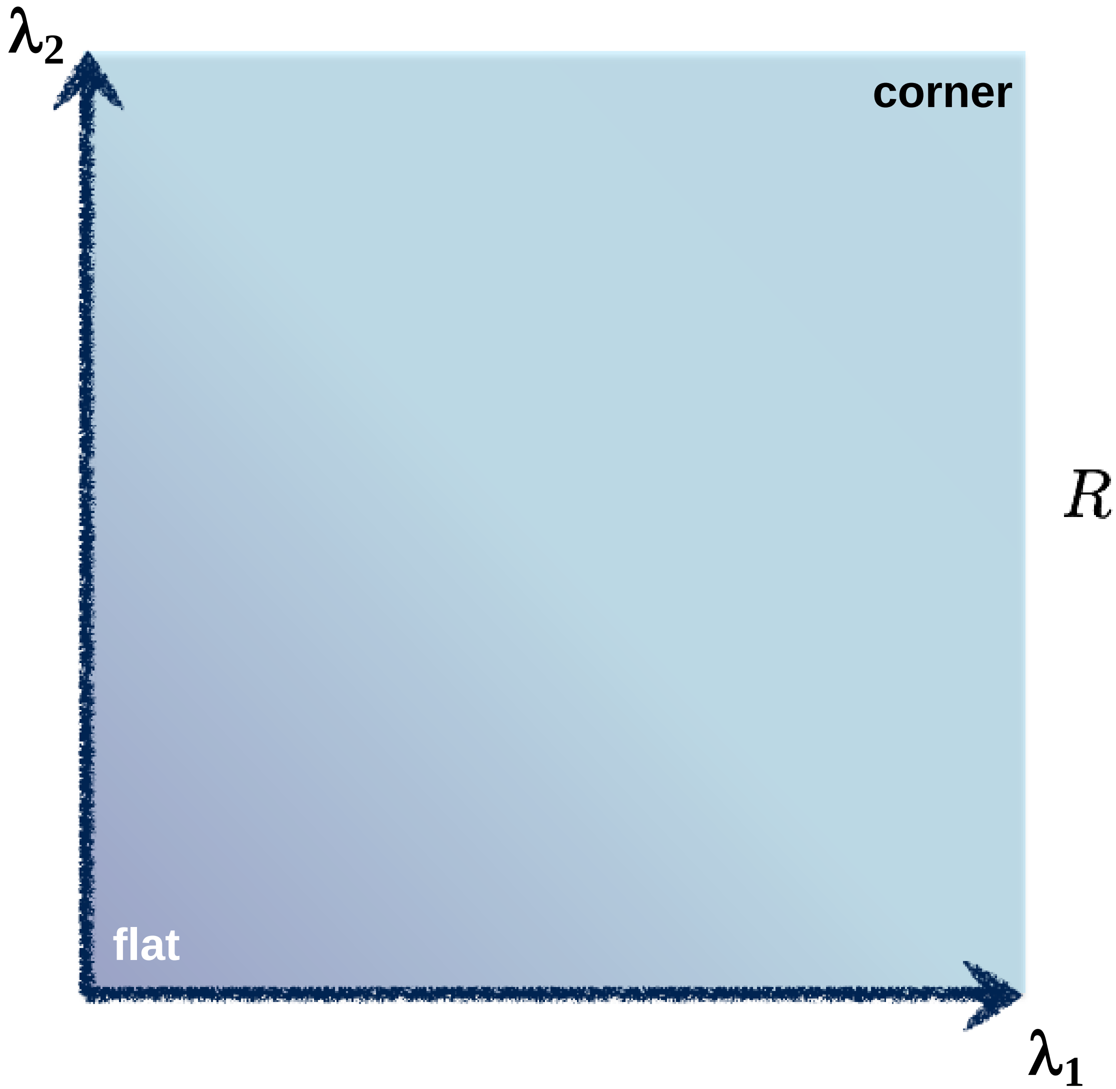
$$R = \min(\lambda_1, \lambda_2)$$



Use threshold on eigenvalues to detect corners

^

(a function of)



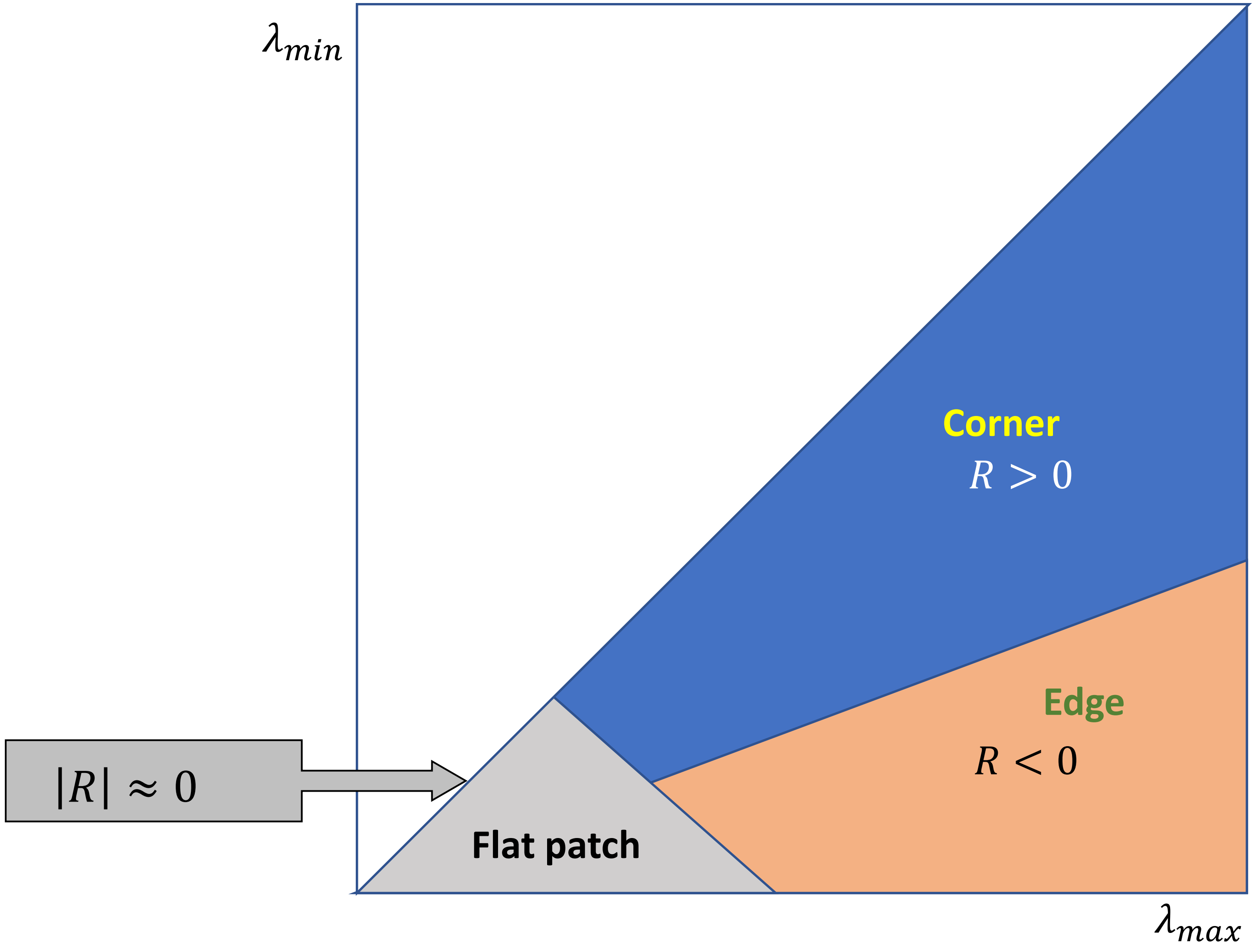
Eigenvalues need to be bigger than one.

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

Can compute this more efficiently…

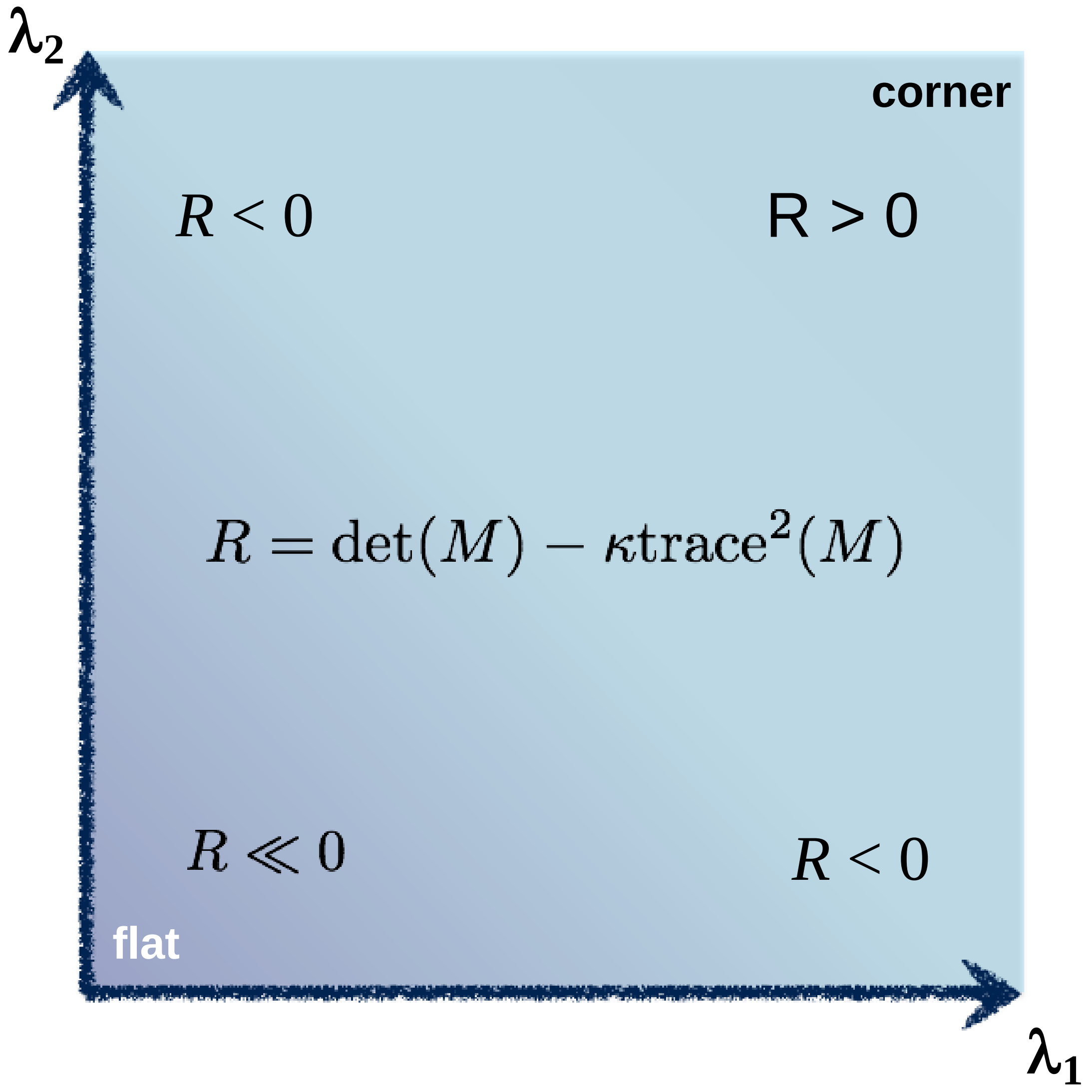
Corner response function

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



Use threshold on eigenvalues to detect corners

^
(a function of)



$$\det M = \lambda_1 \lambda_2$$

$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

$$trace\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = a + d$$

Harris & Stephens (1988)

$$R = \det(M) - \kappa \operatorname{trace}^2(M)$$

Kanade & Tomasi (1994)

$$R = \min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$R = \frac{\det(M)}{\operatorname{trace}(M) + \epsilon}$$

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

Harris Detector

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \qquad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x^2} = I_x \cdot I_x \qquad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{xy} = G_{\sigma'} * I_{xy} \qquad S_{y^2} = G_{\sigma'} * I_{y^2} \qquad S_{x^2} = G_{\sigma'} * I_{x^2}$$

4. Define the matrix at each pixel

$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

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

6. Threshold on value of R; compute non-max suppression.

Final step: Non-maxima suppression

- Pick a pixel as corner if cornerness at patch centered on it > cornerness of neighboring pixels
- And if cornerness exceeds a threshold

Harris Detector

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x^2} = I_x \cdot I_x \quad I_{y^2} = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{xy} = G_{\sigma'} * I_{xy} \qquad S_{y^2} = G_{\sigma'} * I_{y^2} \qquad S_{x^2} = G_{\sigma'} * I_{x^2}$$

4. Define the matrix at each pixel

$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

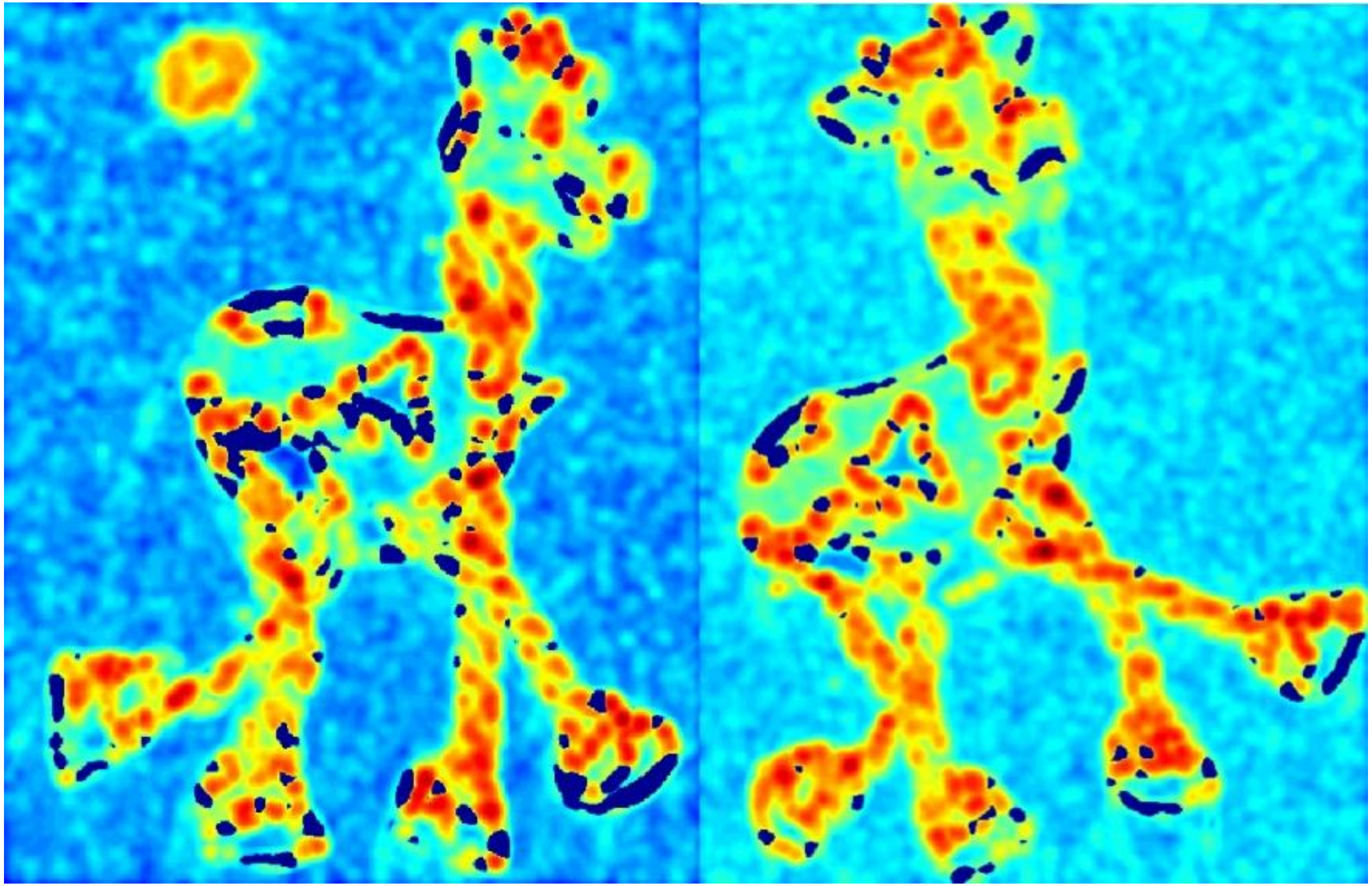
5. Compute the response of the detector at each pixel

