Feature Detection and Matching: Part 2

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

CIVE 497 – CIVE 700: Smart Structure Technology

Last updated: 2024-03-02

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

SIFT: Scale-Invariant Feature Transform

Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

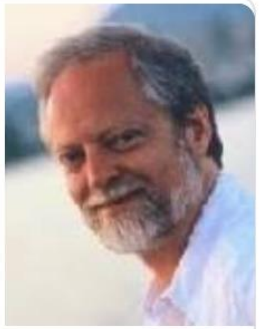
January 5, 2004

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

David G. Lowe

Canadian computer scientist



David G. Lowe is a Canadian computer scientist working for Google as a Senior Research Scientist. He was a former professor in the Computer Science Department at the University of British Columbia and New York University. Wikipedia

Known for: Scale-invariant feature transform

Residence: Seattle, Washington, United States

Alma maters: The University of British Columbia, Stanford University (1985, PhD)

Academic advisor: Thomas Binford

Notable student: Ken Perlin

Distinctive image features from scale-invariant keypoints

DG Lowe - International journal of computer vision, 2004 - Springer

This paper presents a method for extracting **distinctive** invariant **features** from images that can be used to perform reliable matching between different views of an object or scene. The **features** are invariant to **image** scale and rotation, and are shown to provide robust matching …

☆ 99 Cited by 59710 Related articles All 168 versions

Steps for Extracting SIFT Keypoints

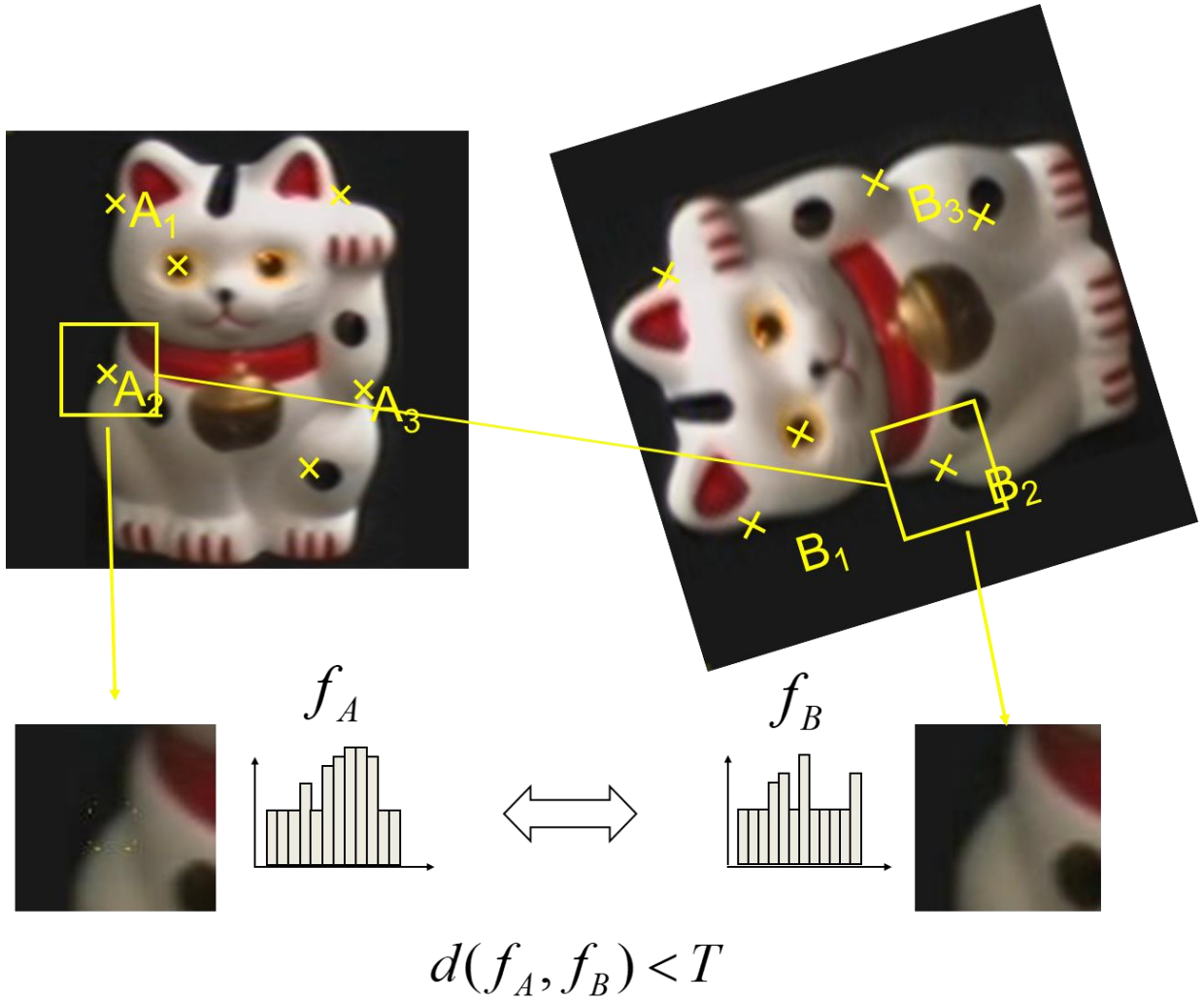
Scale-space extrema detection: Difference-of-Gaussian function is used to identify potential interest points that are invariant to scale and orientation.

Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.

Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

Revisit: Overview of Feature Matching



Step 1. Find a set of distinctive key-points

Step 2. Define a region around each keypoint

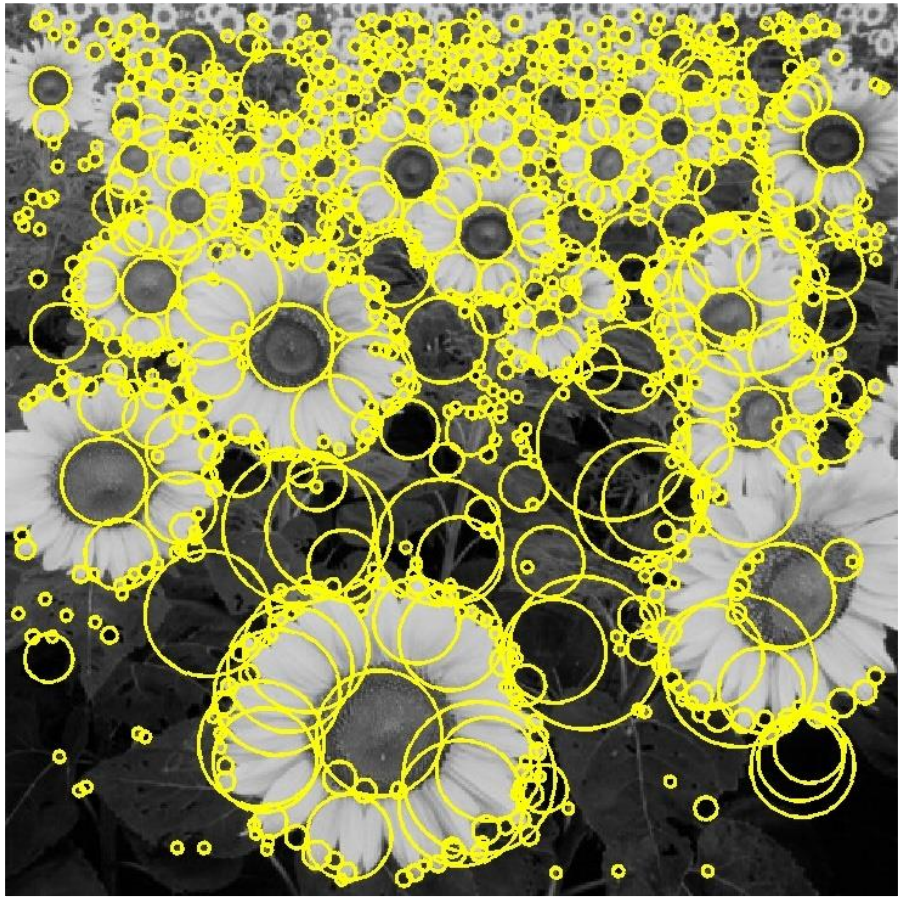
Step 3. Extract and normalize the region content

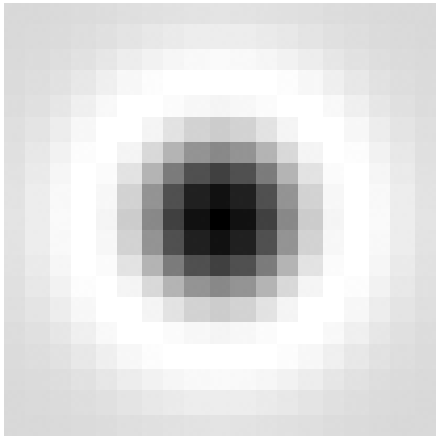
Step 4. Compute a local descriptor from the normalized region

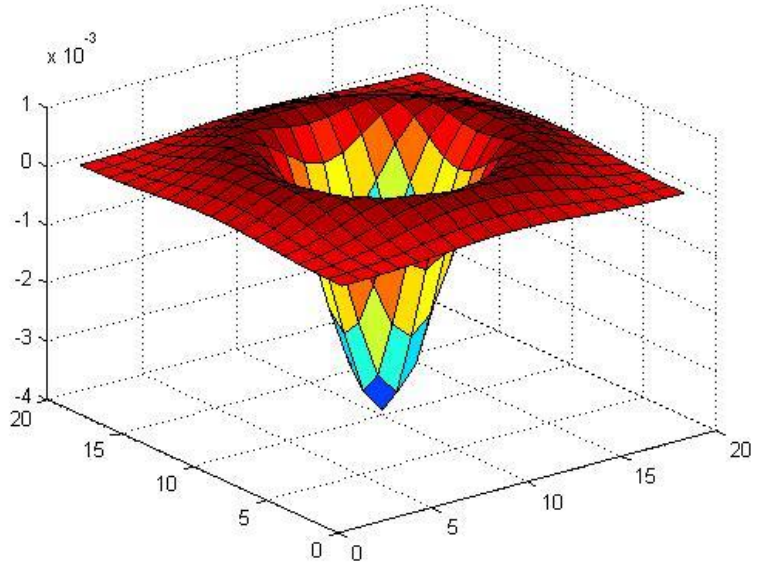
Step 5. Match local descriptors

Basic Idea: Using Laplacian of Gaussian as a Blob Filter

Convolve the image with a “blob filter” at multiple scales and look for extrema of filter response in the resulting *scale space*