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

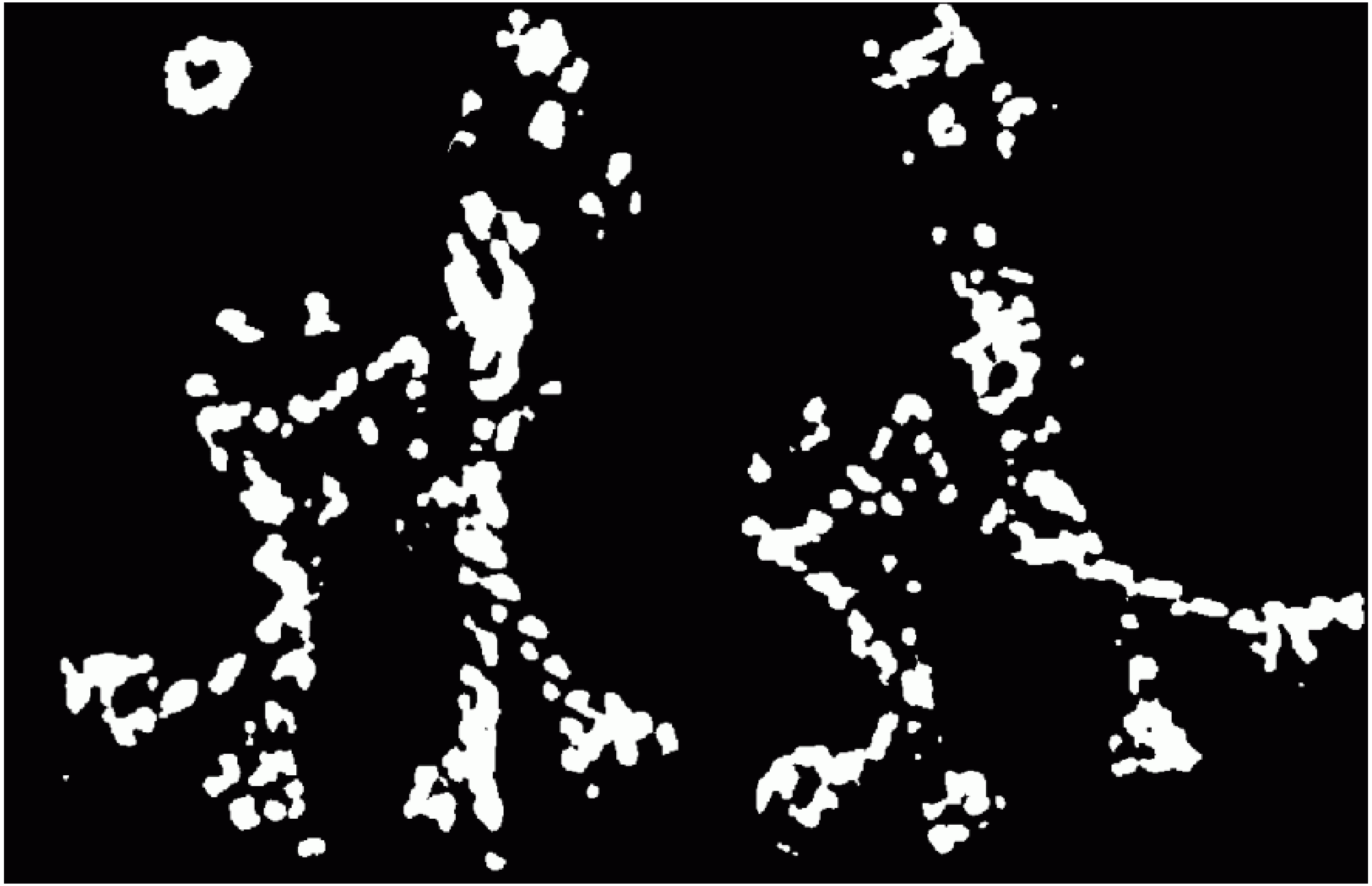
6. Threshold on value of R; compute non-max suppression.

Harris detector example

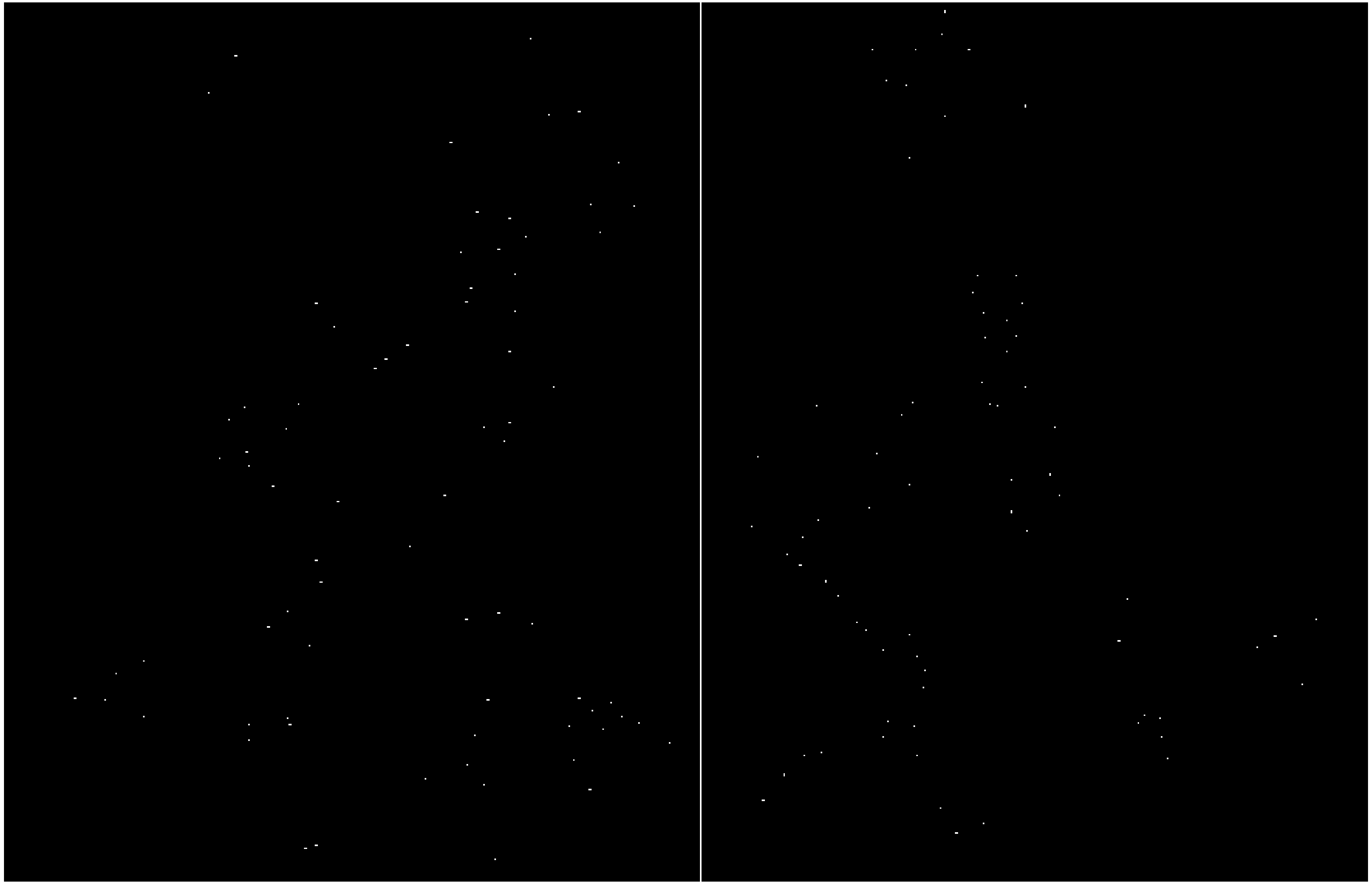
f value (red high, blue low)



Threshold (f > value)

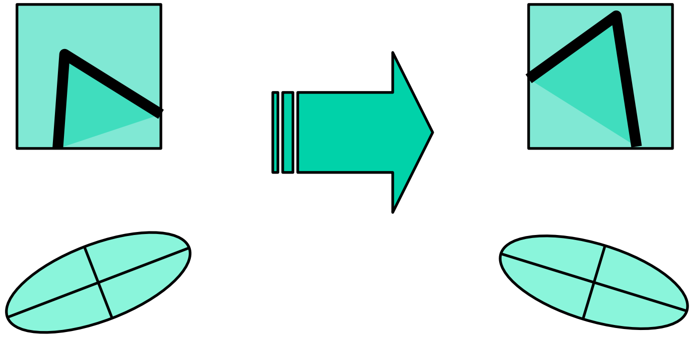


Find local maxima of f



Harris features (in red)

Harris corner response is invariant to rotation



Ellipse rotates but its shape (**eigenvalues**) remains the same

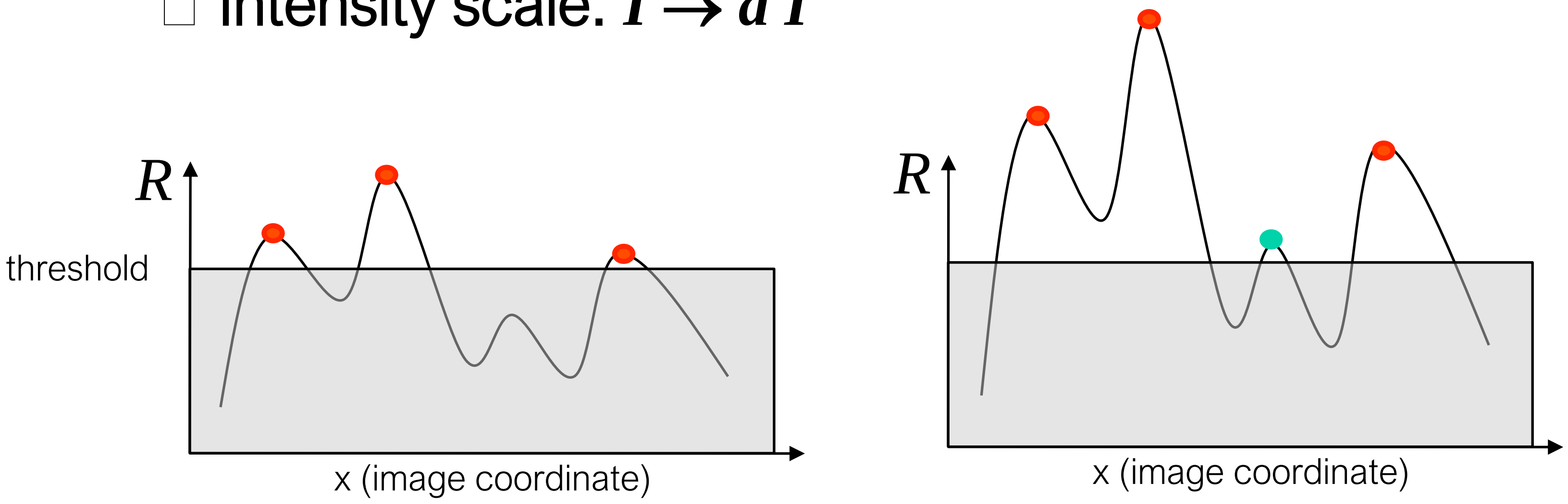
Corner response R is invariant to image rotation

Harris corner response is invariant to intensity changes

Partial invariance to *affine intensity* change

- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$

- Intensity scale: $I \rightarrow a\, I$



The Harris detector is not invariant to changes in …

The Harris corner detector is not invariant to scale

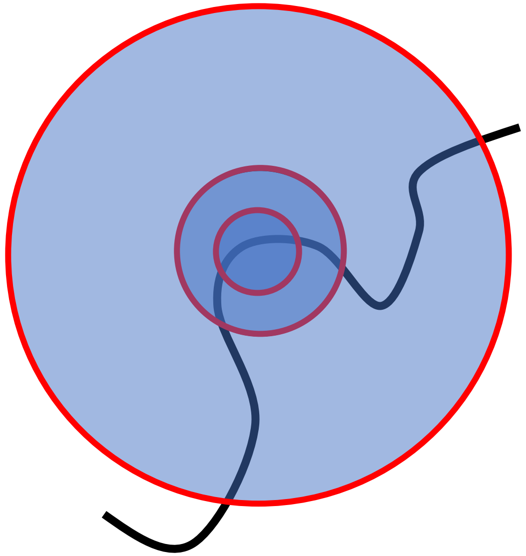
Multi-scale detection

How can we make a feature detector scale-invariant?

How can we automatically select the scale?