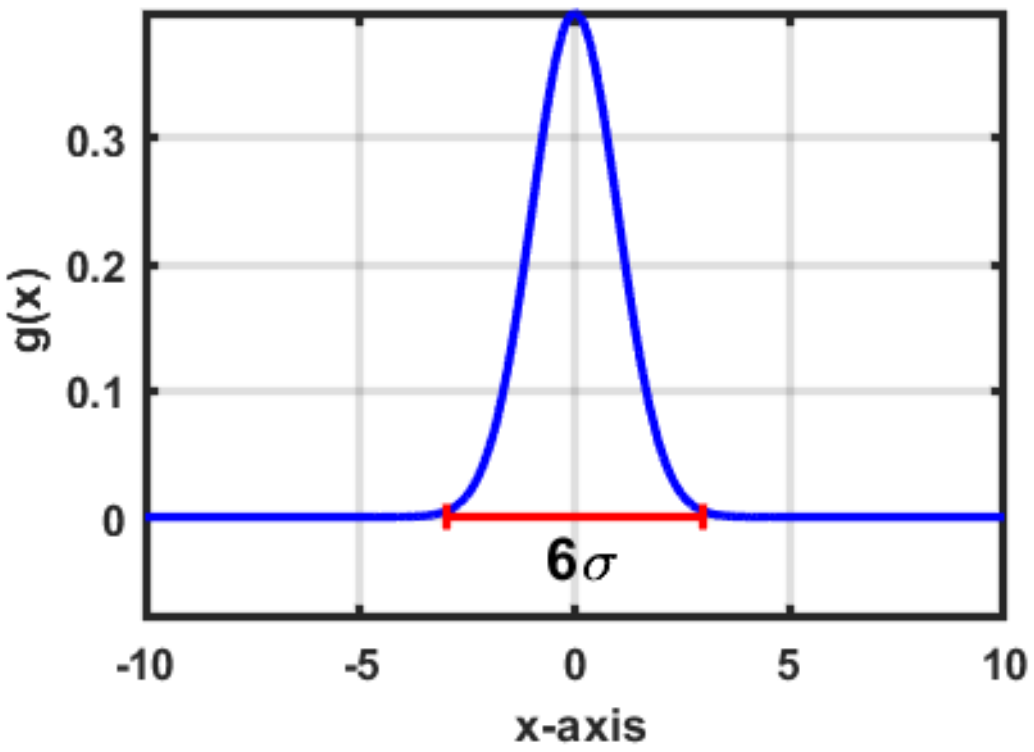


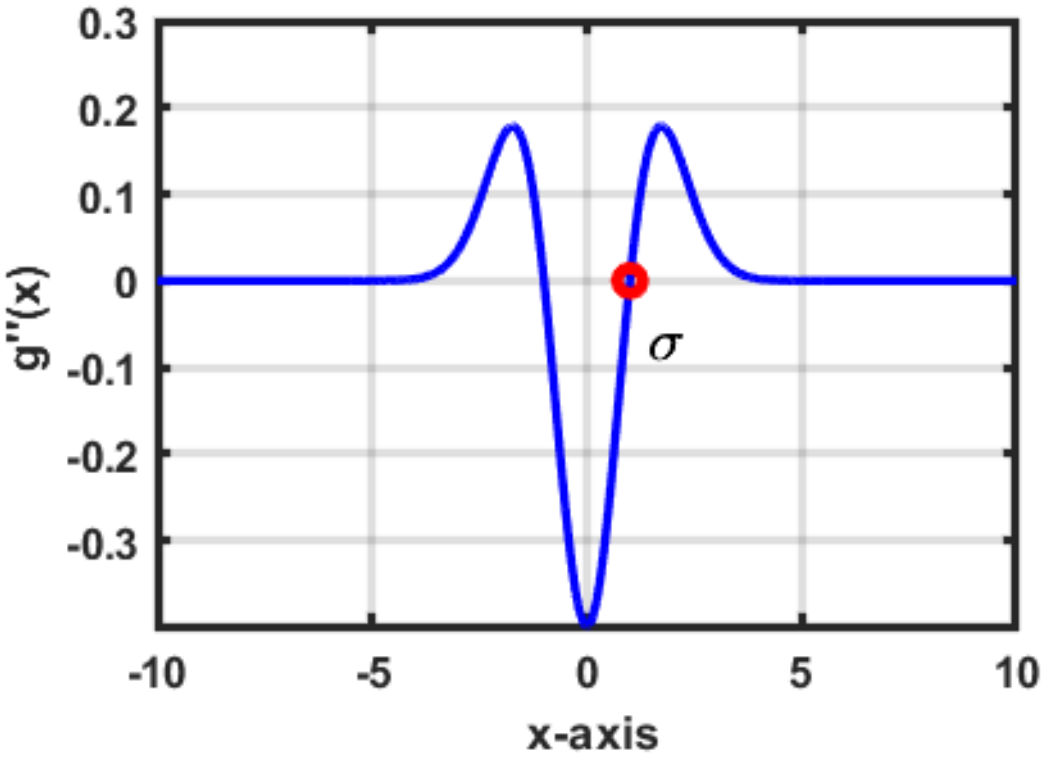


$$\nabla^2 G = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$$

Laplacian of Gaussian in 1D

Supplement



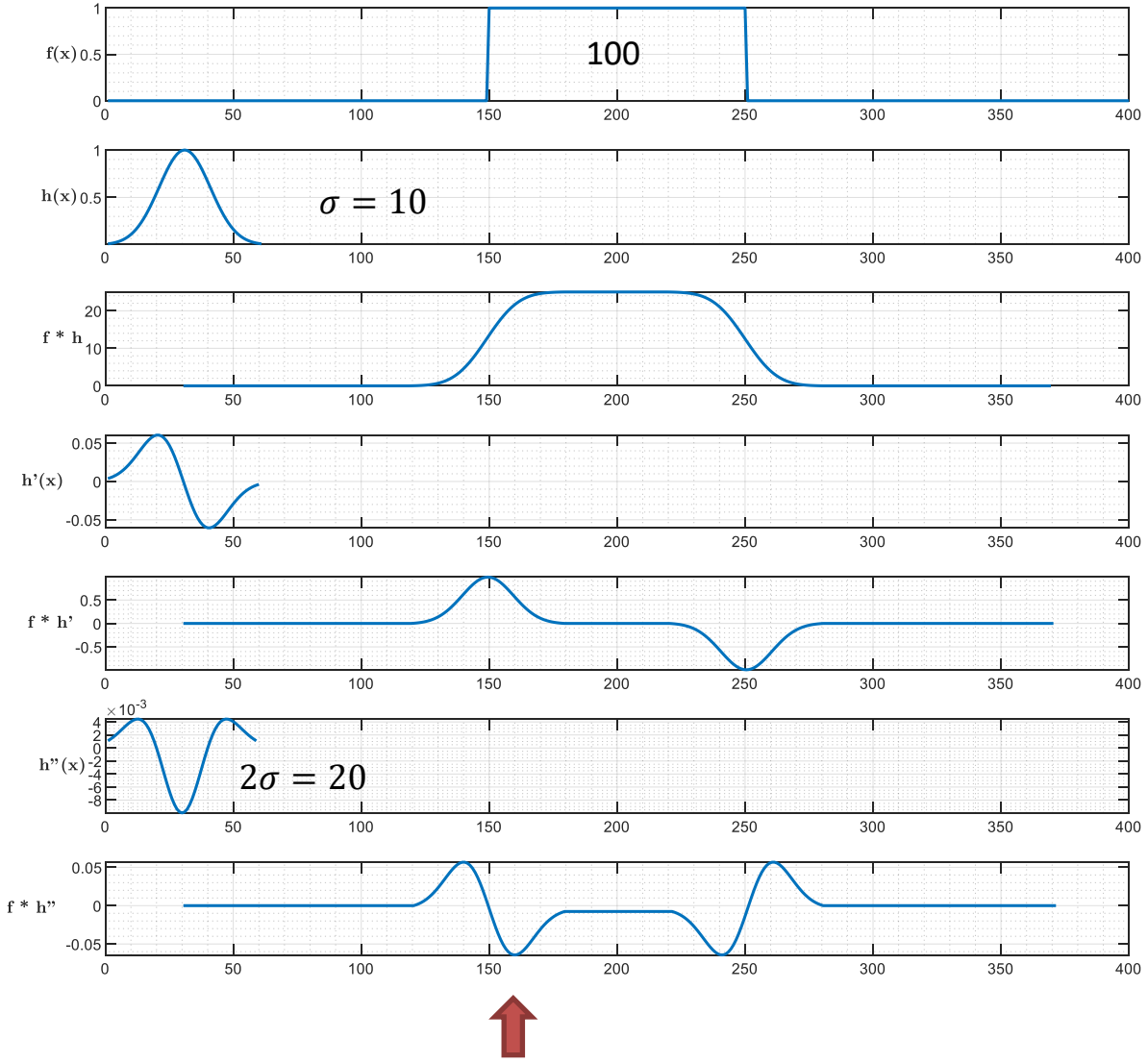


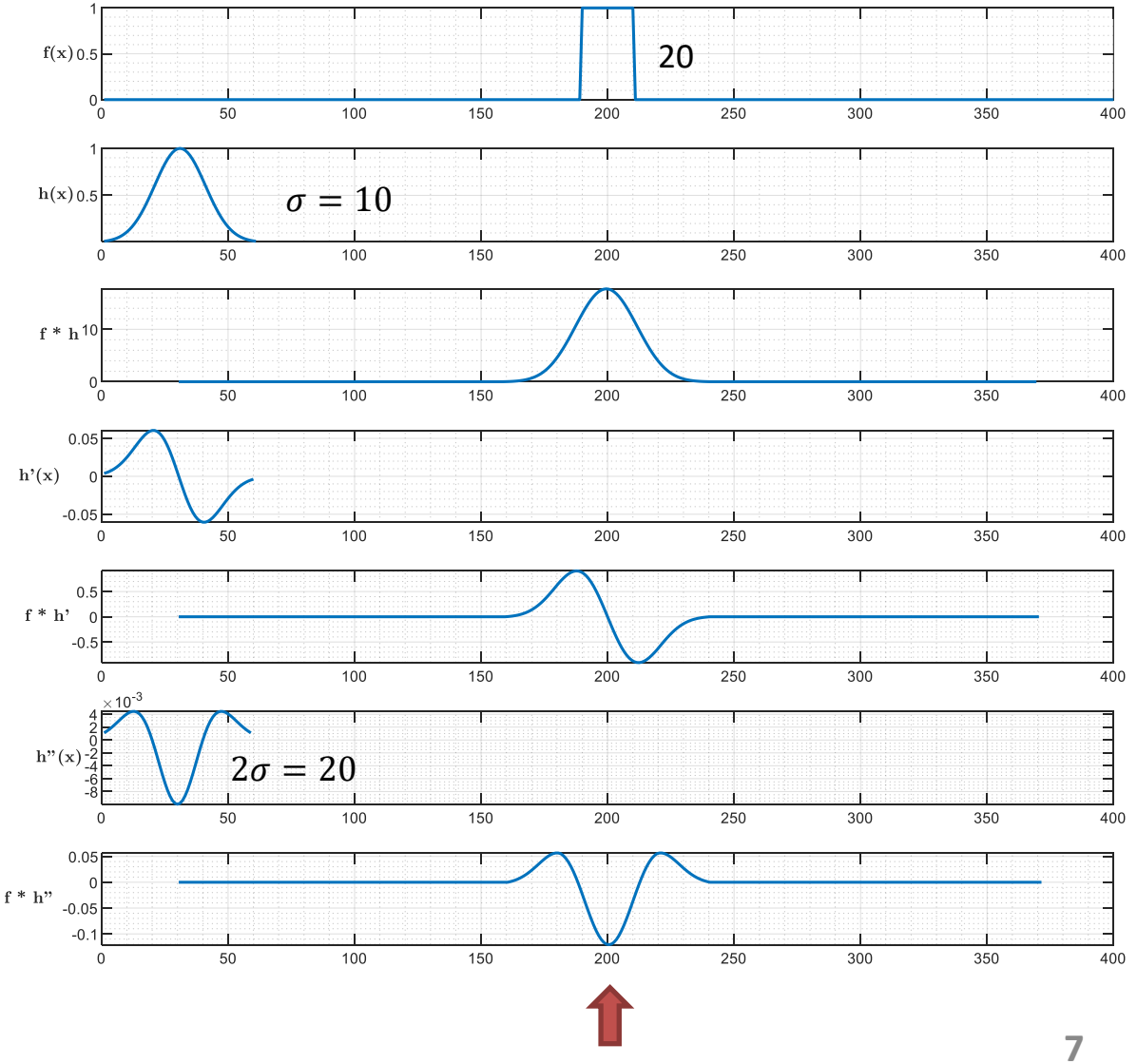
$$g(x) = -\frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{x^2}{2\sigma^2}}$$

$$g''(x) = -\frac{1}{\sqrt{2\pi}} \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) \exp^{-\frac{x^2}{2\sigma^2}}$$