Scale invariant detection

Suppose you're looking for corners

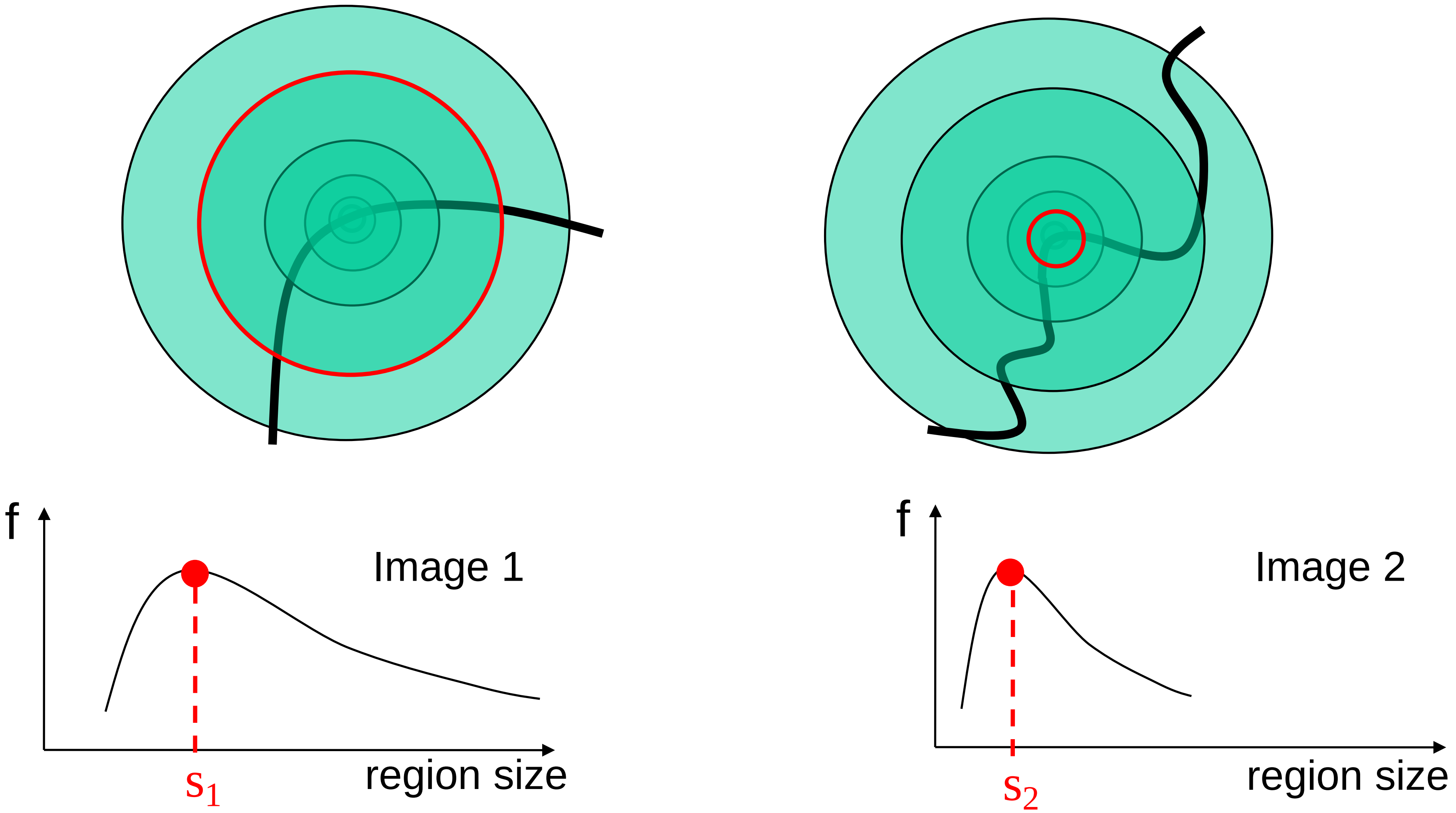


Key idea: find scale that gives local maximum of *cornerness*

- in both position and scale
- One definition of *cornerness*: the Harris operator

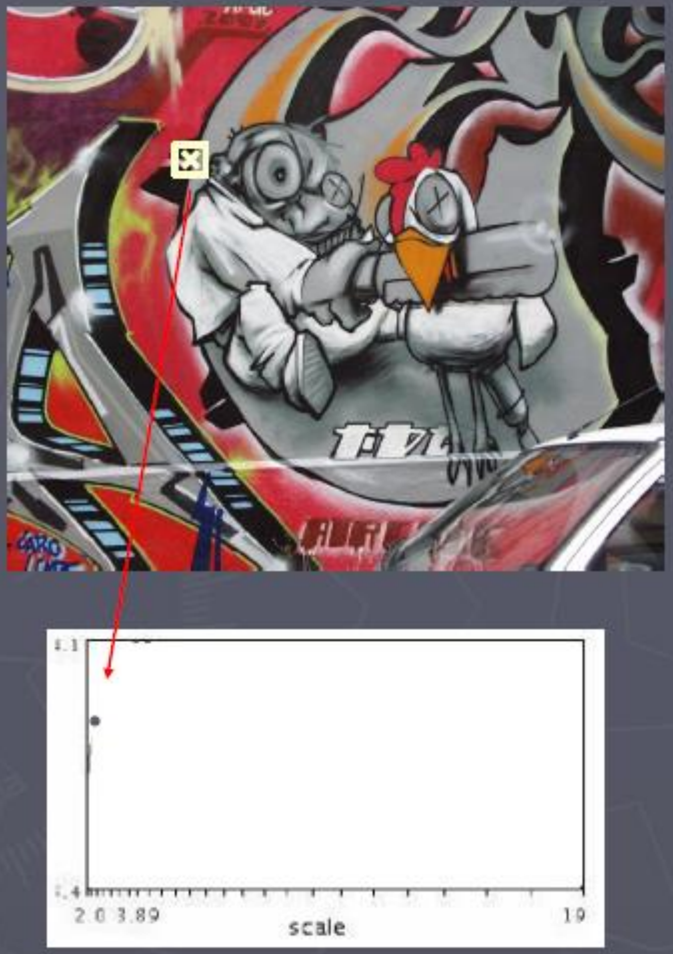
Intuitively…

Find local maxima in both **position** and **scale**



Automatic scale selection

Lindeberg et al., 1996



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

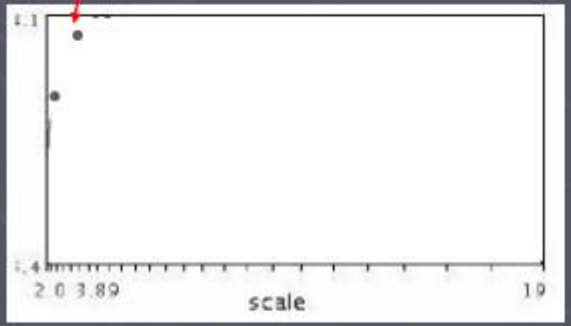
$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$

Slide from Tinne Tuytelaars

Automatic scale selection

$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$

Increased

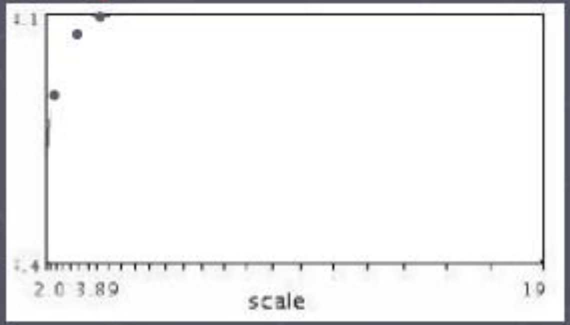


$f(I_{i_1 \ldots i_m}(x, \sigma))$

Automatic scale selection

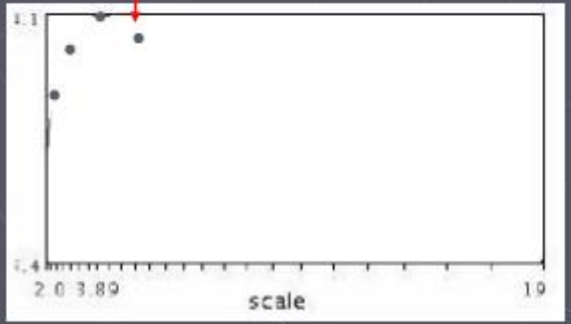
$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$

Increased



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

Automatic scale selection

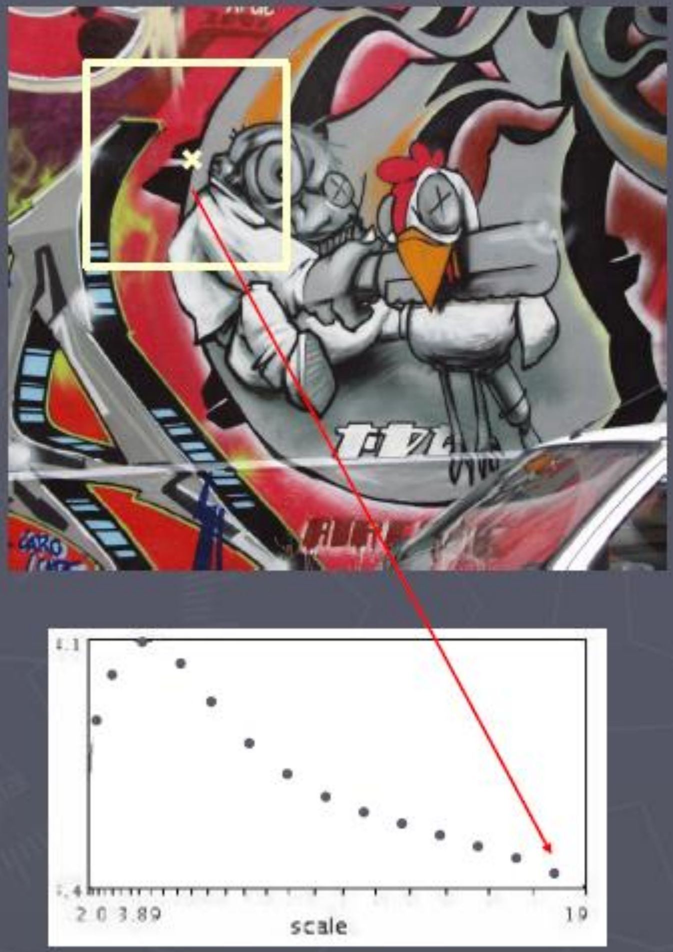


$f(I_{i_1 \dots i_m}(x, \sigma))$

$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$

Increased

Automatic scale selection



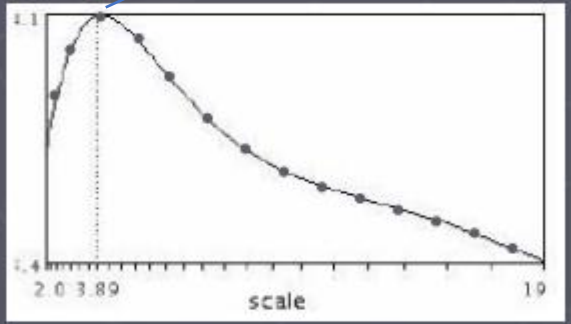
$f(I_{i_1 \ldots i_m}(x,\sigma))$

$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$

Increased

Automatic scale selection

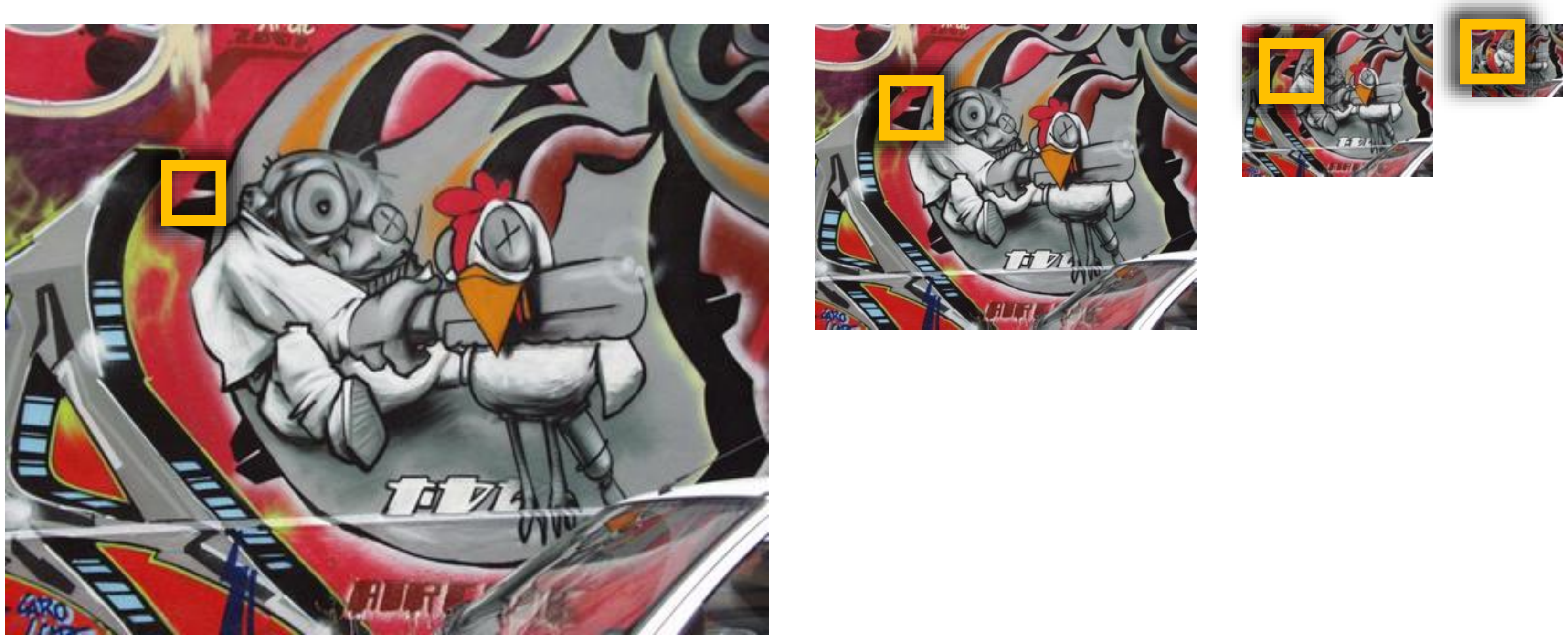
$$M = \mu(\mathbf{x}, \sigma_I, \sigma_D) = \sigma_D^2 g(\sigma_I) \otimes \begin{bmatrix} L_x^2(\mathbf{x}, \sigma_D) & L_x L_y(\mathbf{x}, \sigma_D) \\ L_x L_y(\mathbf{x}, \sigma_D) & L_y^2(\mathbf{x}, \sigma_D) \end{bmatrix}$$



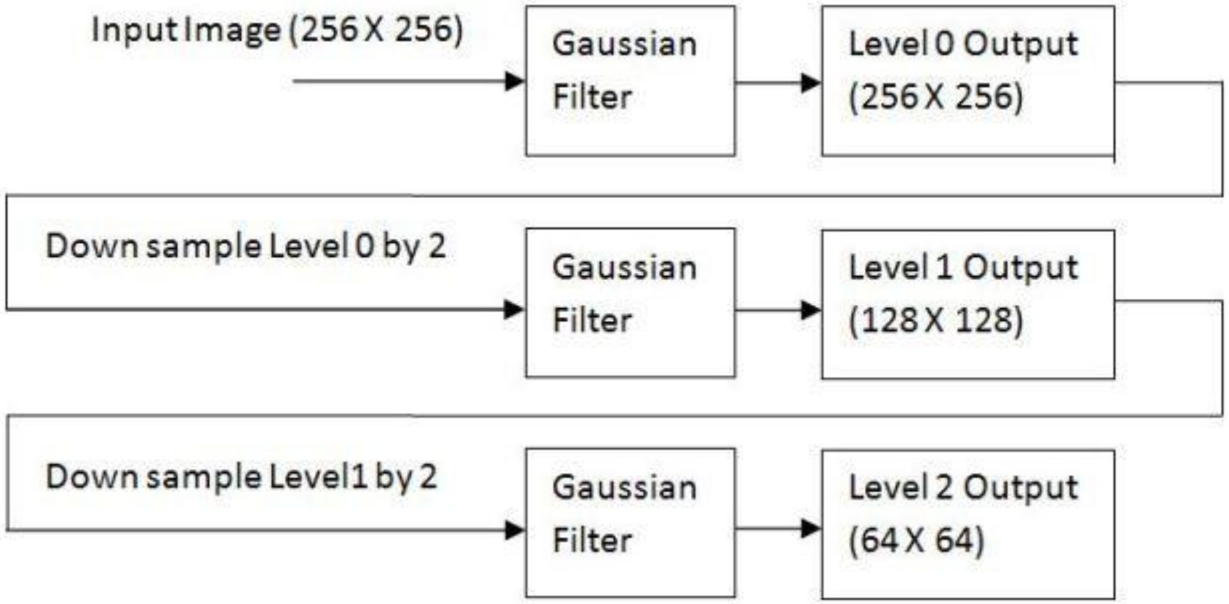
$$f(I_{i_1 \dots i_m}(x, \sigma))$$

Implementation

- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid



Gaussian pyramid implementation



How would you implement scale selection?

implementation

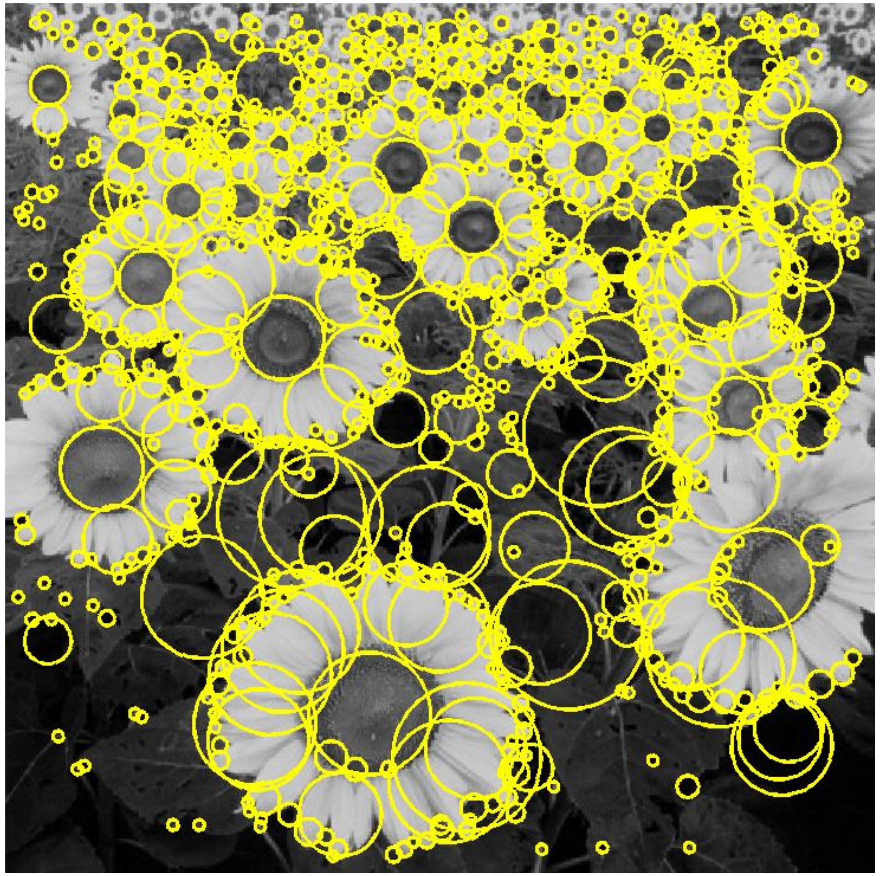
```
For each level of the Gaussian pyramid

        compute feature response (e.g. Harris, Laplacian)


For each level of the Gaussian pyramid

        if local maximum and cross-scale

        save scale and location of feature
```

$$(x, y, s)$$

Blob detection

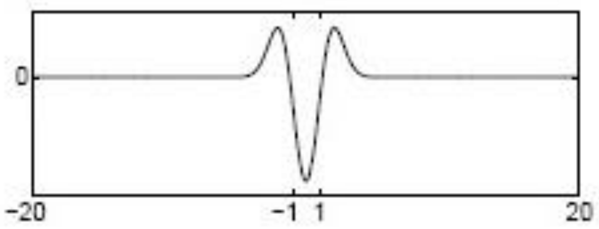
Scale-space blob detector: Example

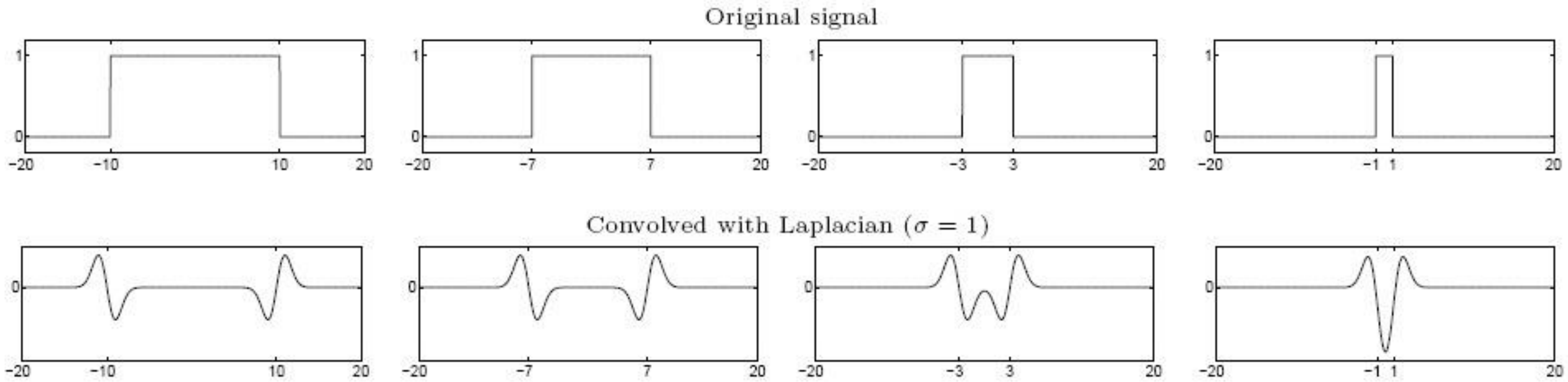
Feature extraction: Corners and blobs



Formally…

Laplacian filter

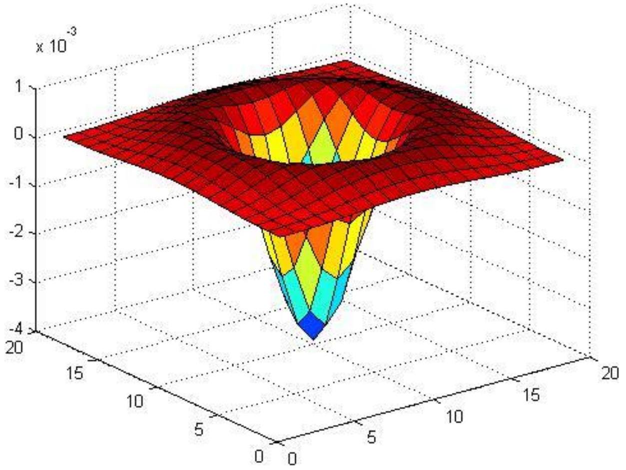


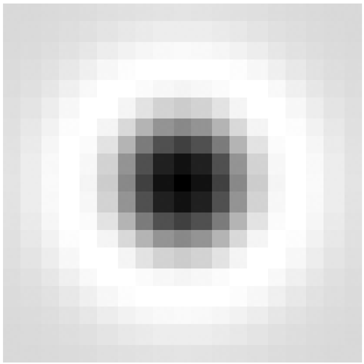


Highest response when the signal has the same **characteristic scale** as the filter

Another common definition of *f*

- The *Laplacian of Gaussian* (*LoG*)



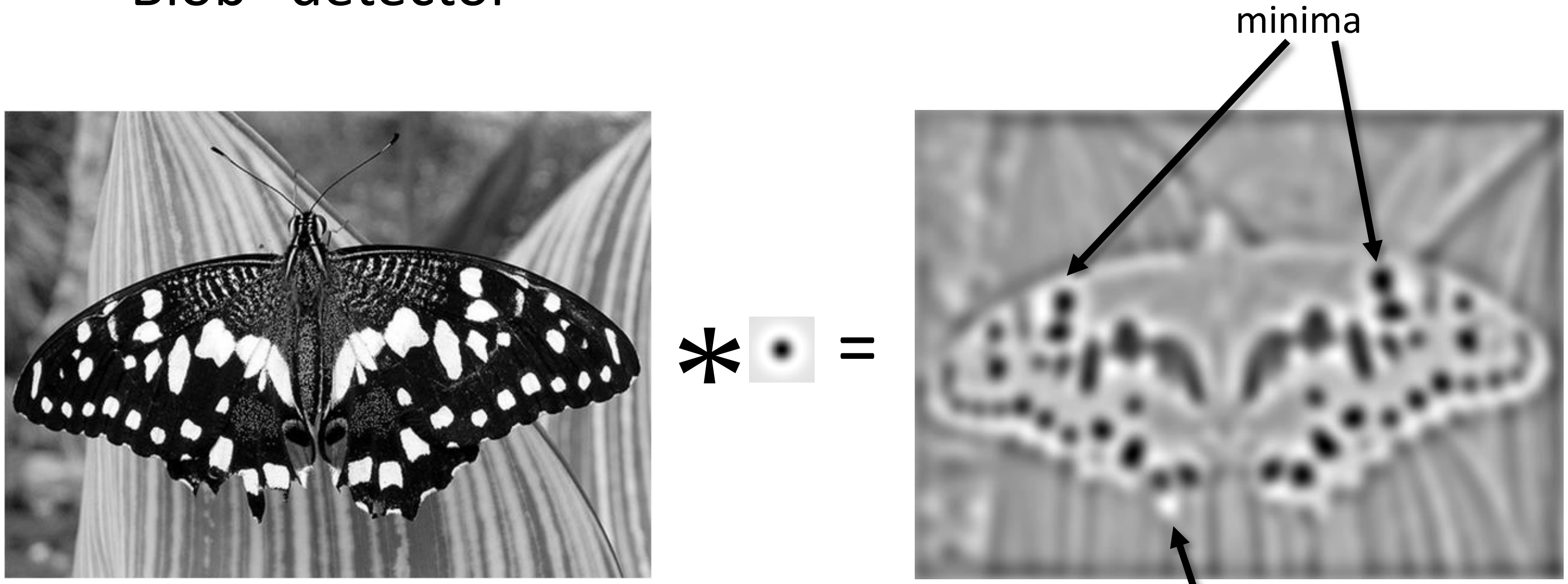


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

(very similar to a Difference of Gaussians (DoG) – i.e. a Gaussian minus a slightly smaller Gaussian)