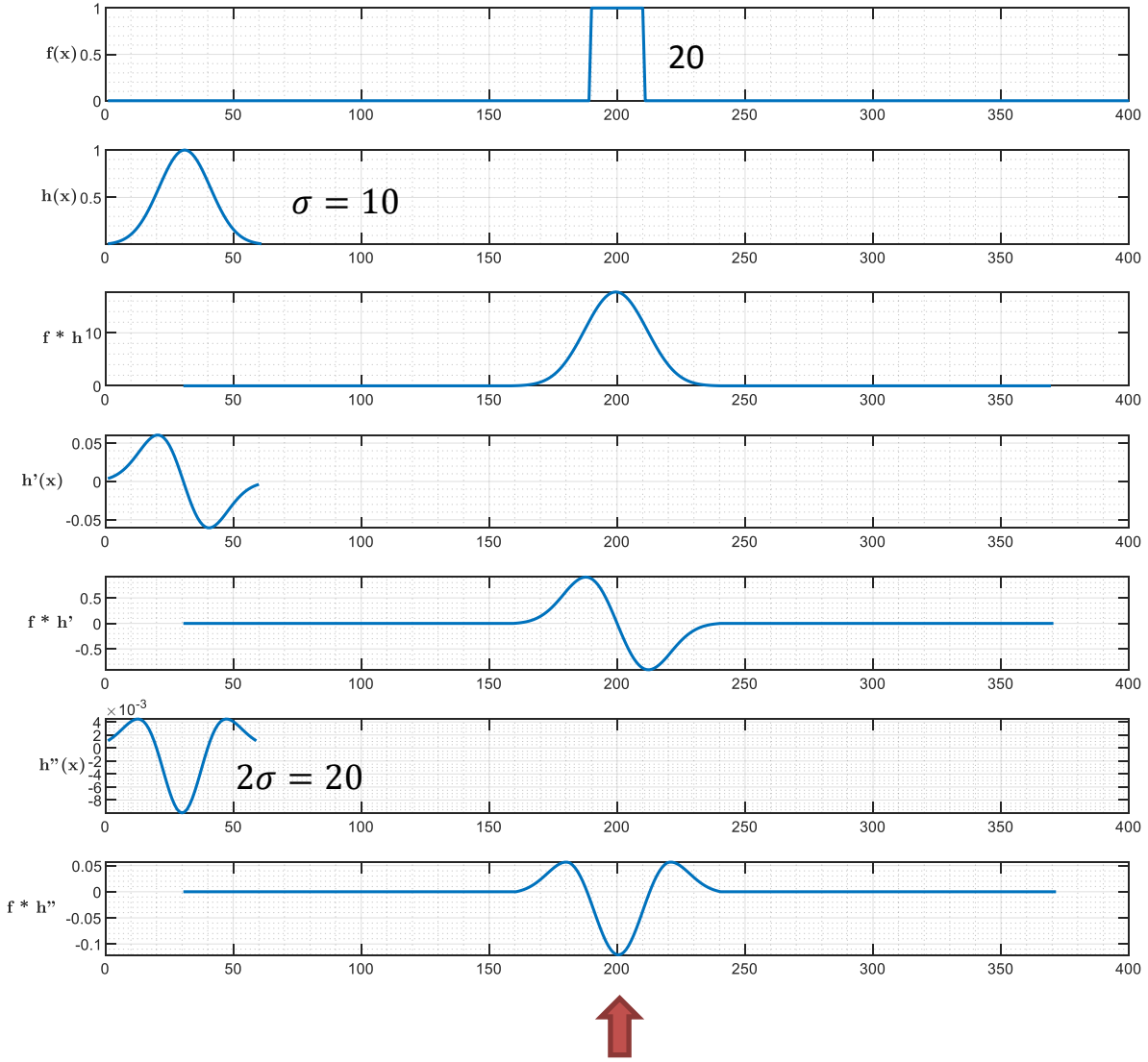
Tutorial

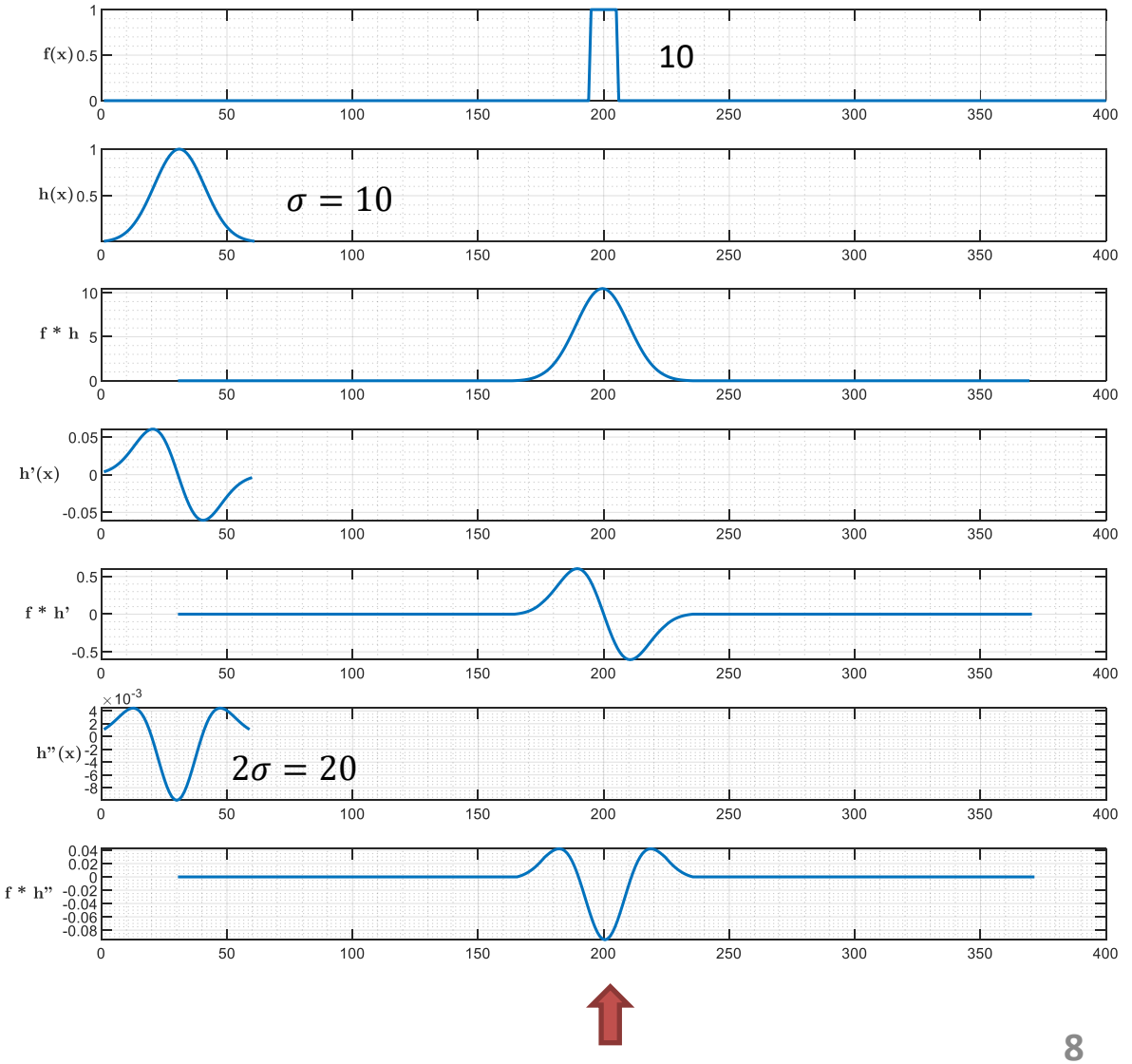
Example: Blob Filter (1D Case)





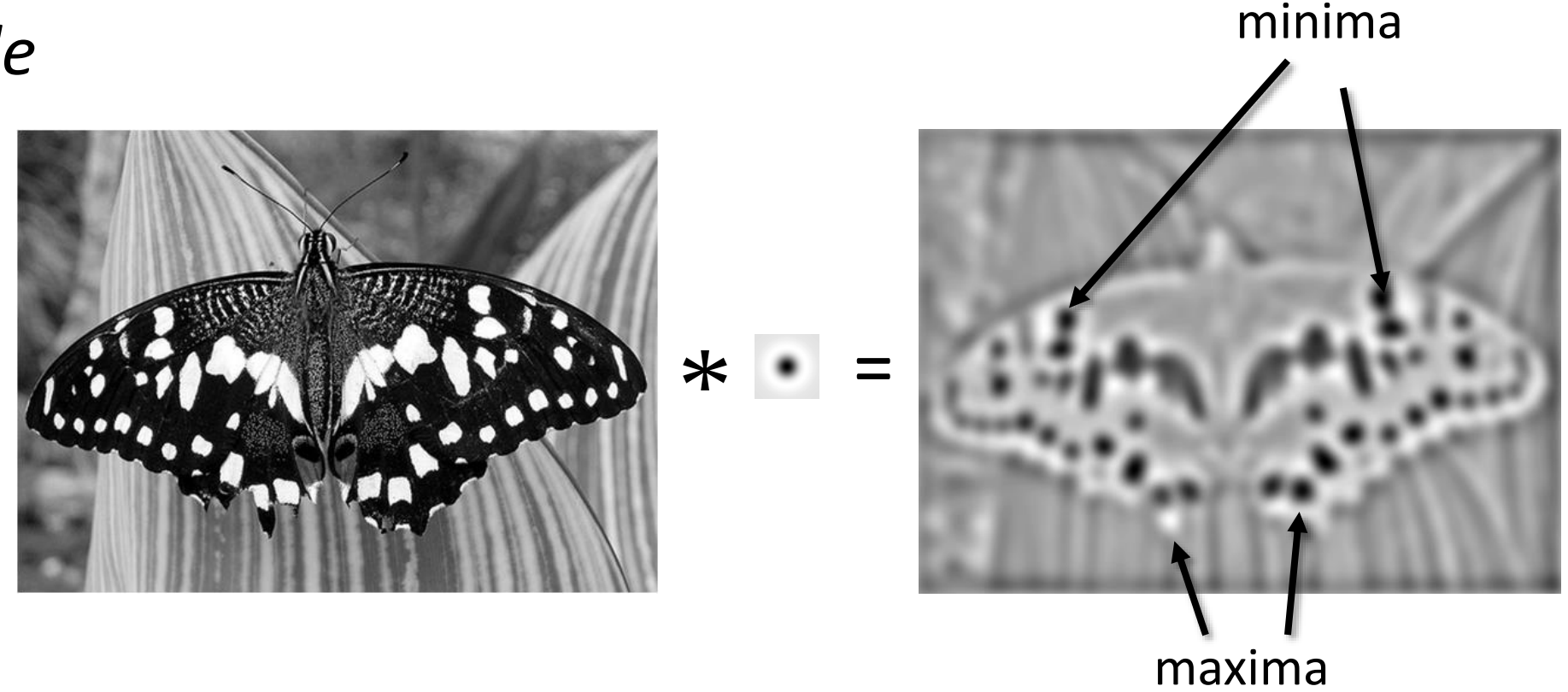
Example: Blob Filter (1D Case) - Continue