Laplacian of Gaussian

- “Blob” detector

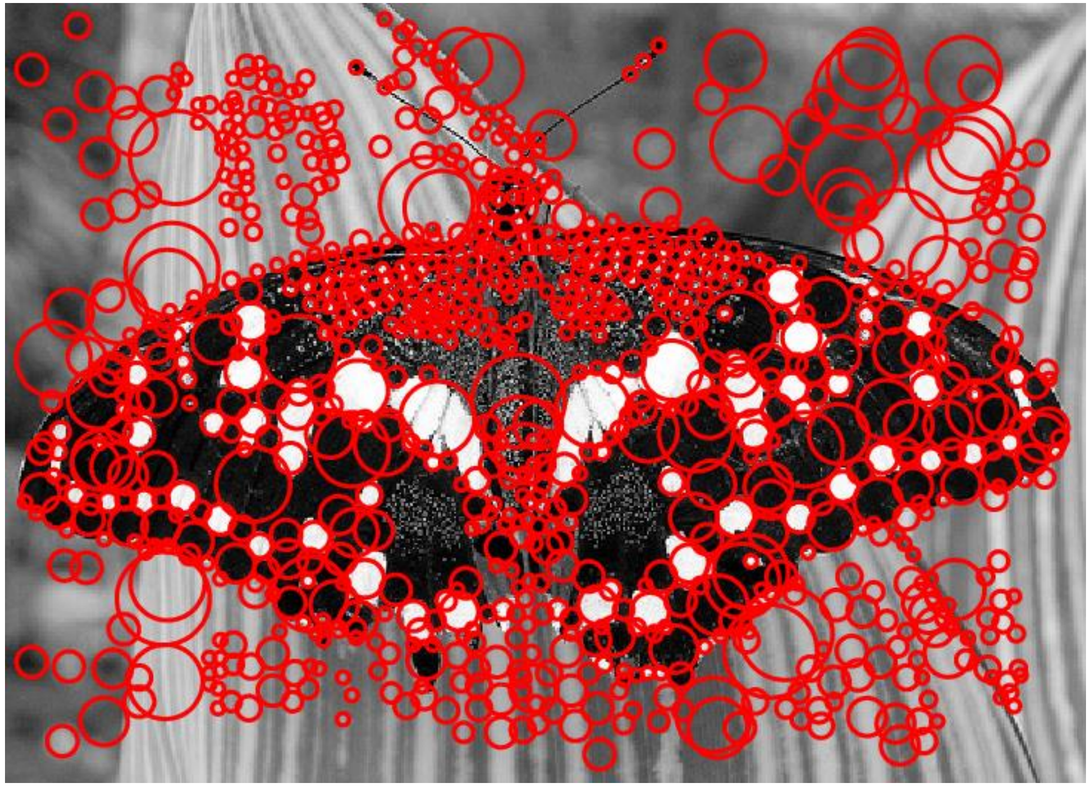


- Find maxima *and minima* of LoG operator in space and scale

Scale-space blob detector: Example

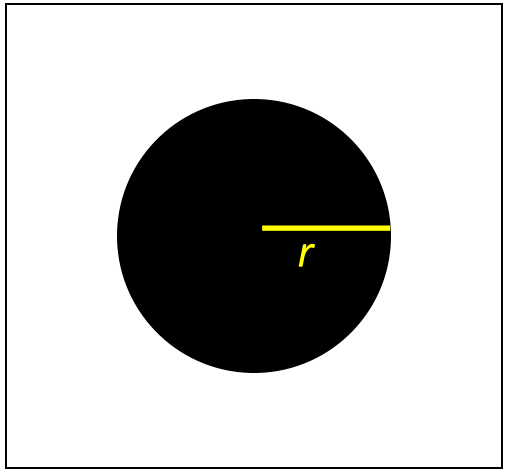
sigma = 11.9912

Scale-space blob detector: Example

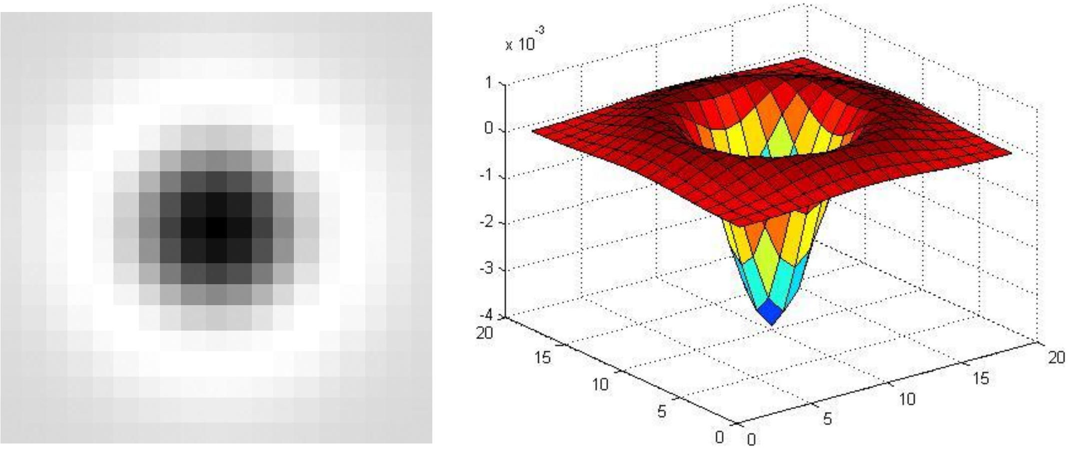


Scale selection

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



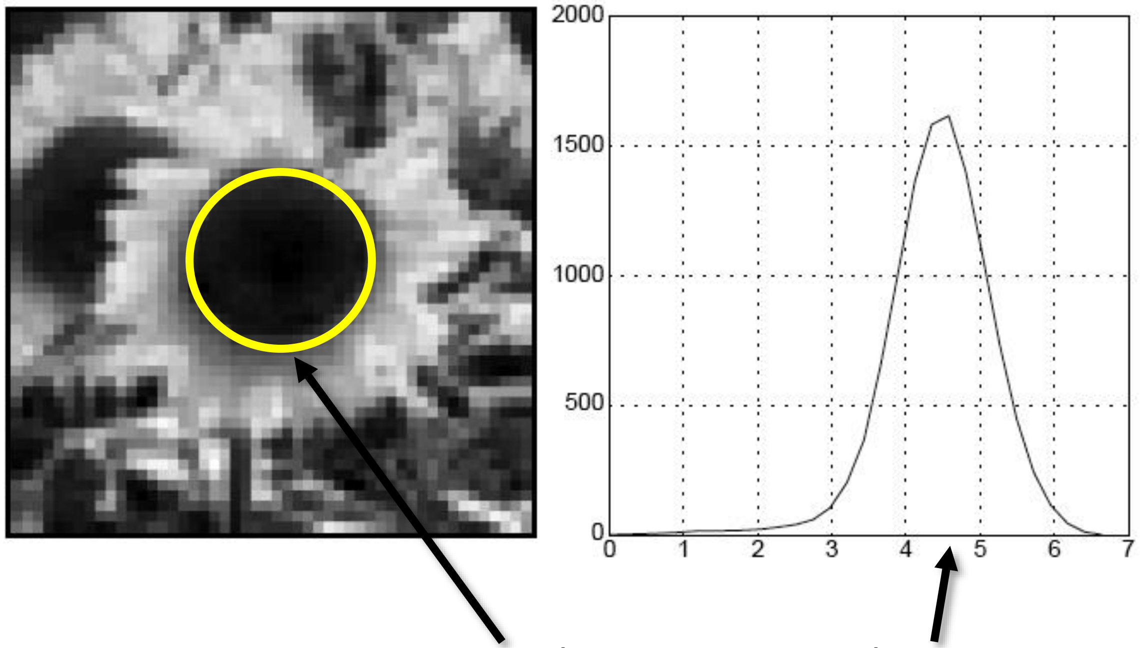
image



Laplacian

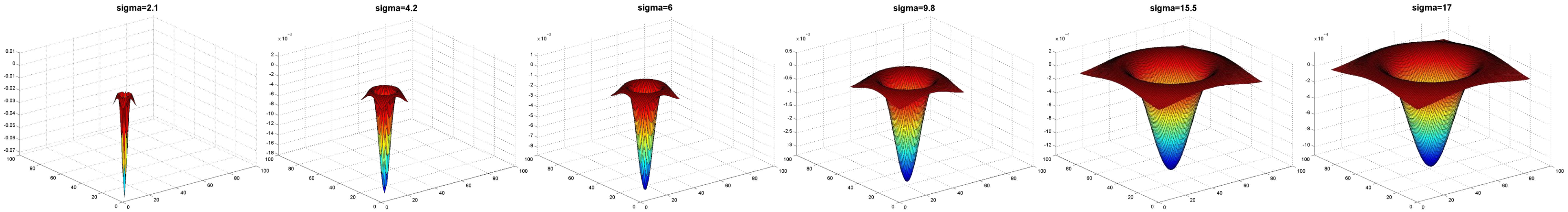
Characteristic scale

- We define the characteristic scale as the scale that produces peak of Laplacian response



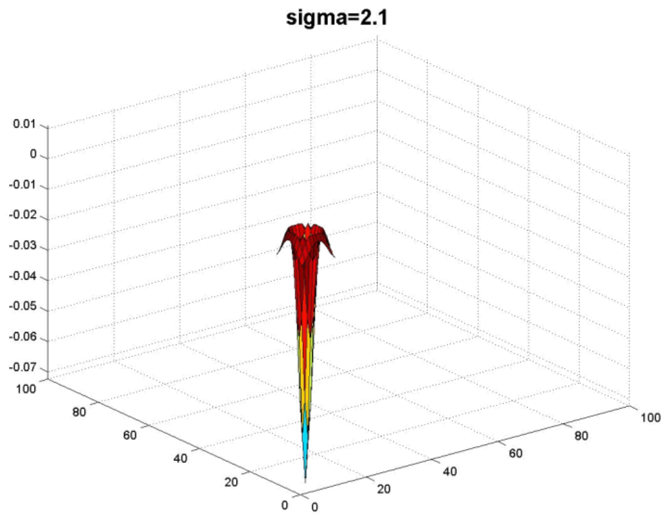
T. Lindeberg (1998). ["Feature detection with automatic scale selection."]() *International Journal of Computer Vision* **30** (2): pp 77--116.

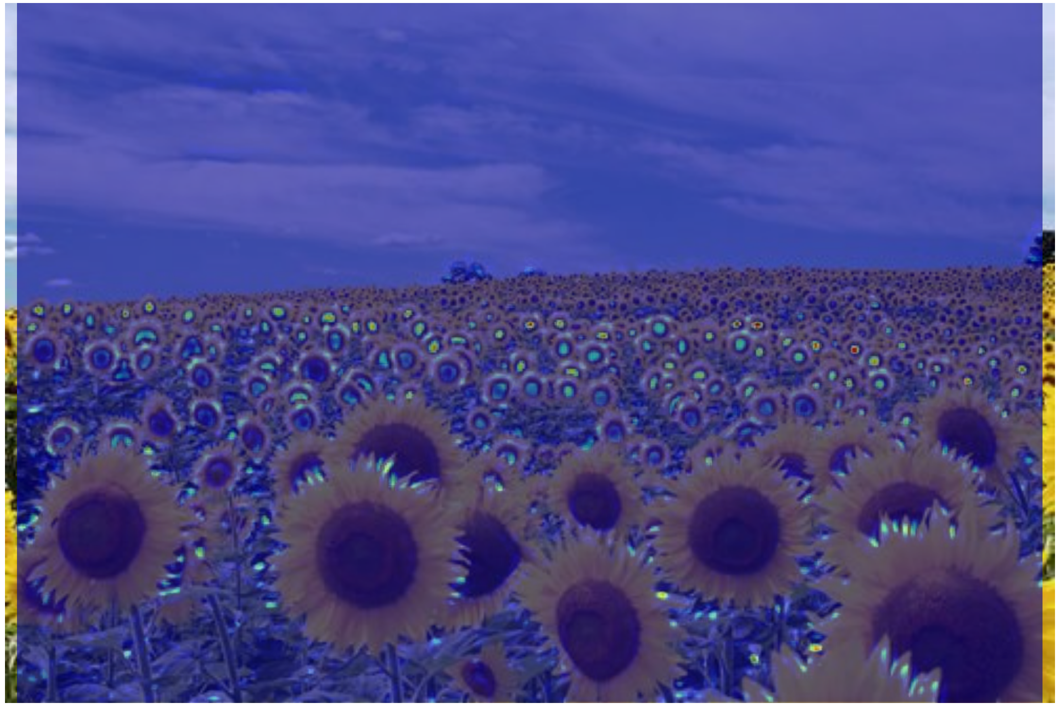
What happens if you apply different Laplacian filters?



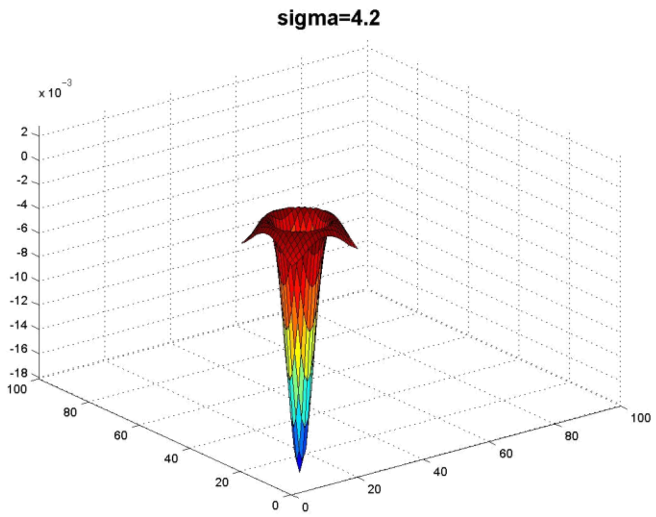
Full size

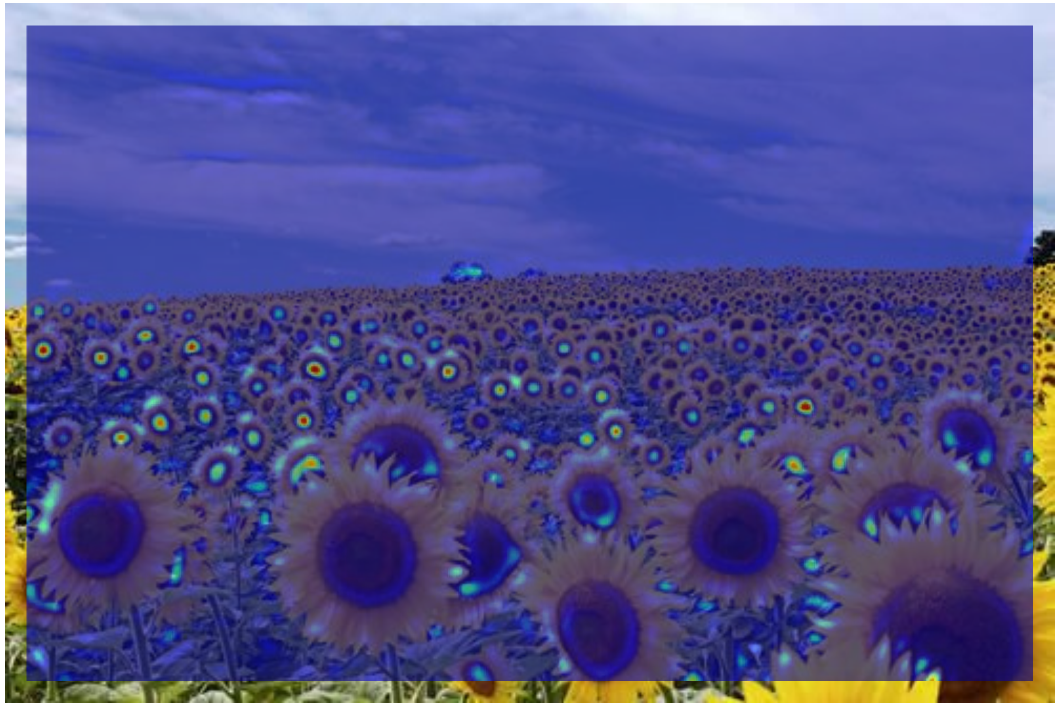
3/4 size

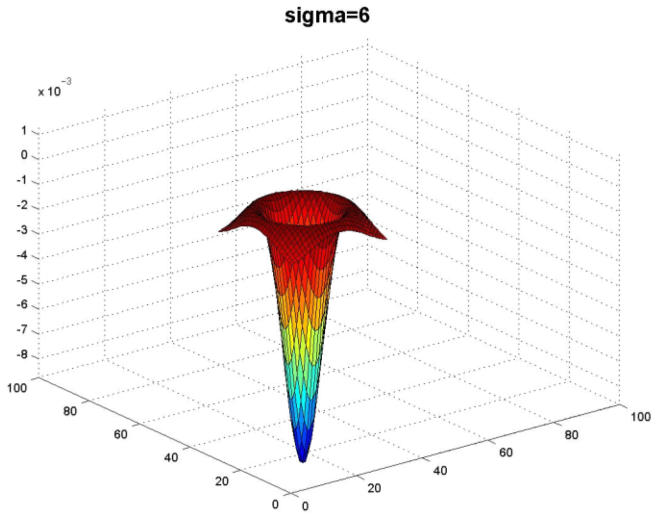


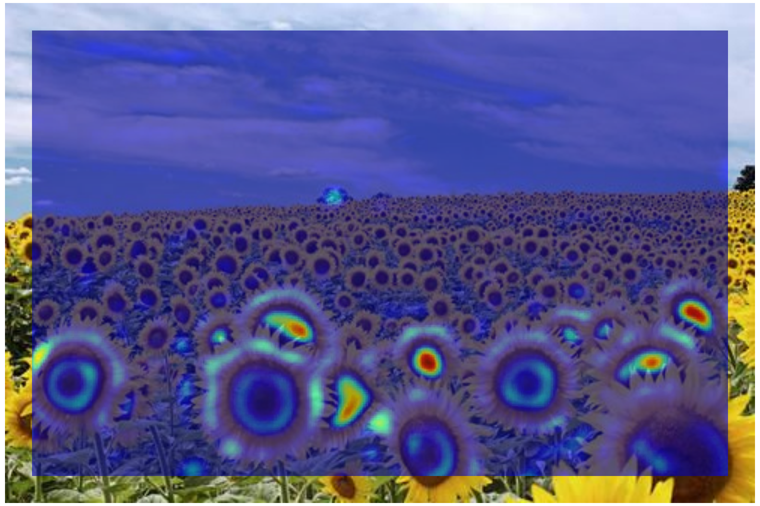


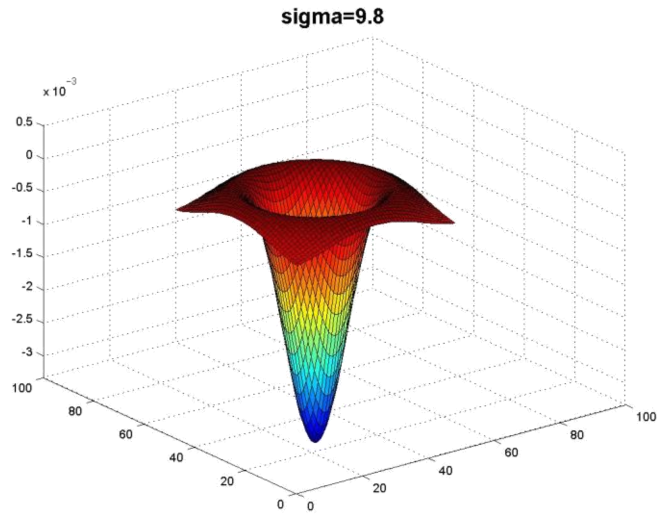
jet color scale
blue: low, red: high

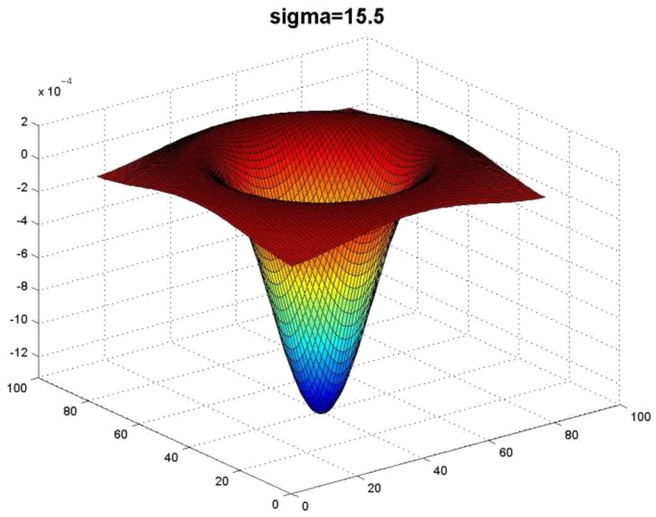

sigma=4.2
x 10^-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100

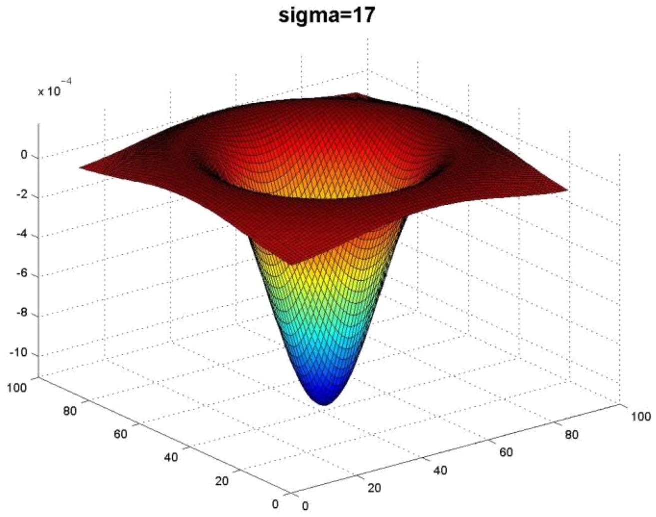



sigma=6
x 10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
-8
100
80
60
40
20
0
0
20
40
60
80
100




sigma=9.8
x 10^-3
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
100
80
60
40
20
0
0
20
40
60
80
100

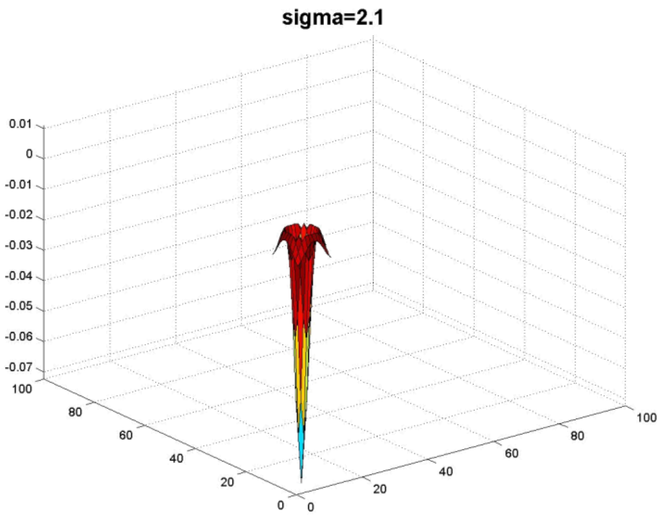

sigma=15.5
x 10^-4
2
0
-2
-4
-6
-8
-10
-12
100
80
60
40
20
0
0
20
40
60
80
100

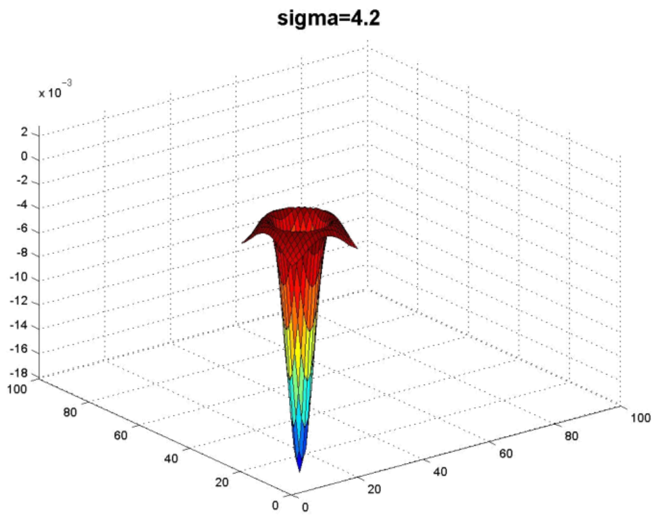

sigma=17
x 10^-4
0
-2
-4
-6
-8
-10
100
80
60
40
20
0
0
20
40
60
80
100

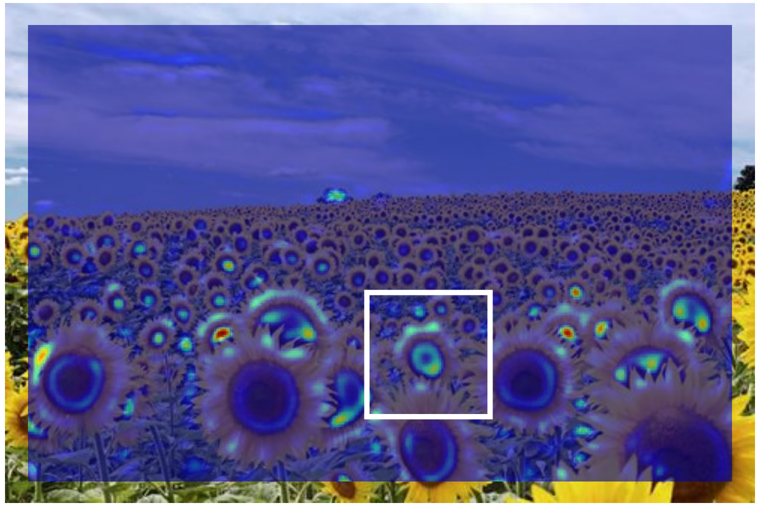
What happened when you applied different Laplacian filters?

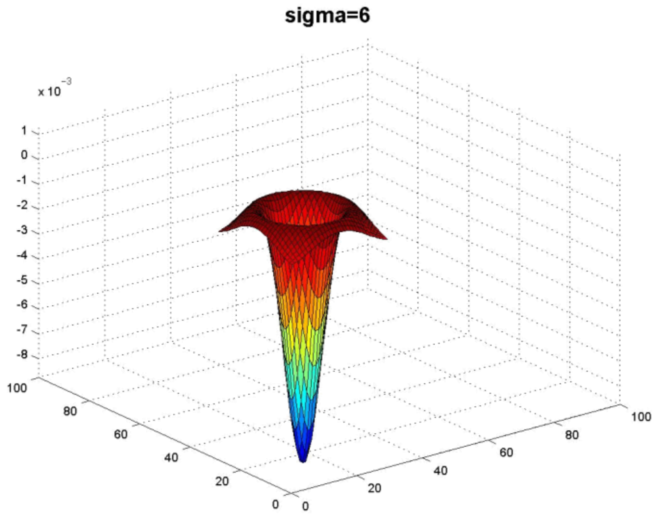
Full size

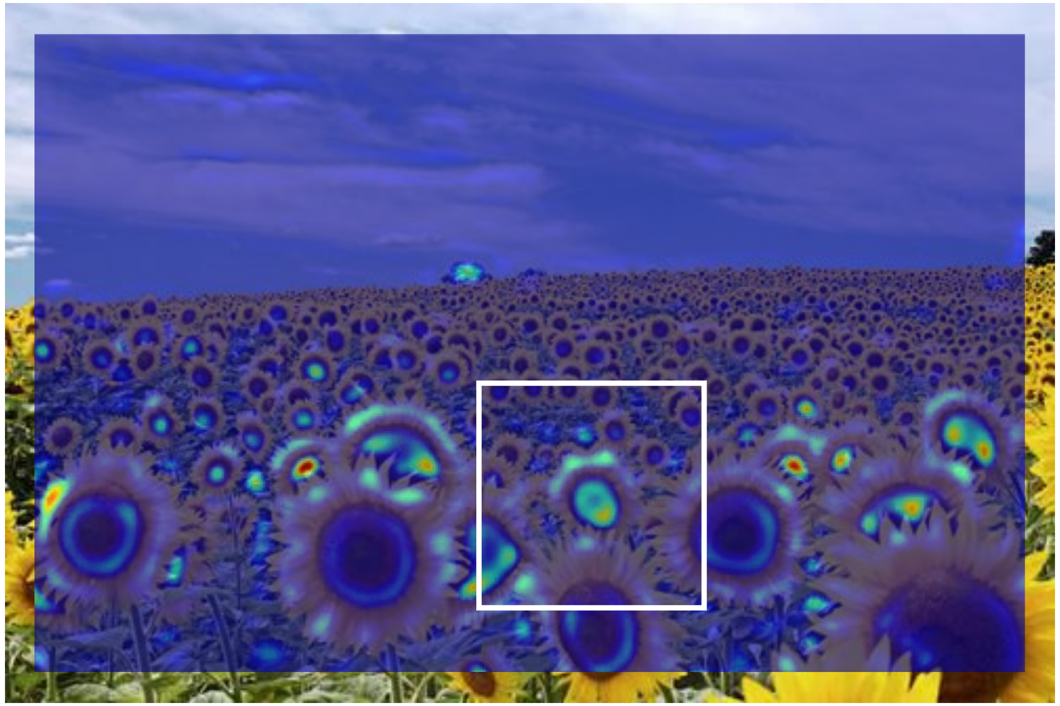
3/4 size


sigma=2.1
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
100
80
60
40
20
0
0
20
40
60
80
100

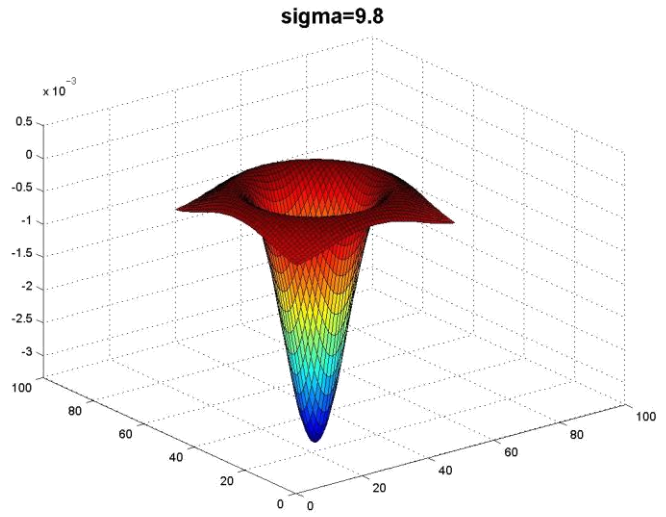

sigma=4.2
x 10^-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100



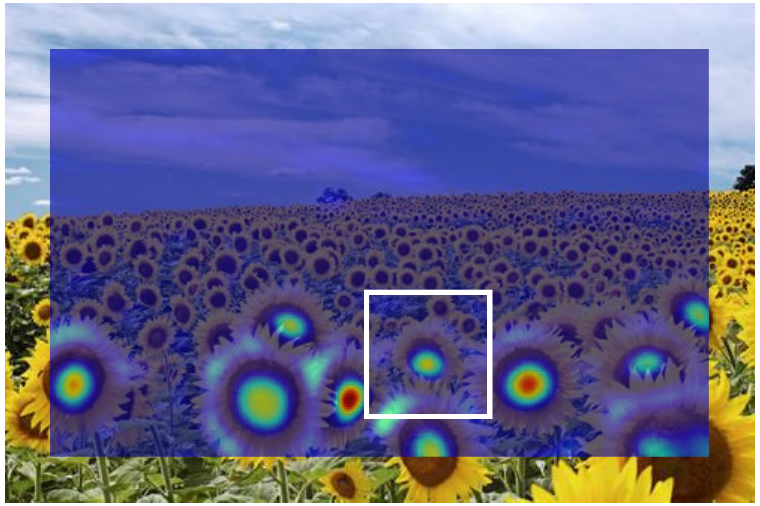

sigma=6
x 10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
-8
100
80
60
40
20
0
0
20
40
60
80
100

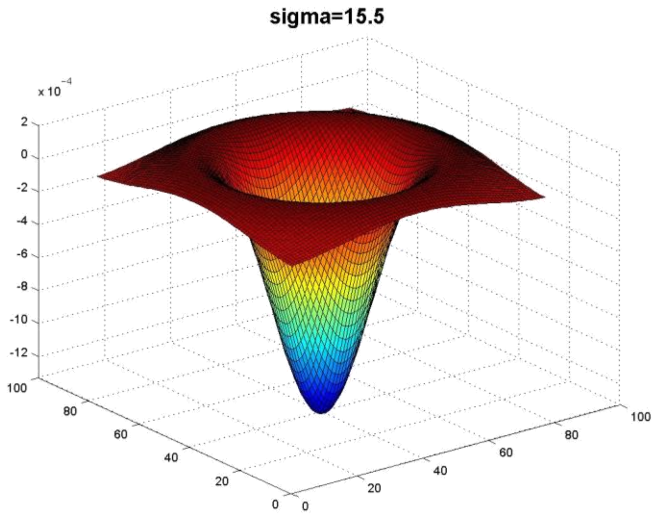


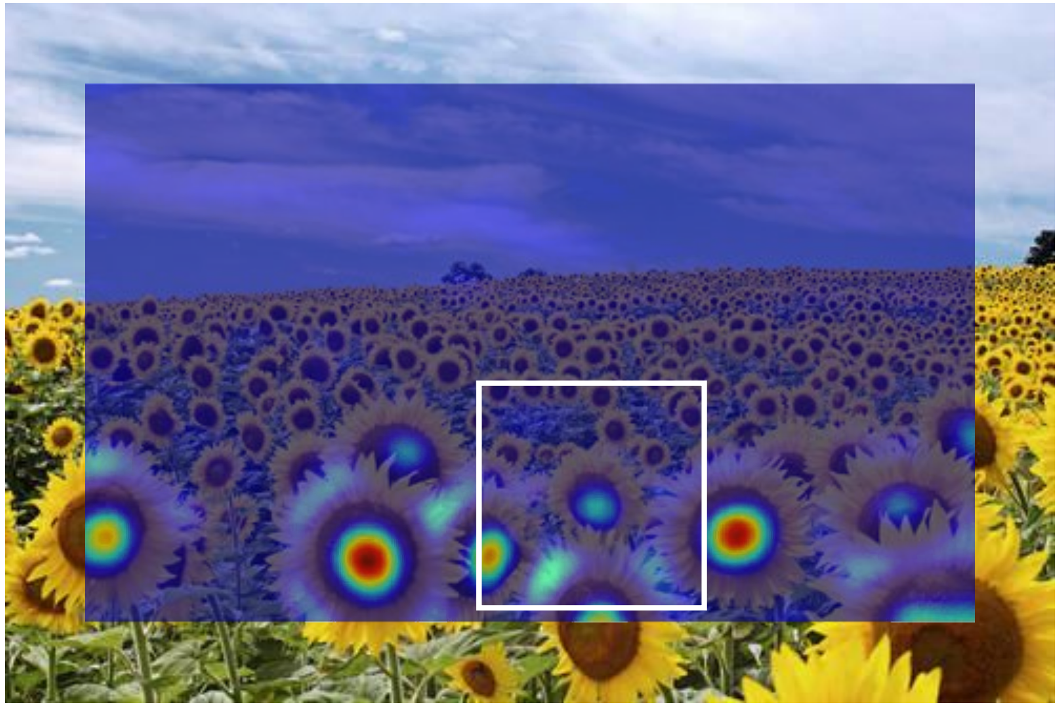

peak!