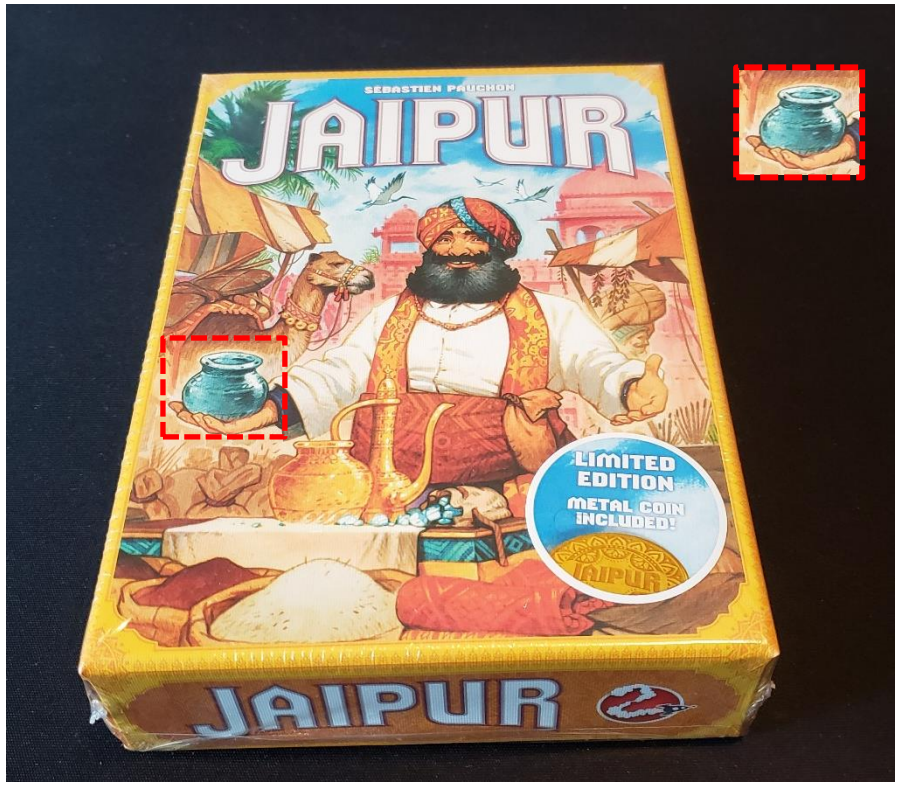


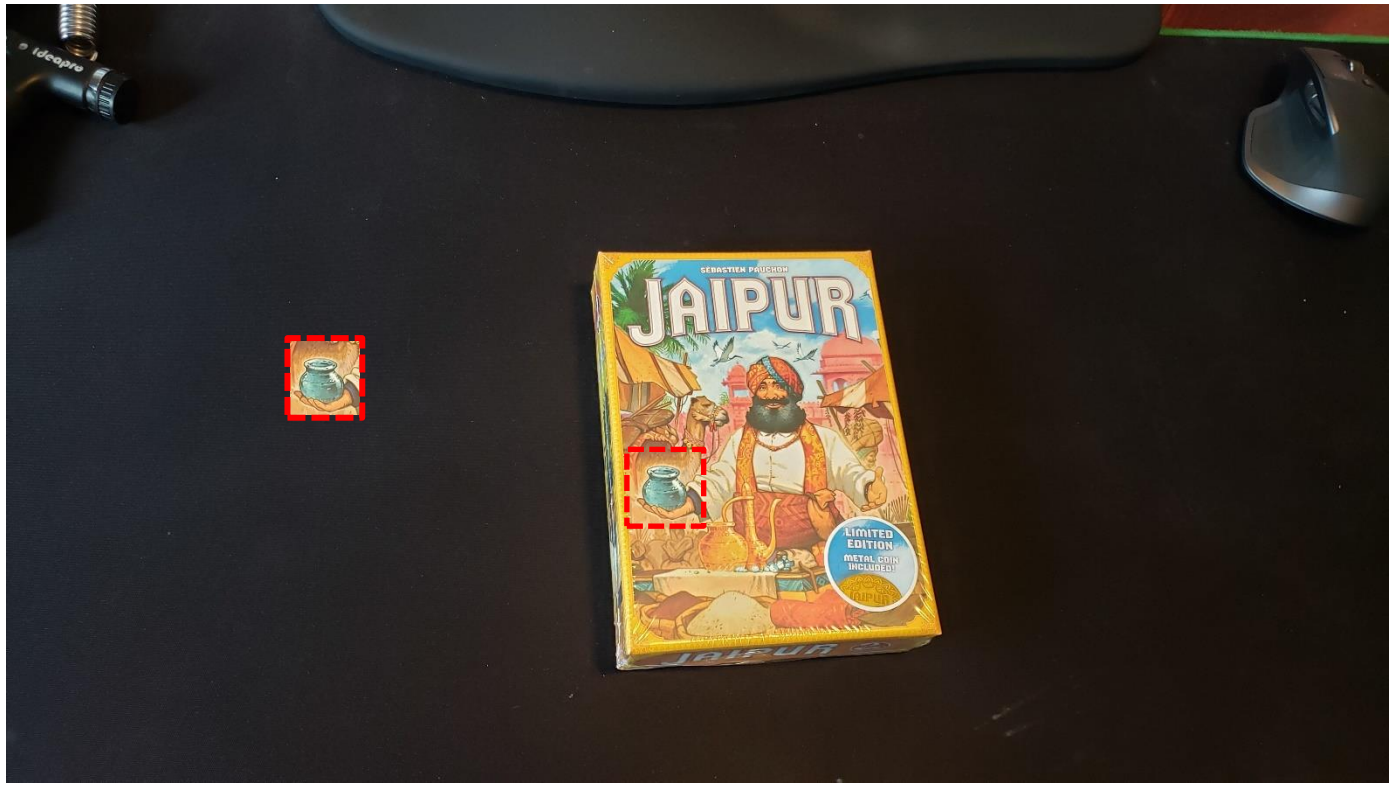
Blob Detection

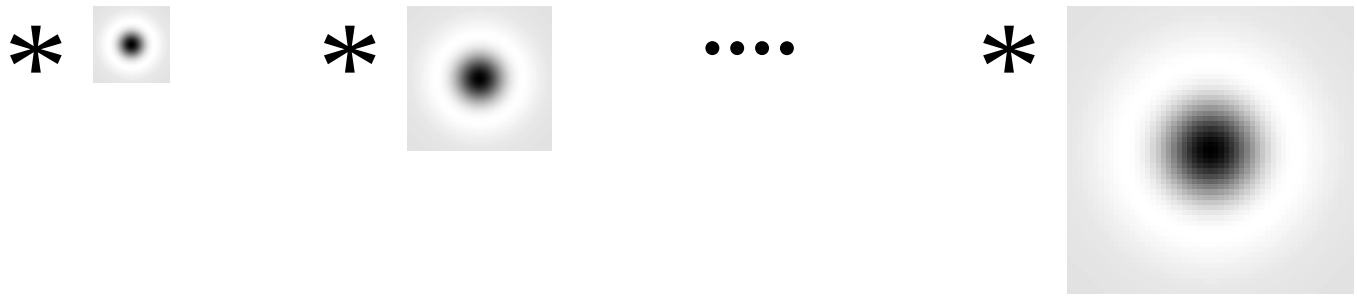
- Find maxima *and minima* of blob filter response in space *and scale*



Challenge: How To Determine Blob Sizes?

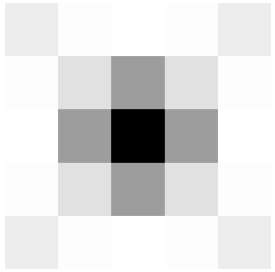




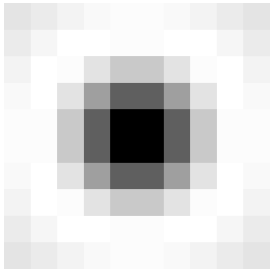


We do not know the size of each blob in advance, so we should detect blob with various sizes by changing the size of LOG filters. The problem is that this process is computationally demanding.

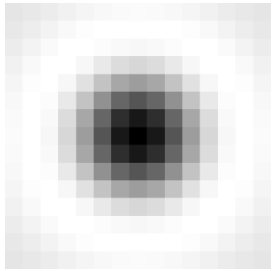
Example: Applying Blob (LOG) Filters with Different Size



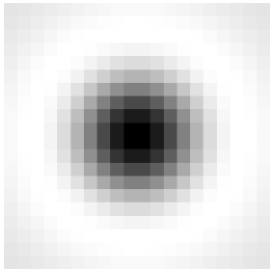
H1: 5x5 ($\sigma = 1$)



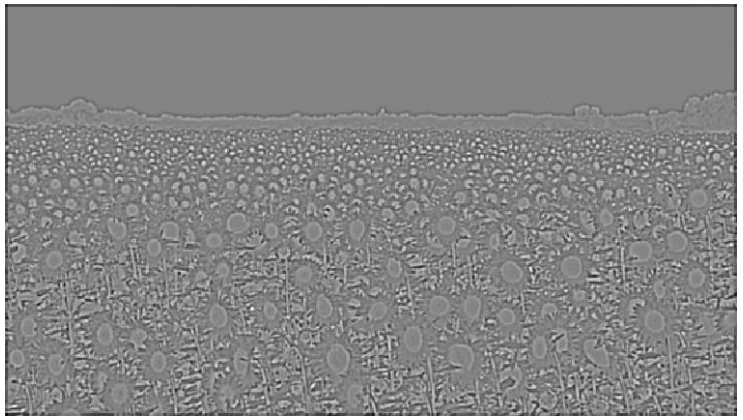
H2: 10x10 ($\sigma = 2$)



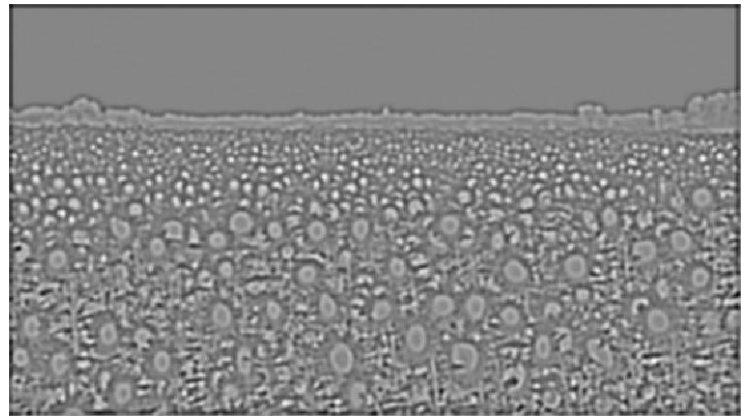
H3: 15x15 ($\sigma = 3$)



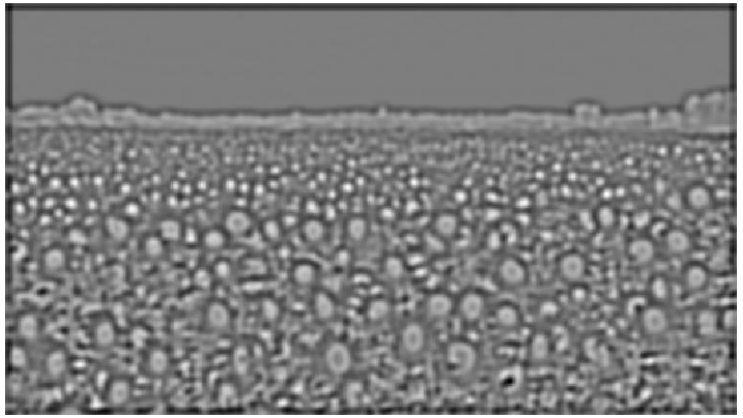
H4: 20x20 ($\sigma = 4$)



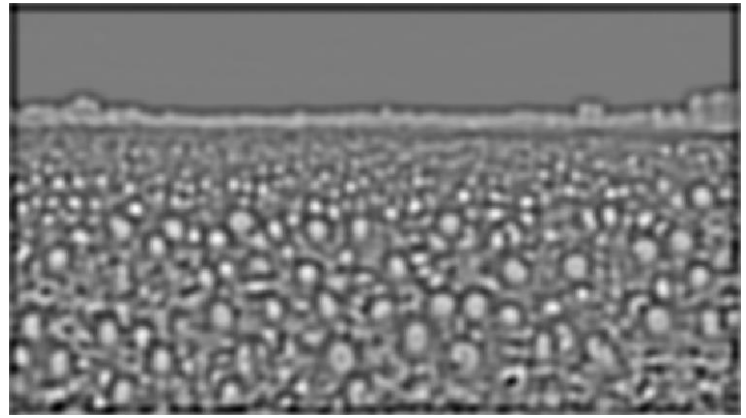
Img * H1



Img * H2



Img * H3



Img * H4

Tutorial

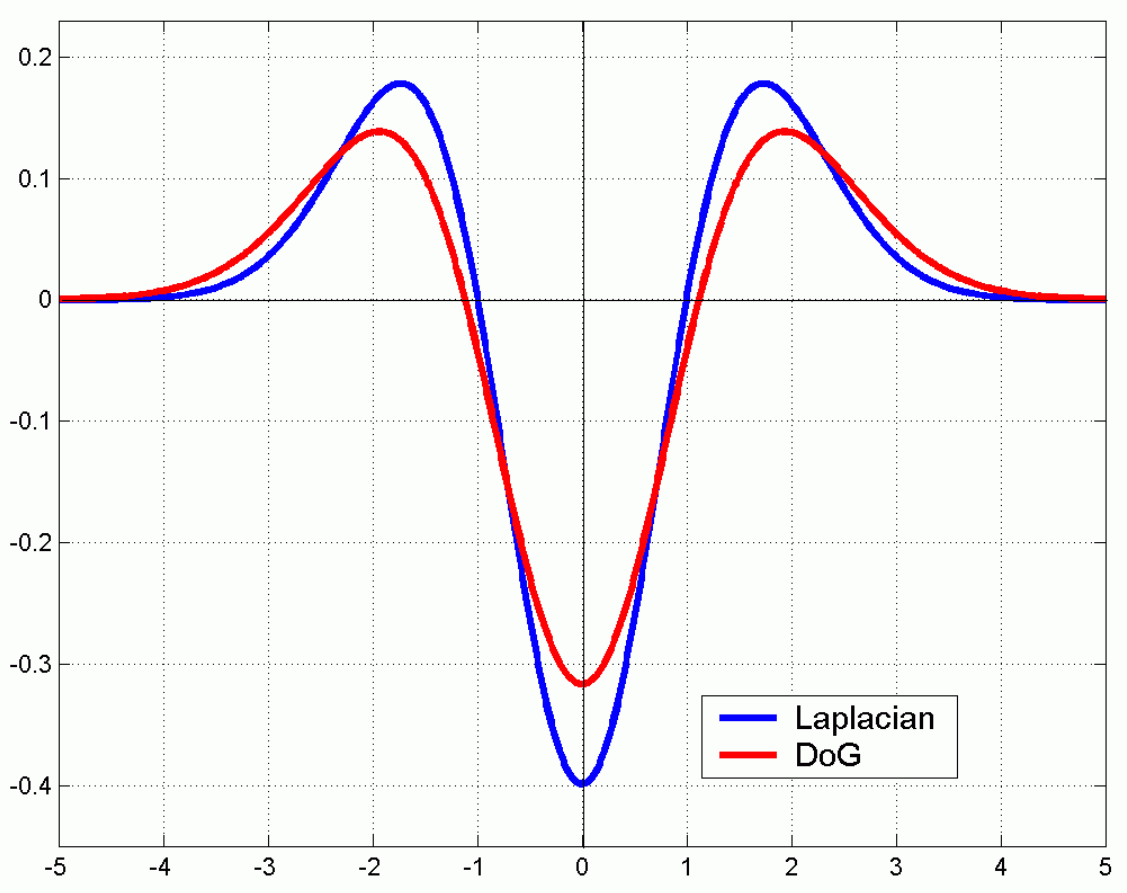
Efficient Implementation

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

(Laplacian)

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

(Difference of Gaussians)



Proof of Similarity between LOG and DOG

Supplement

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \in \text{Gaussian kernel.}$$

$$\frac{\partial G_\sigma}{\partial x} = \frac{1}{2\pi\sigma^2}\left(\frac{\partial}{\partial x} e^{-(x^2+y^2)/2\sigma^2}\right)$$

$$= \frac{1}{2\pi\sigma^2}\left(\frac{-x}{\sigma^2} e^{-(x^2+y^2)/2\sigma^2}\right)$$

$$\frac{\partial^2}{\partial x^2} G_\sigma = \frac{1}{2\pi\sigma^2}\left(\frac{x^2-\sigma^2}{\sigma^4} e^{-(x^2+y^2)/2\sigma^2}\right)$$

Similarly,

$$\frac{\partial}{\partial y^2} G_\sigma = \frac{1}{2\pi\sigma^2}\left(\frac{y^2-\sigma^2}{\sigma^4} e^{-(x^2+y^2)/2\sigma^2}\right)$$

$$\nabla^2 G_\sigma = \frac{\partial^2 G_\sigma}{\partial x^2} + \frac{\partial^2 G_\sigma}{\partial y^2} = \frac{1}{2\pi\sigma^4}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4} e^{-(x^2+y^2)/2\sigma^2}\right) \cdots ①$$

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \in \text{Gaussian kernel.}$$

$$\frac{\partial}{\partial\sigma} G_\sigma(x,y) = \frac{1}{2\pi}\left(\frac{x^2-y^2-2\sigma^2}{\sigma^5} e^{-\frac{x^2+y^2}{2\sigma^2}}\right) ②$$

According to ① and ②

$$\therefore \frac{\partial G_\sigma}{\partial\sigma} = \sigma\nabla^2 G_\sigma$$

$$\frac{\partial G_\sigma}{\partial\sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx \sigma^2\nabla^2 G_\sigma (k-1)$$

$$\therefore \nabla^2 G_\sigma \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{(k-1)\sigma^2}.$$

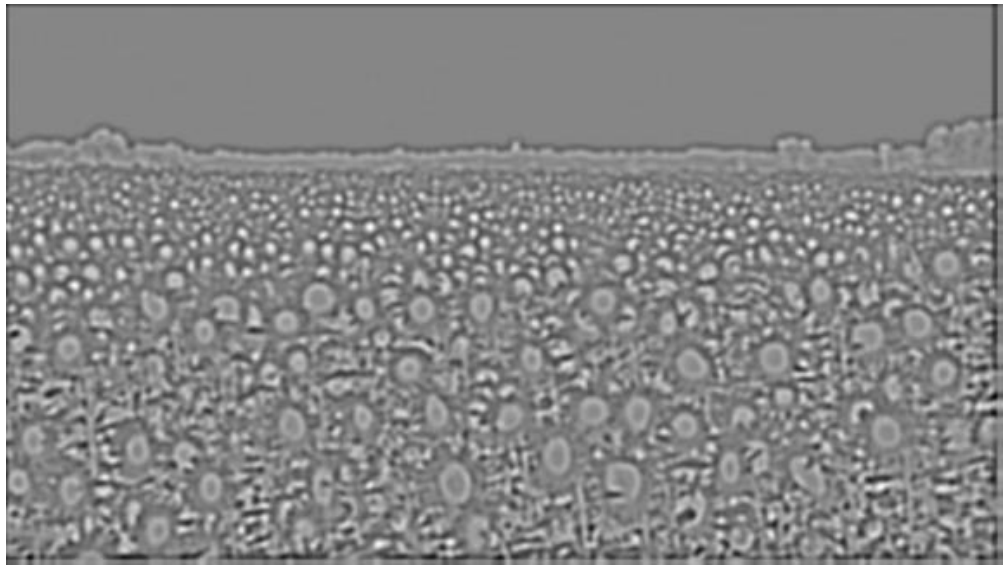
When k = 2

$$\sigma^2\nabla^2 G_\sigma \approx G(x,y,k\sigma) - G(x,y,\sigma)$$

Comparison of LoG and DoG

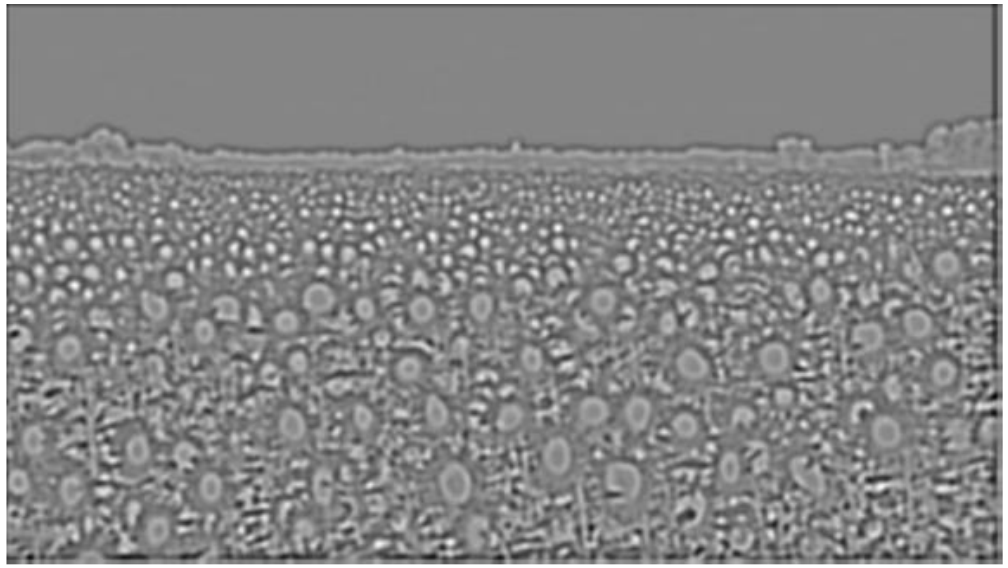
Img * G1

Img * G2



Img * LOG(G1)

k = 1.1
G1: 2
G2: 2*k



$(Img*G2-Img*G1)/((k-1)\sigma^2)$

Why Is DoG Better than LoG?

Supplement

1. Separability

Gaussian filter: $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{y^2}{2\sigma^2}}\right) = G_x \cdot G_y$$

Seperable 2D convolution : $F * (H_1 \cdot H_2) = (F * H_1) * H_2$

```
gx = fspecial('gaussian',[3 1],1);
gy = fspecial('gaussian',[1, 3],1);
Gxy = fspecial('gaussian',[3 3],1);
```

```
>> gx

gx =

    0.2741
    0.4519
    0.2741

>> gy

gy =

    0.2741    0.4519    0.2741
```

```
>> Gxy

Gxy =

    0.0751    0.1238    0.0751
    0.1238    0.2042    0.1238
    0.0751    0.1238    0.0751

>> gx*gy

ans =

    0.0751    0.1238    0.0751
    0.1238    0.2042    0.1238
    0.0751    0.1238    0.0751
```

Tutorial

Image: M X N
2D Kernel: L X L

Computation cost for 2D kernel: 2**MNL²**
Computation cost for two decomposed 1D kernels : **4MNL**

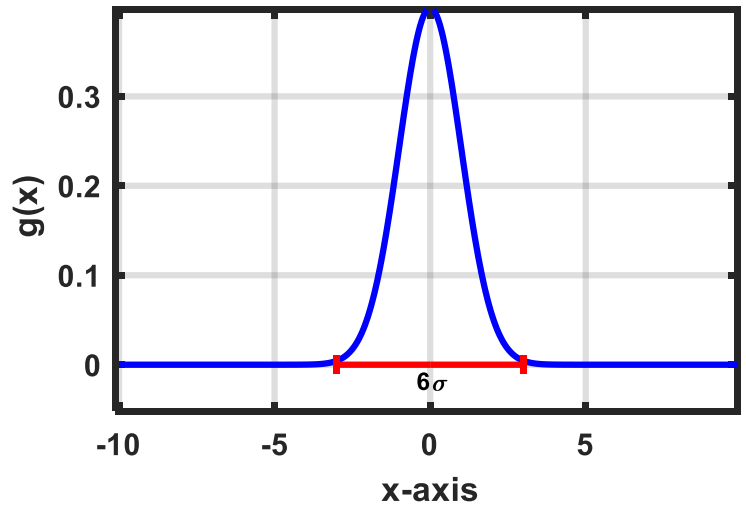
Use of two 1D kernel has much cheaper computational cost

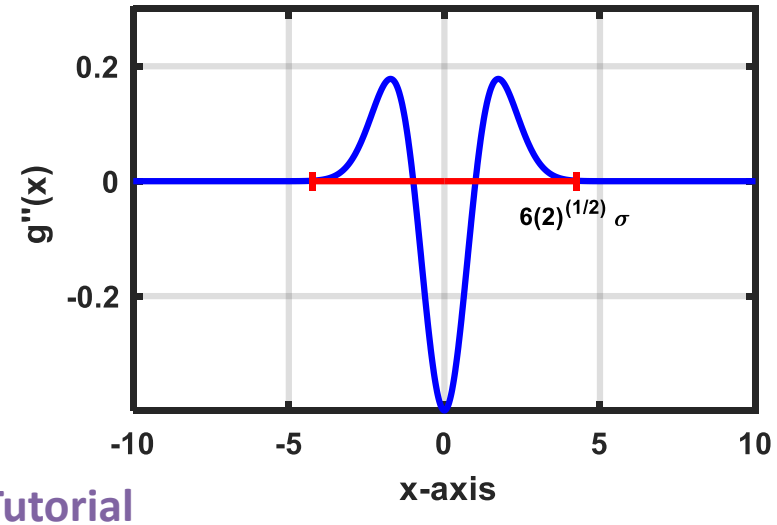
http://www.songho.ca/dsp/convolution/convolution2d_separable.html
https://dsp.stackexchange.com/questions/36962/why-does-the-separable-filter-reduce-the-cost-of-computing-the-operator

Why Is DoG Better than LoG? (Continue)

Supplement

2. 2D Filter Size





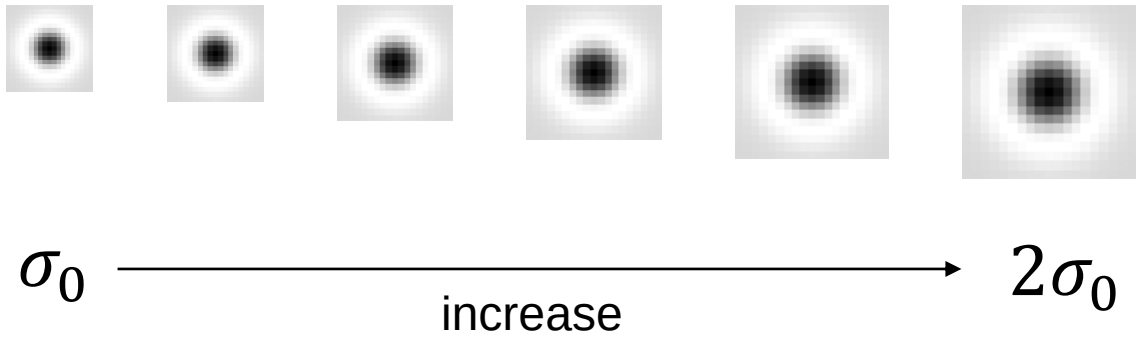

```
h1 = fspecial('gaussian', [10,10],1);
h2 = fspecial('log', [10,10],1);
```

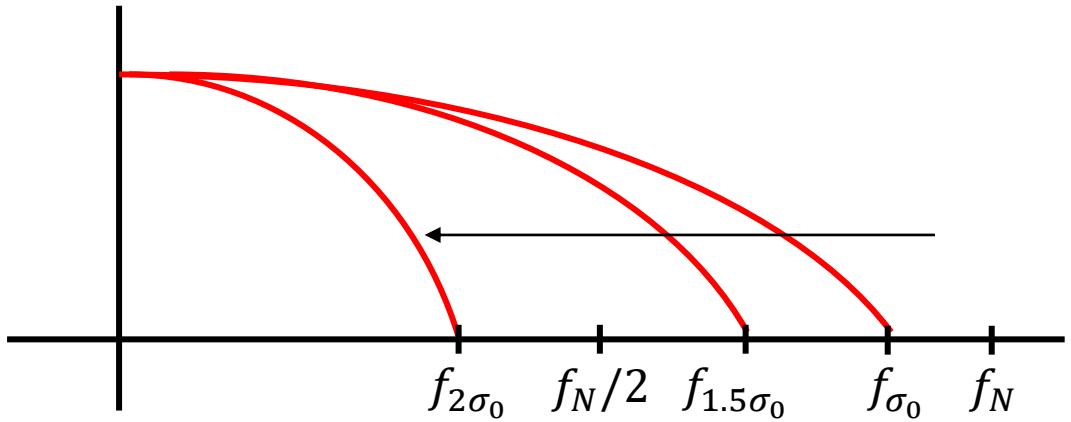
```
h1 =

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0001    0.0003    0.0003    0.0001    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0003    0.0023    0.0062    0.0062    0.0023    0.0003    0.0000    0.0000
    0.0000    0.0001    0.0023    0.0168    0.0456    0.0456    0.0168    0.0023    0.0001    0.0000
    0.0000    0.0003    0.0062    0.0456    0.1240    0.1240    0.0456    0.0062    0.0003    0.0000
    0.0000    0.0003    0.0062    0.0456    0.1240    0.1240    0.0456    0.0062    0.0003    0.0000
    0.0000    0.0001    0.0023    0.0168    0.0456    0.0456    0.0168    0.0023    0.0001    0.0000
    0.0000    0.0000    0.0003    0.0023    0.0062    0.0062    0.0023    0.0003    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0001    0.0003    0.0003    0.0001    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