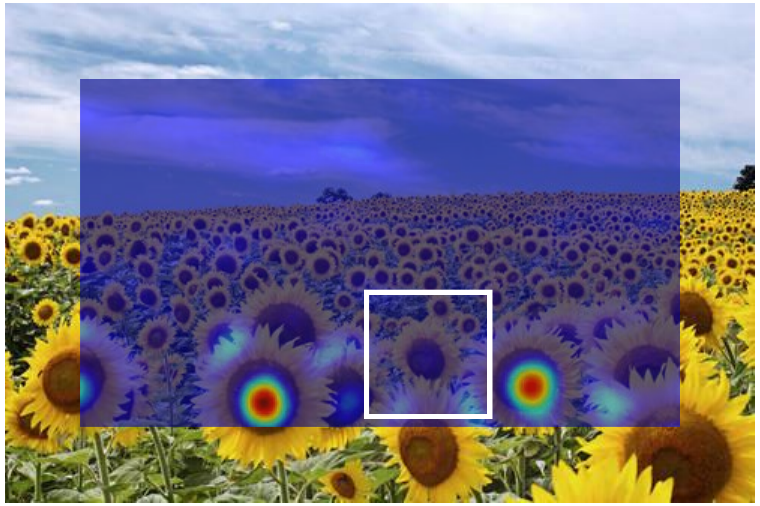

sigma=9.8
x 10^-3
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
100
80
60
40
20
0
0
20
40
60
80
100

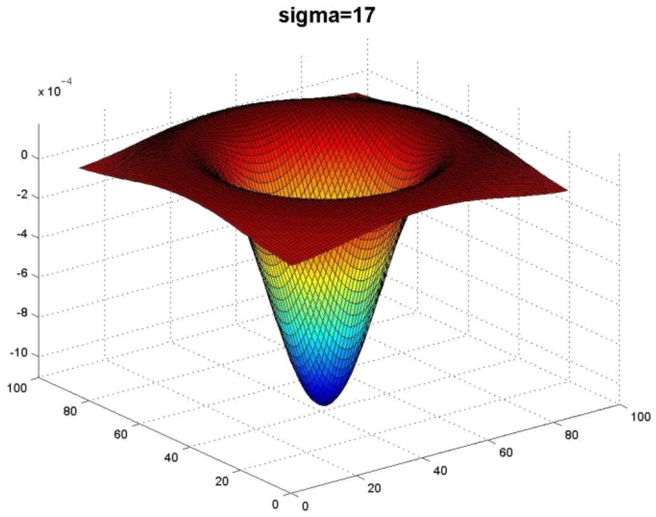

peak!

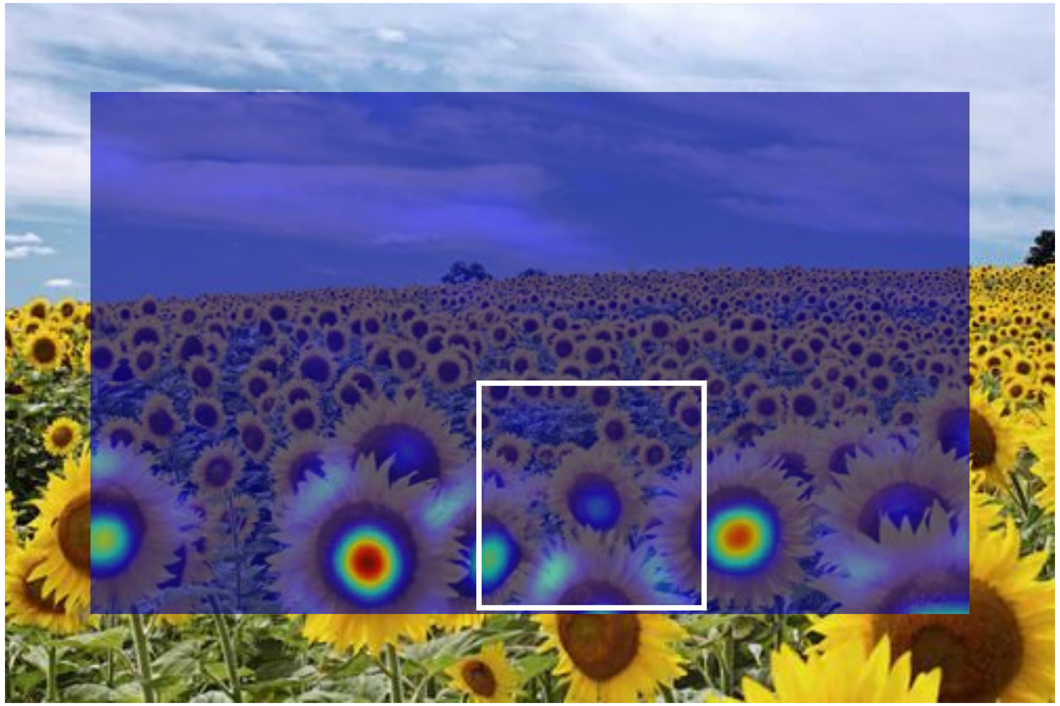


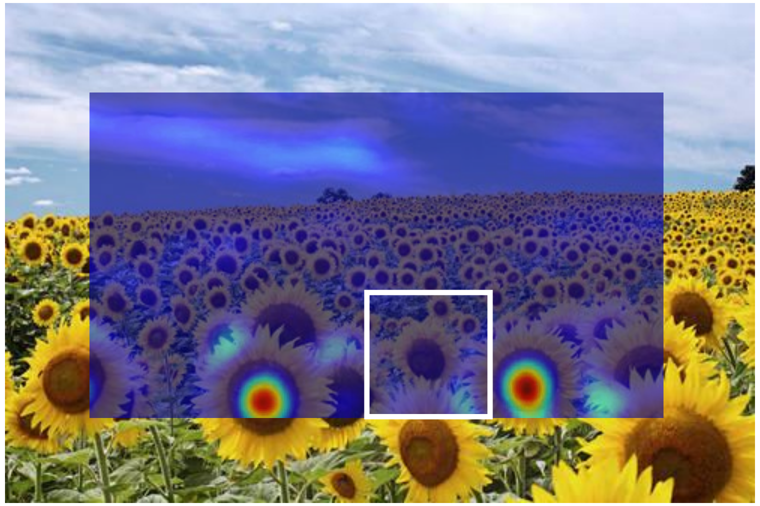

sigma=15.5
x 10^-4
2
0
-2
-4
-6
-8
-10
-12
100
80
60
40
20
0
0
20
40
60
80
100






sigma=17
x 10^-4
0
-2
-4
-6
-8
-10
100
80
60
40
20
0
0
20
40
60
80
100

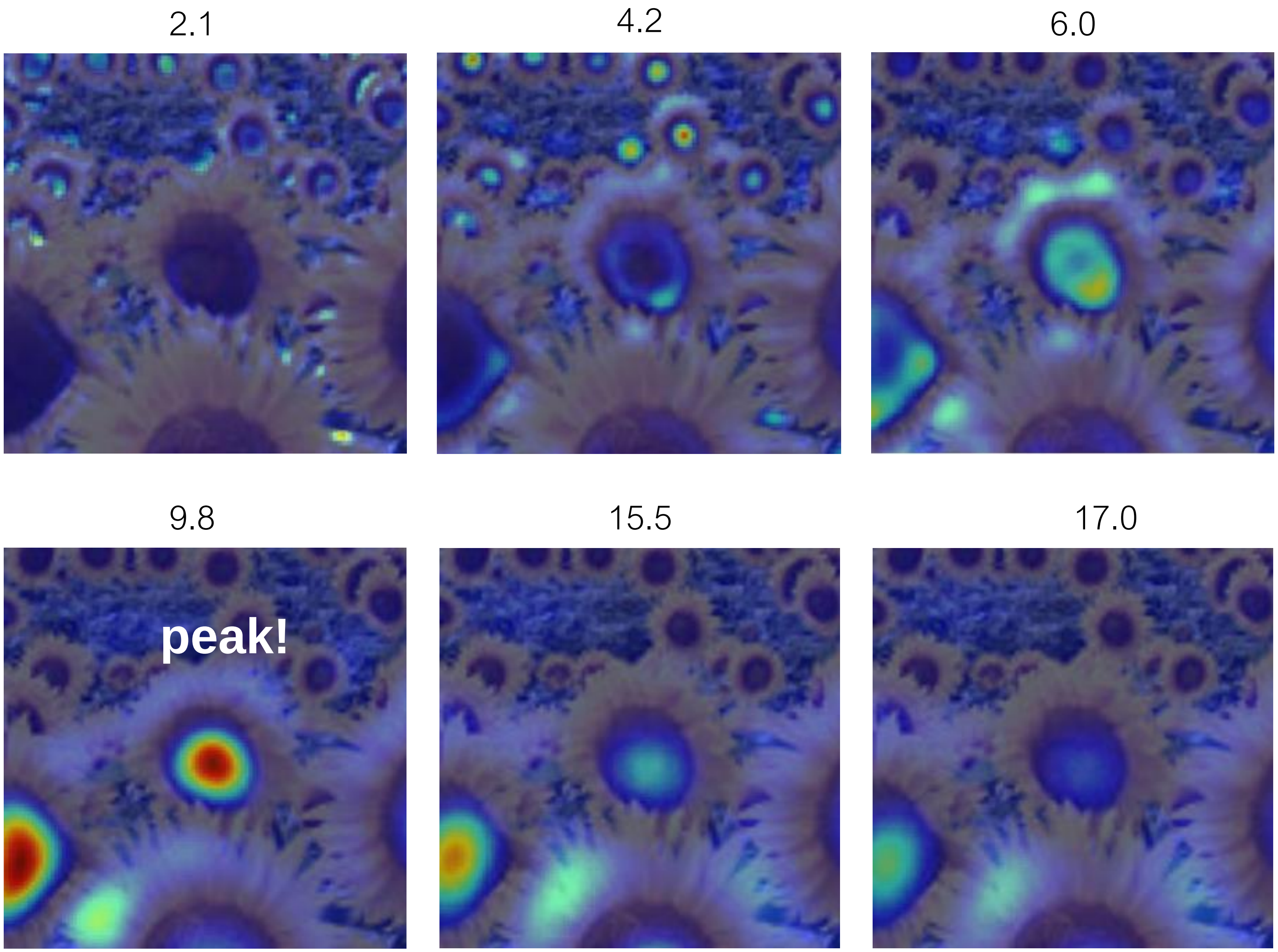


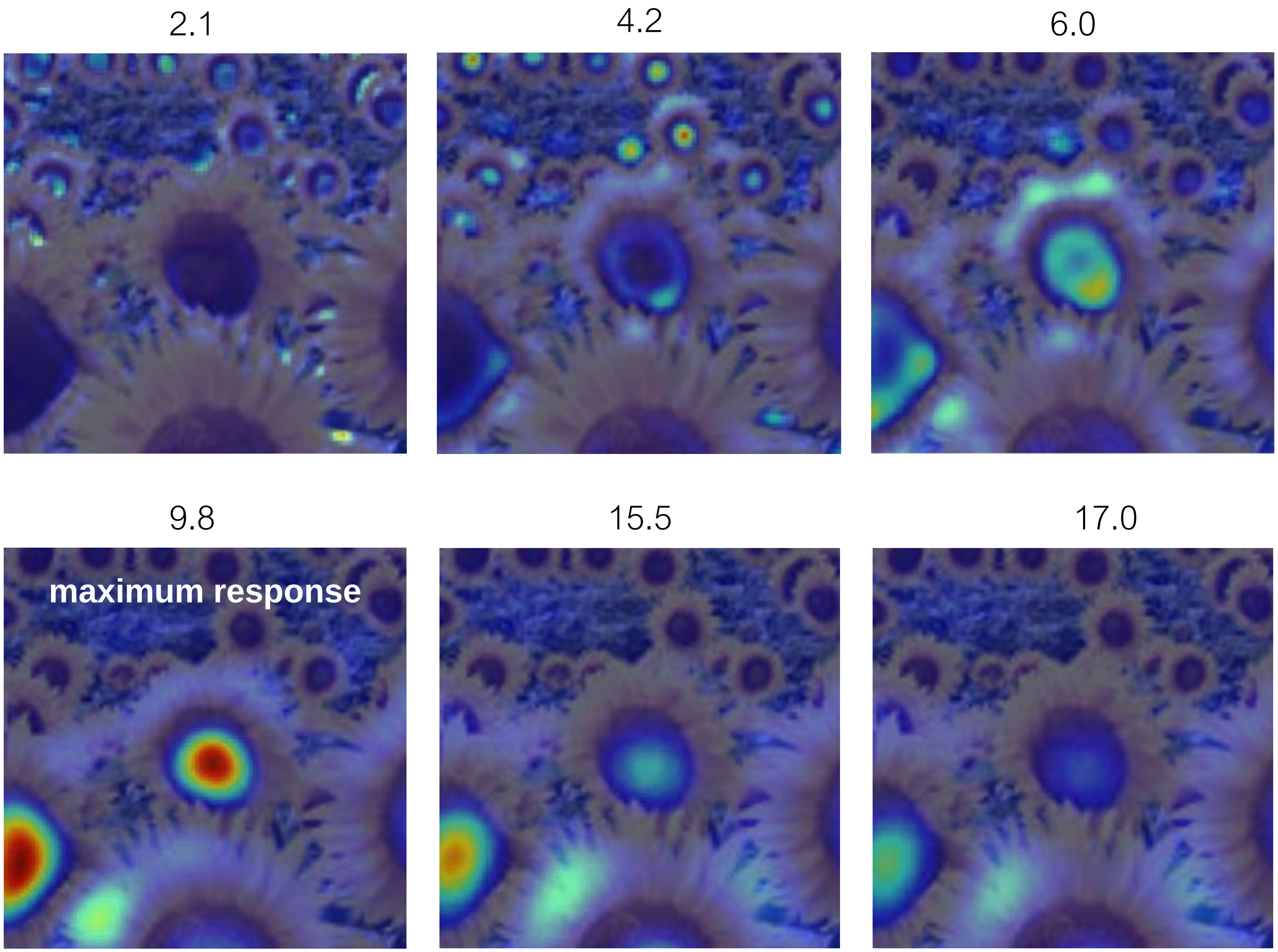


What happened when you applied different Laplacian filters?