```
h2 =

    0.0000    0.0000    0.0000    0.0000    0.0001    0.0001    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0003    0.0014    0.0032    0.0032    0.0014    0.0003    0.0000    0.0000
    0.0000    0.0003    0.0032    0.0148    0.0278    0.0278    0.0148    0.0032    0.0003    0.0000
    0.0000    0.0014    0.0148    0.0419    0.0228    0.0228    0.0419    0.0148    0.0014    0.0000
    0.0001    0.0032    0.0278    0.0228   -0.1859   -0.1859    0.0228    0.0278    0.0032    0.0001
    0.0001    0.0032    0.0278    0.0228   -0.1859   -0.1859    0.0228    0.0278    0.0032    0.0001
    0.0000    0.0014    0.0148    0.0419    0.0228    0.0228    0.0419    0.0148    0.0014    0.0000
    0.0000    0.0003    0.0032    0.0148    0.0278    0.0278    0.0148    0.0032    0.0003    0.0000
    0.0000    0.0000    0.0003    0.0014    0.0032    0.0032    0.0014    0.0003    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0001    0.0001    0.0000    0.0000    0.0000    0.0000
```

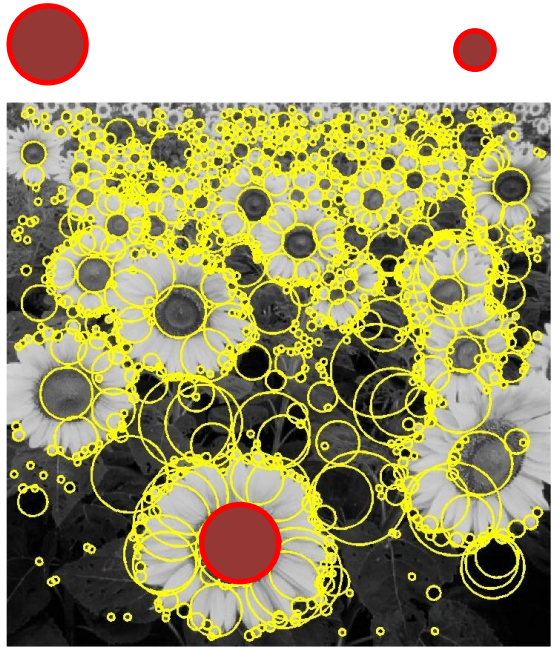
Tutorial

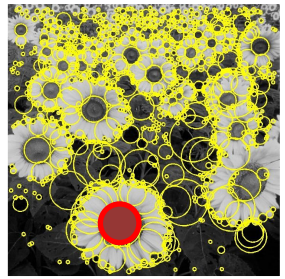
Concept of Octave





As σ increases, the size of the blob filter increases. Consequently, computational costs increases. (**MNL²** or **2MNL**). Is there any way to detect the blob without increasing the blob size?

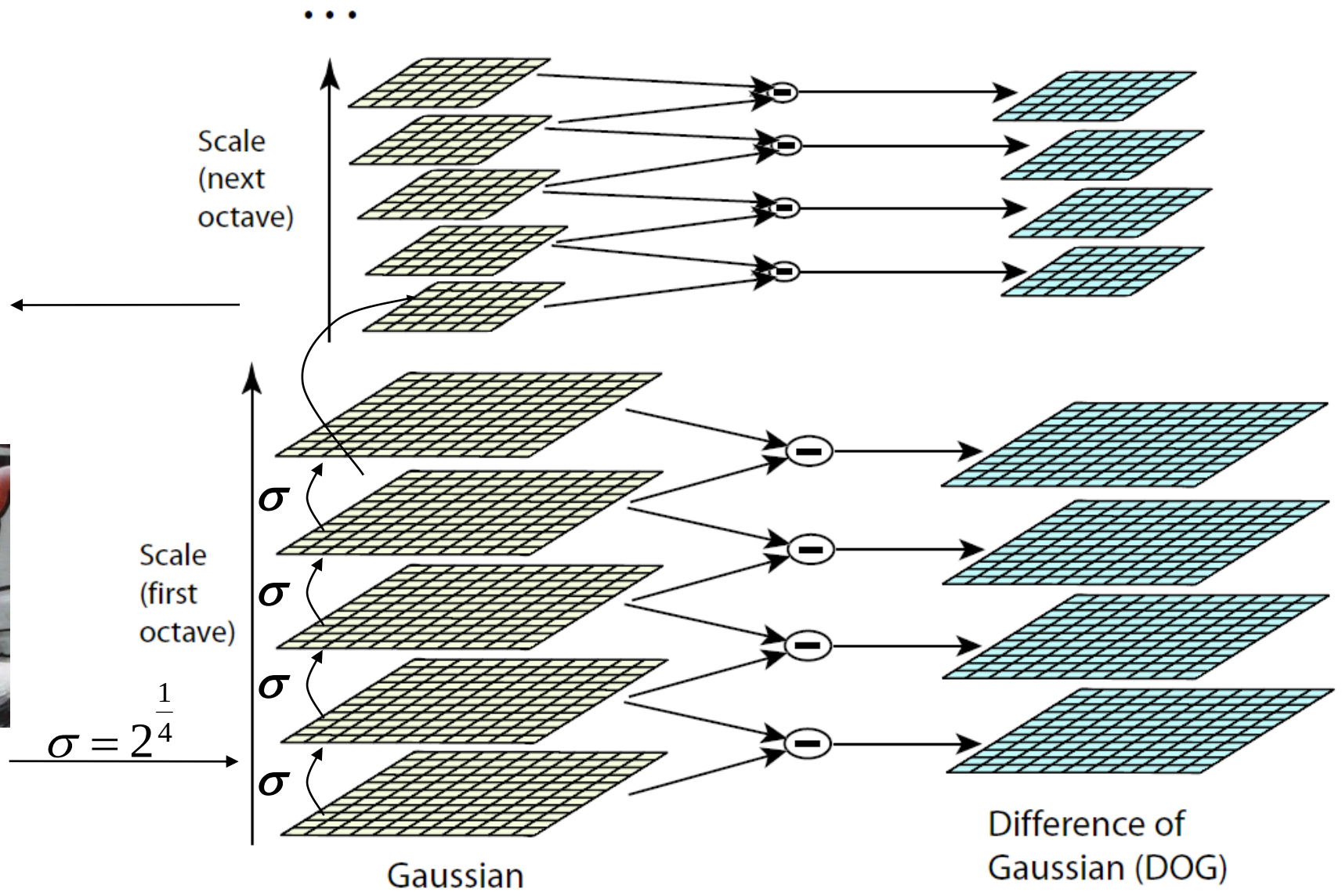




Gaussian Scale Pyramid

Sampling with step $\sigma^4 = 2$

Original image



Detection of scale-space extrema

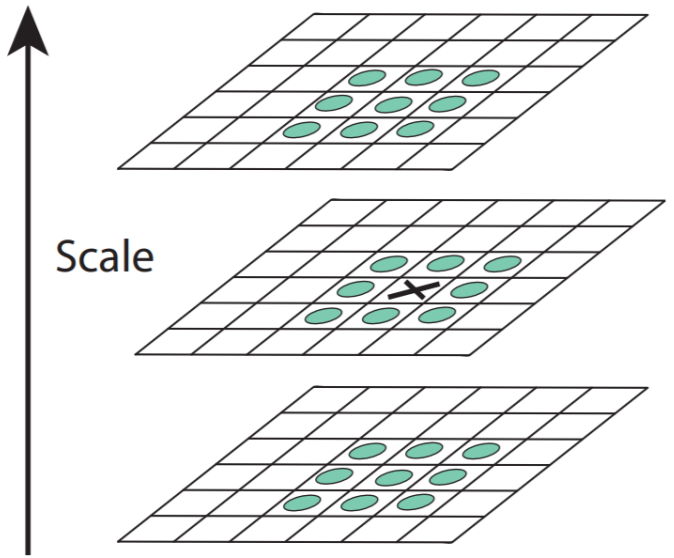
The scale space of an image is defined as a function, $L(x,y,\sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x,y,\sigma)$, with an input image, $I(x,y)$:

where * is the convolution operation in *x* and *y*, and

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y)$$

Difference-of-Gaussian function:

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



$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) = L(x,y,k\sigma) - L(x,y,\sigma)$$

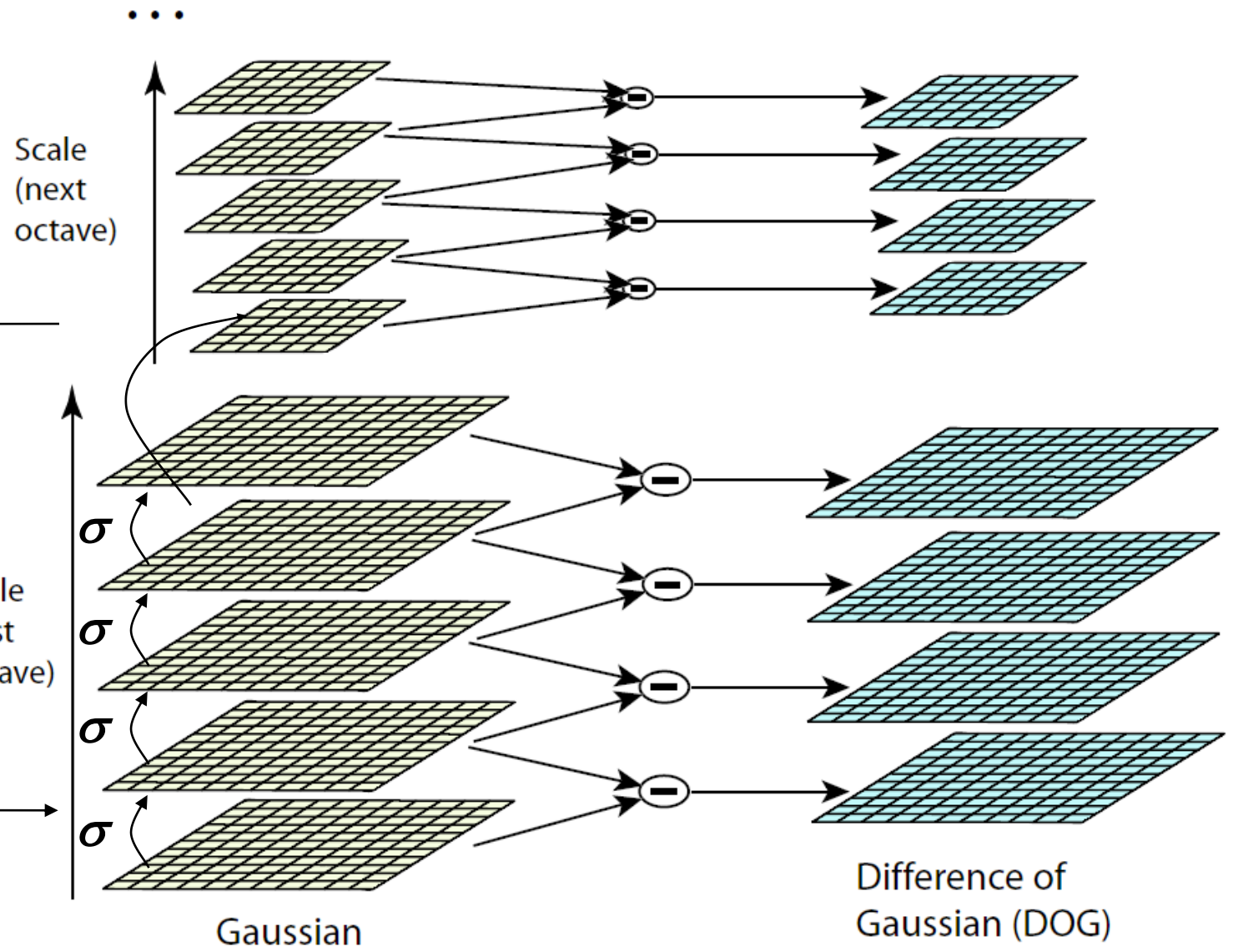
Example: Sunflower

Sampling with step σ^4 =2

Original image

$\sigma_0 = 1$

$\sigma = 2^{\frac{1}{4}}$



Example: Gaussian Smoothing (First Octave)

sigma = 1.000

sigma = 1.189

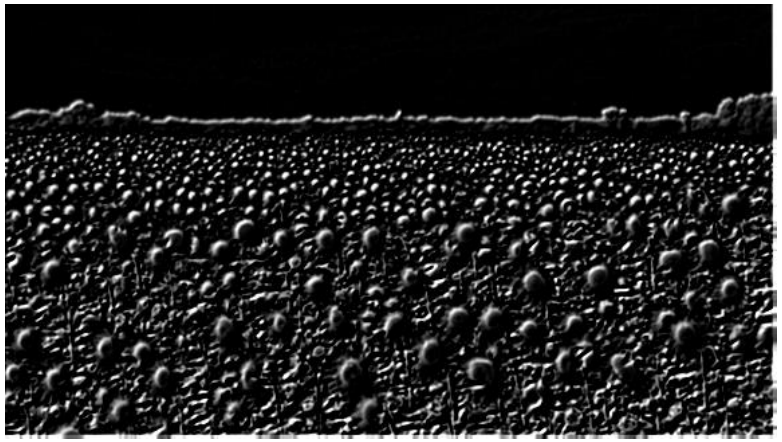
sigma = 1.414

sigma = 1.682

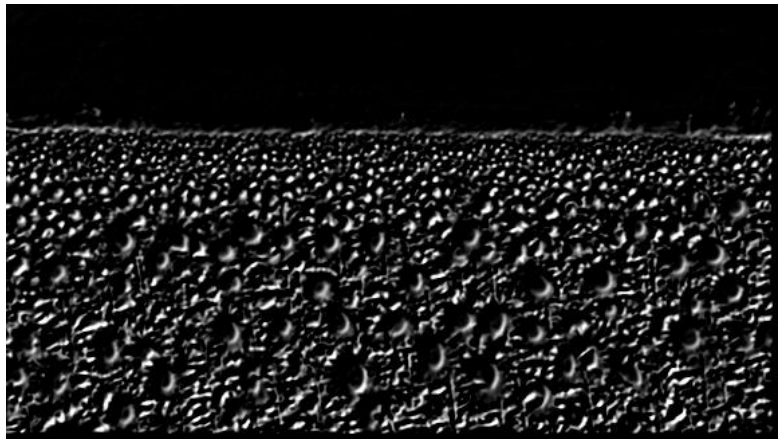
sigma = 2.000

DOG (First Octave)

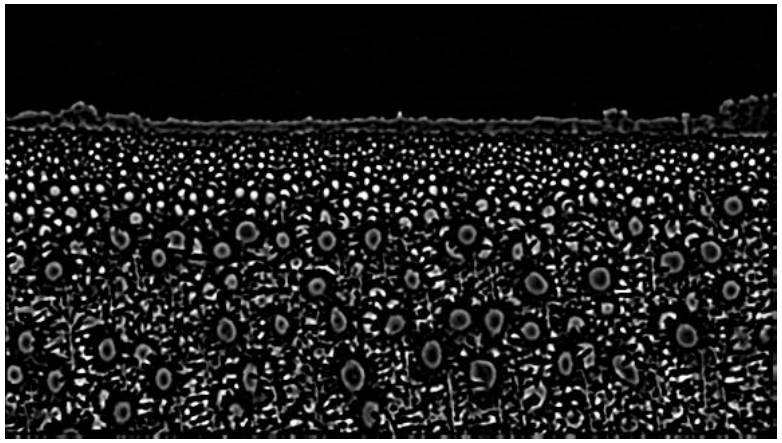
sigma = 1.000



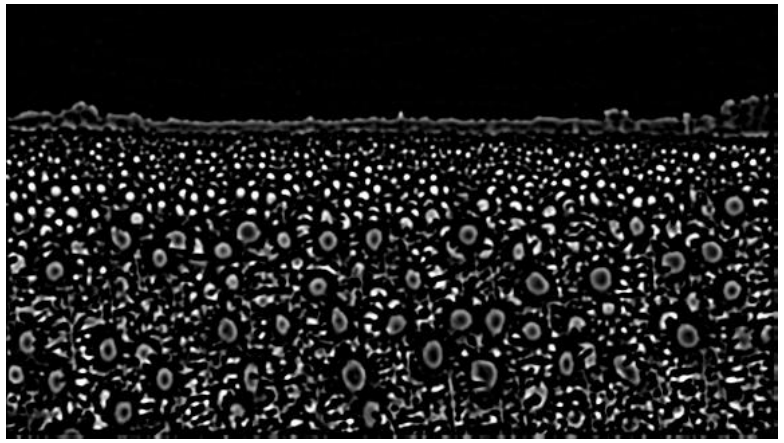
sigma = 1.189



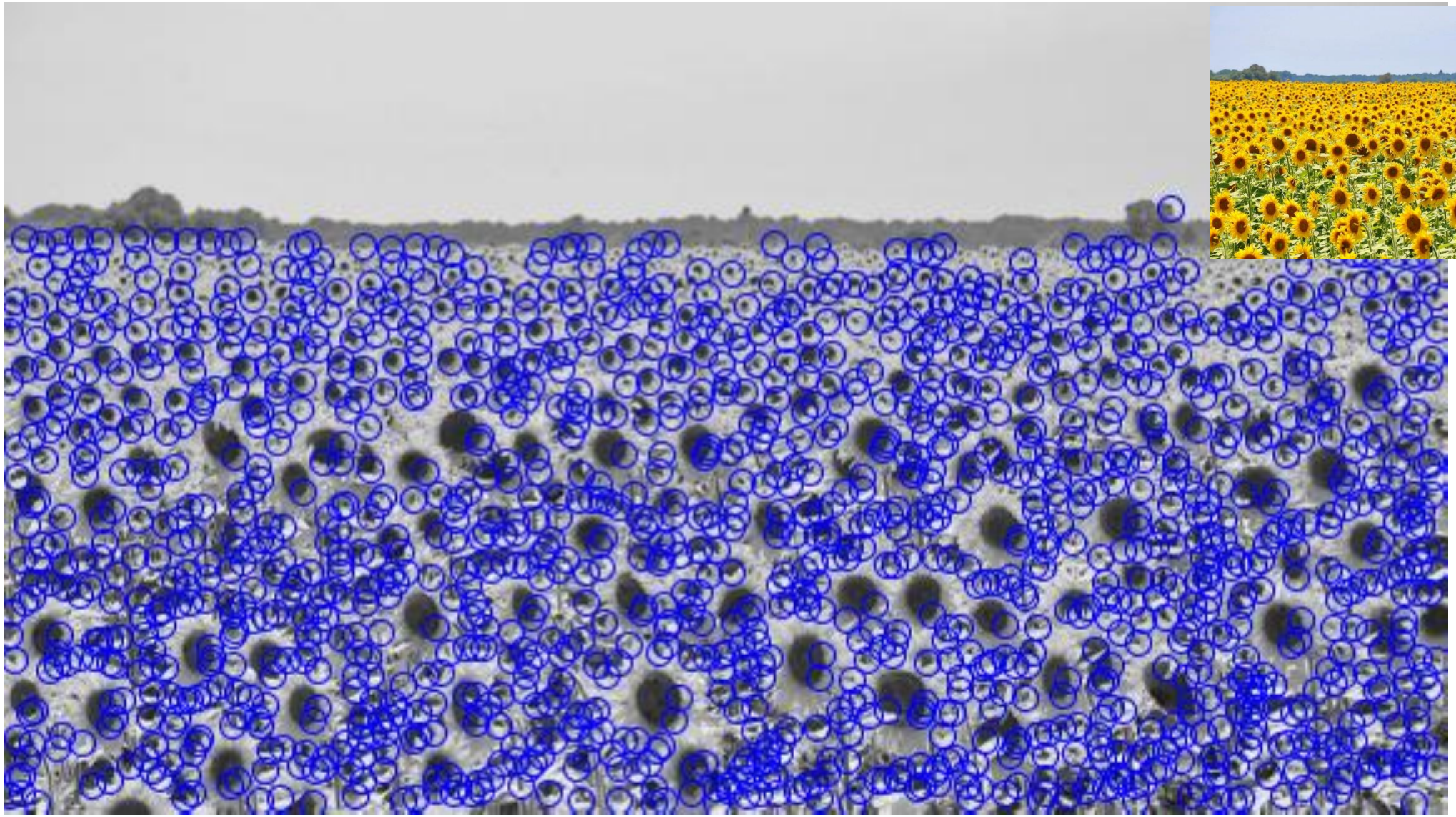
sigma = 1.414



sigma = 1.682



Keypoints (First Octave)



Contrast threshold : O , Edge threshold: X

Gaussian Smoothing (Second Octave)

sigma = 2 * 1.000

sigma = 2 * 1.189

sigma = 2 * 1.414

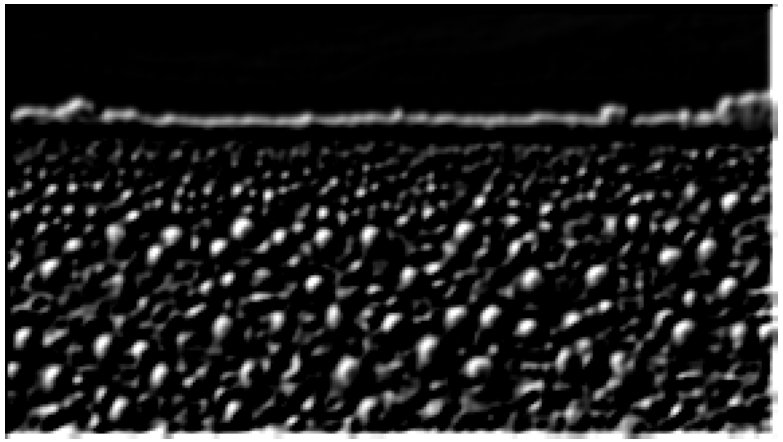
sigma = 2 * 1.682

sigma = 2 * 2.000

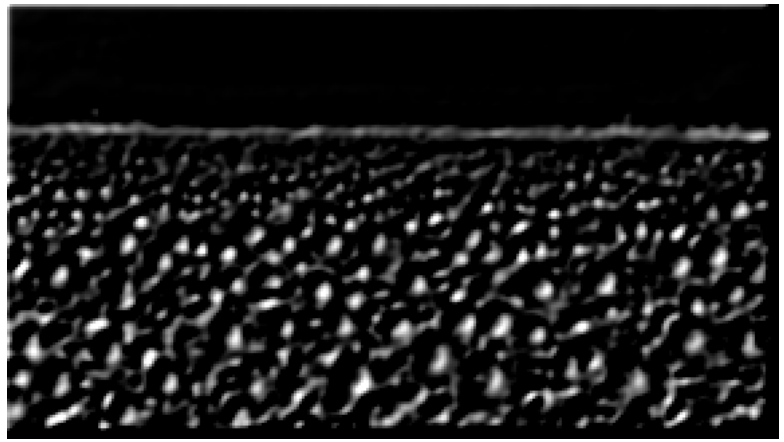
Note that images used in the second octave is the **half size of the images** used in the first octave.

DOG (Second Octave)

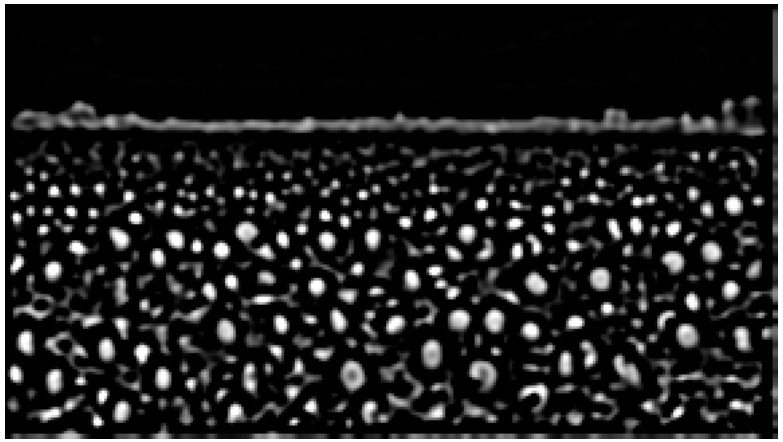
sigma = 2 * 1.000



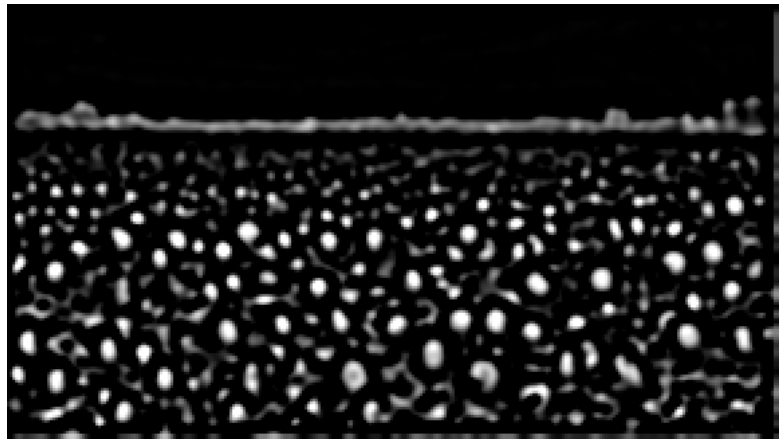
sigma = 2 * 1.189



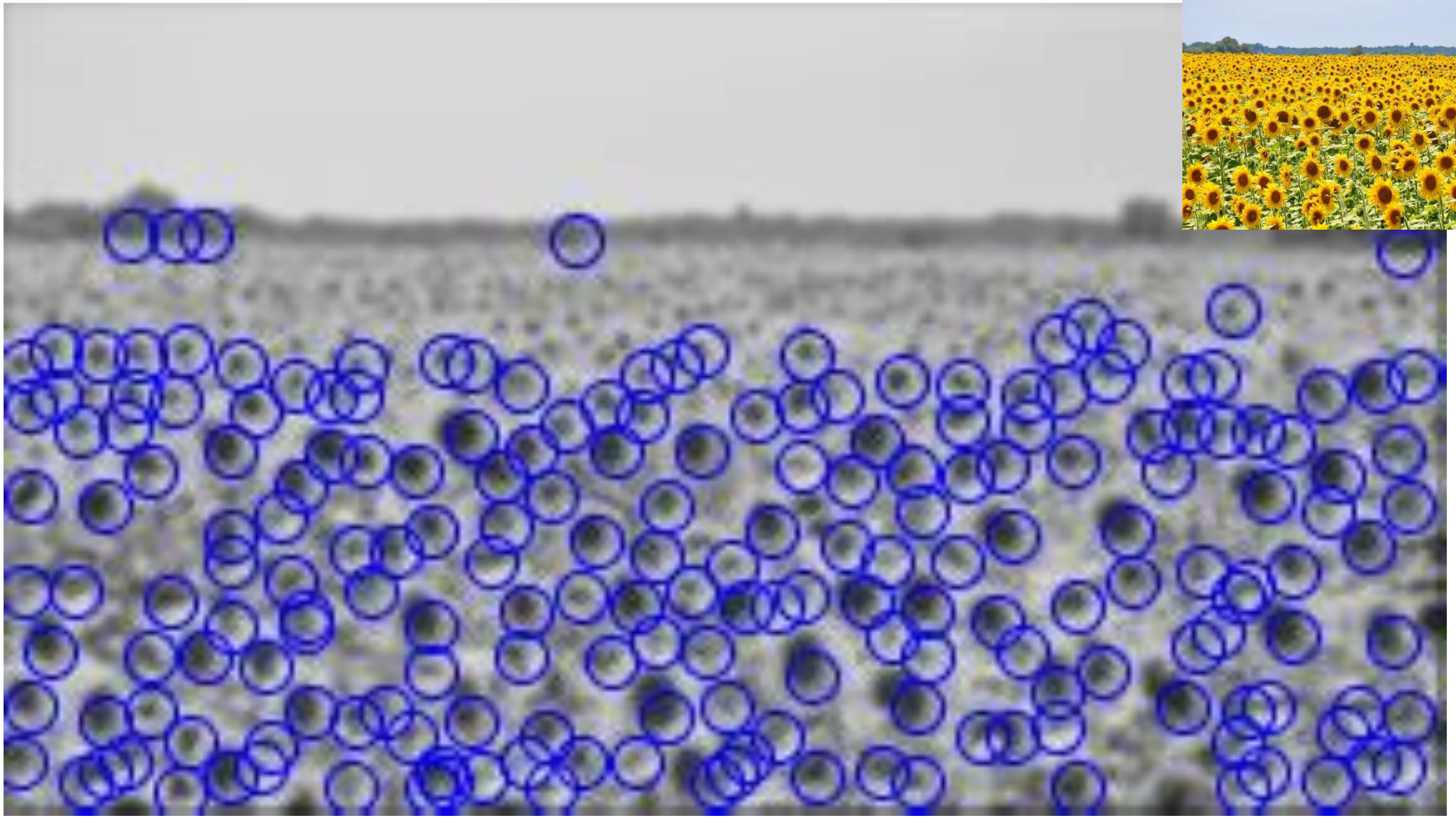
sigma = 2 * 1.414



sigma = 2 * 1.682



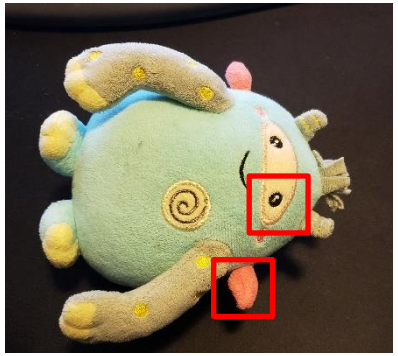
Keypoints (Second Octave)



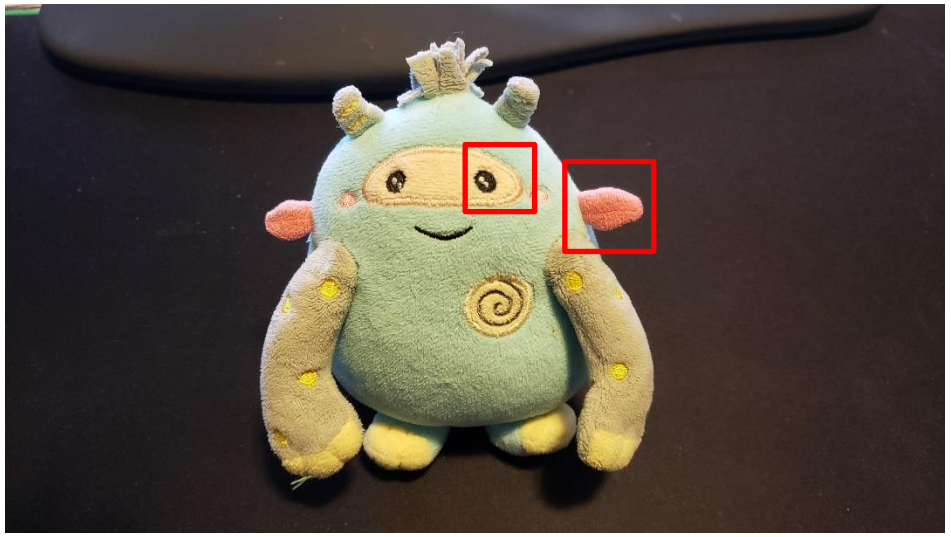
Tutorial

Orientation Assignment

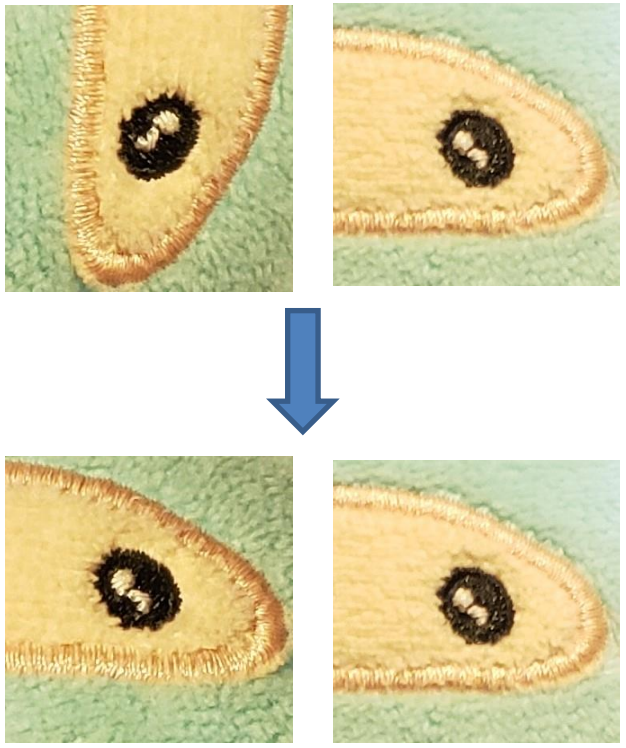
A keypoint is just a point. To match the keypoints across the image, we need to define the descriptors using the scene around each keypoints (local properties). Since each keypoints has a scale, we can define how much scene that we need to consider. However, another challenge is that the scenes are affected by image rotation. By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.



Img1



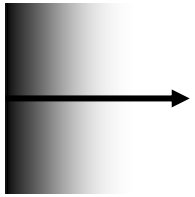
Img2



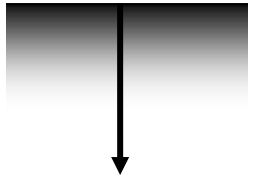