Full size

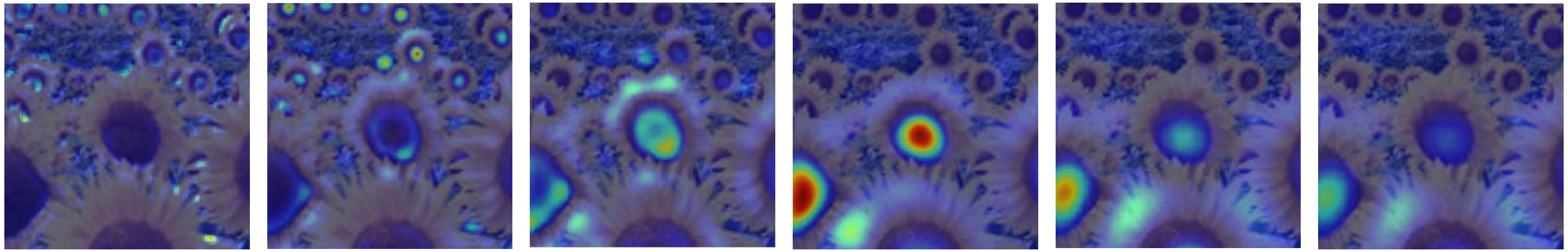
3/4 size


2.1
4.2
6.0
9.8
peak!
15.5
17.0


2.1
4.2
6.0
9.8
maximum response
15.5
17.0

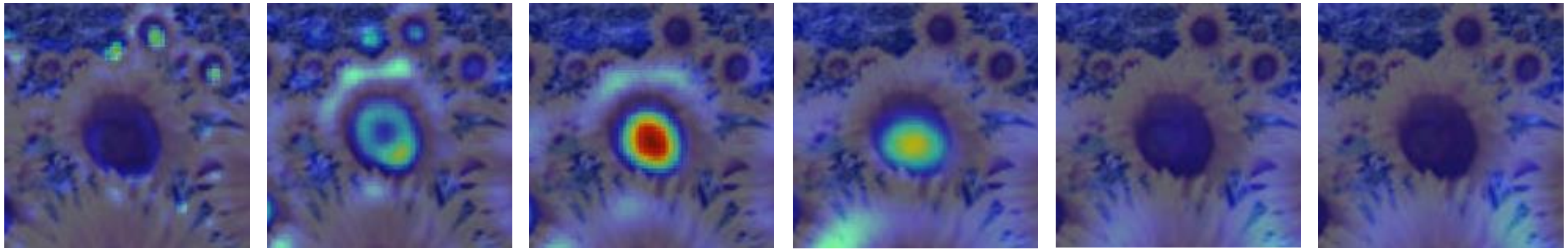
optimal scale

2.1 4.2 6.0 9.8 15.5 17.0



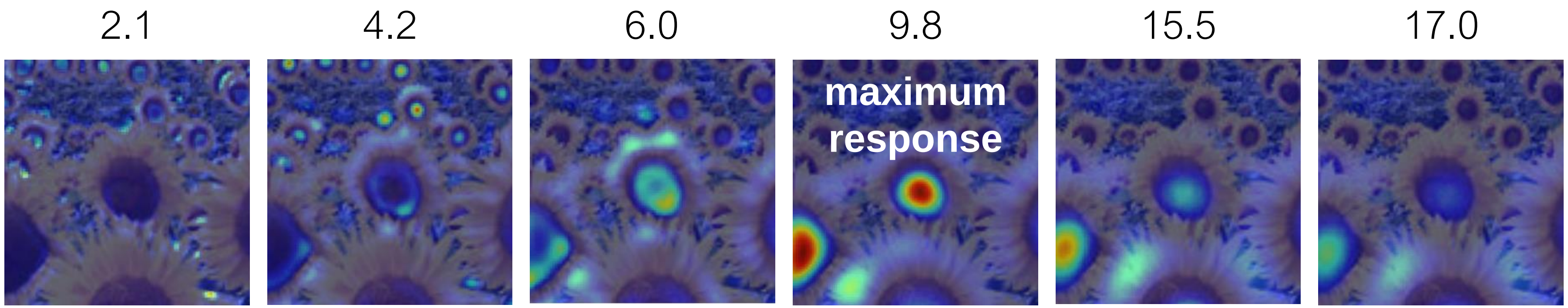
Full size image

2.1 4.2 6.0 9.8 15.5 17.0

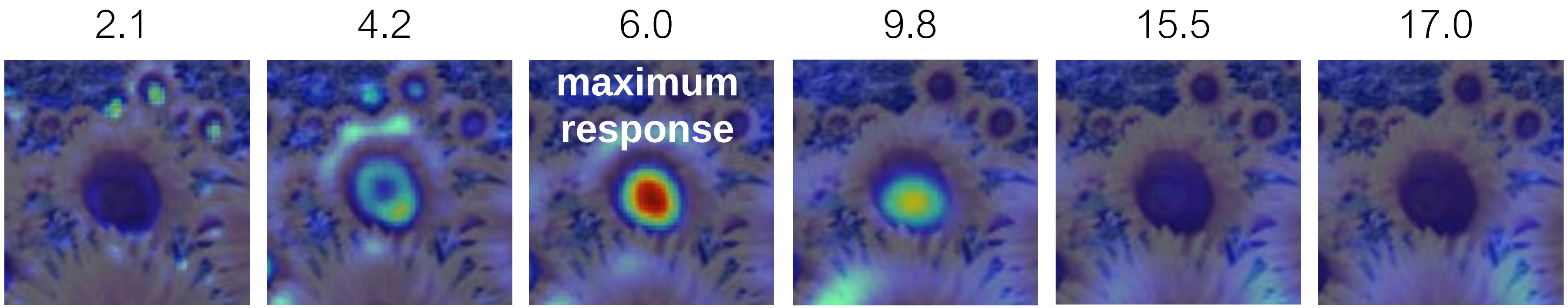


3/4 size image

optimal scale

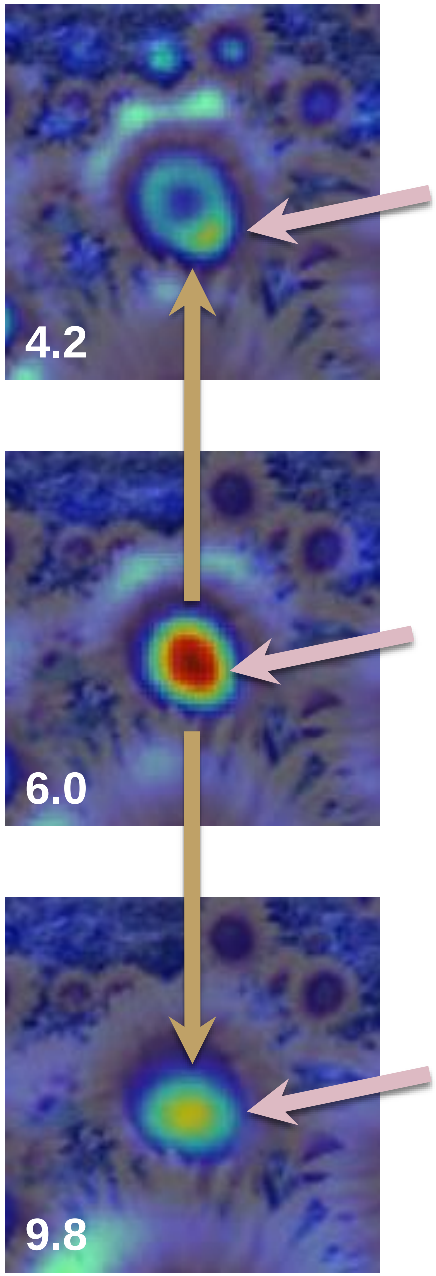


Full size image



3/4 size image

cross-scale maximum



local maximum

local maximum

local maximum

Scale Invariant Detection

- Functions for determining scale $f = \text{Kernel} * \text{Image}$

Kernels:

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

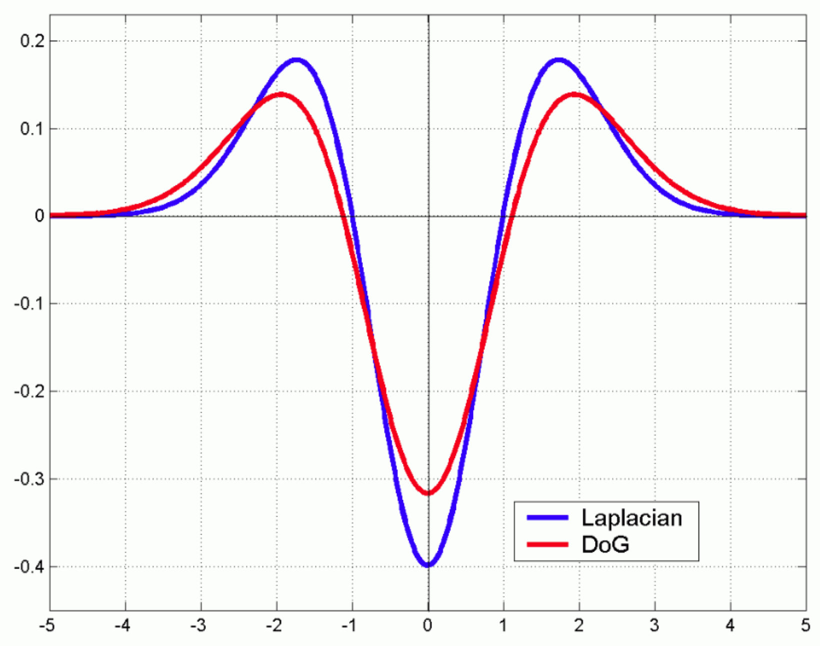
(Laplacian)

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

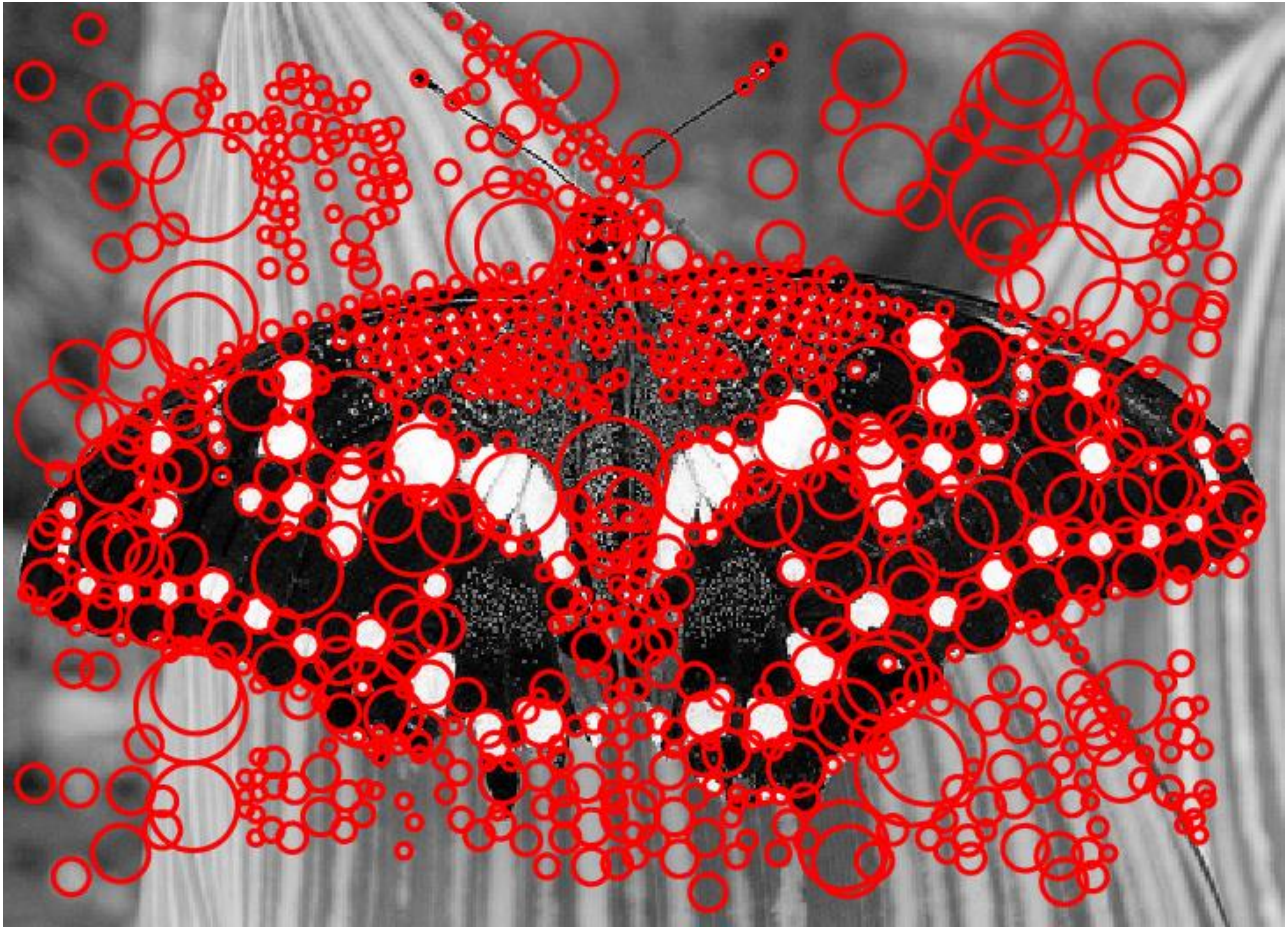
(Difference of Gaussians)

where Gaussian

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



Note: The LoG and DoG operators are both rotation equivariant



References

Basic reading:

- Szeliski textbook, Sections 4.1.

Questions?