Revisit: Image Gradient

Supplement

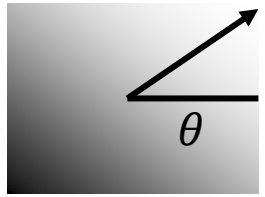
The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



$\nabla f_x = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f_y = \left[0, \frac{\partial f}{\partial x}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The gradient points in the direction of most rapid increase in intensity

The gradient direction is given by

$$\theta = tan^{-1}\left(\frac{\partial f}{\partial x} / \frac{\partial f}{\partial y}\right)$$

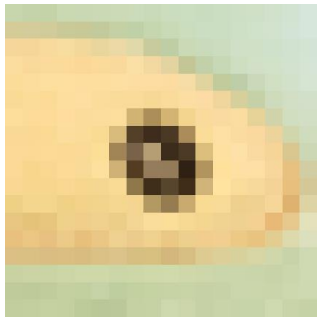
The edge strength is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

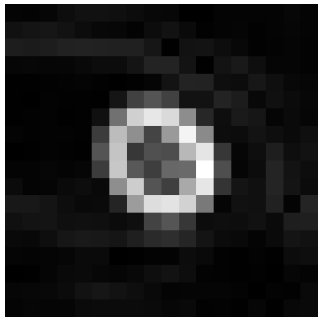
Orientation Normalization

- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation

Note that we do not rotate the actual patches for later computation. We just shift gradient angle histogram for descriptor computation.

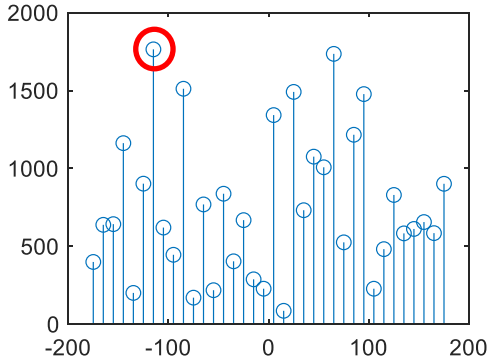


Original patch from img1



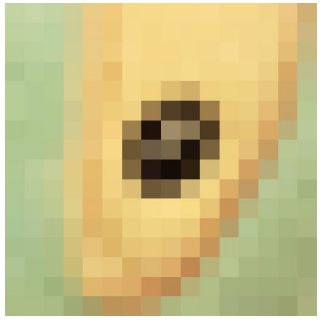
Magnitude of gradient for img1

Histogram of gradient angles for img 1 & 2

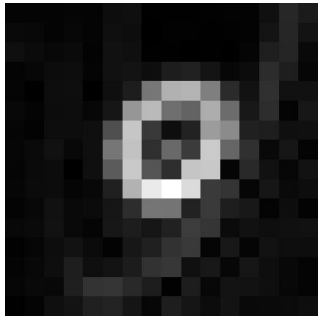


36 bins

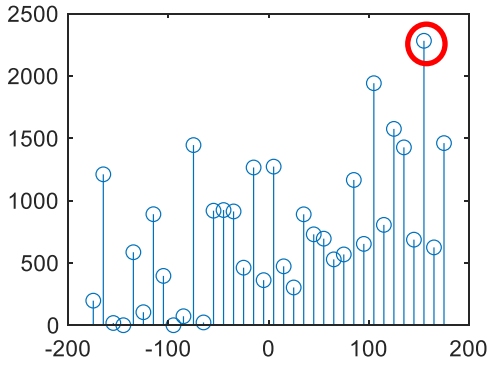
Rotate to fixed orientation



Original patch from img2



Magnitude of gradient for img2

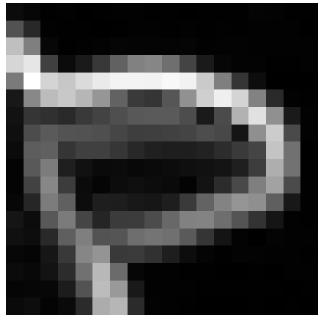


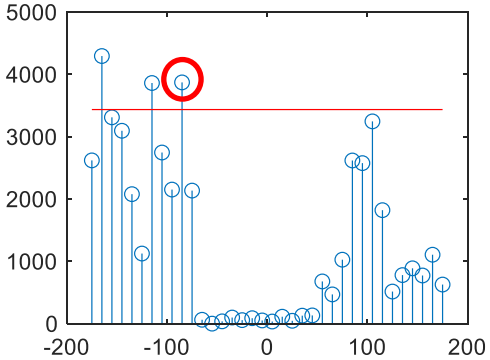
Tutorial

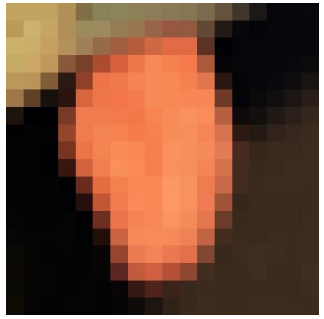
Orientation Normalization (Continue)

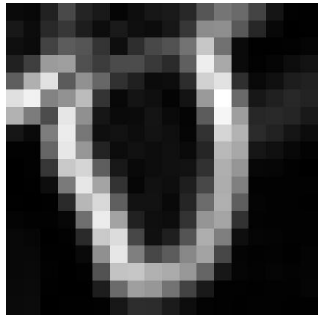
Supplement

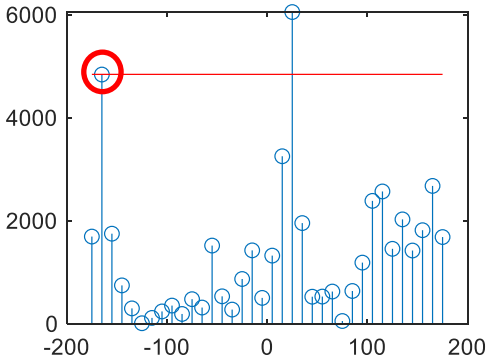
Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is used to also create a keypoint with that orientation. Therefore, for locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale but different orientations. Only about 15% of points are assigned multiple orientations, but these contribute significantly to the stability of matching. Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.









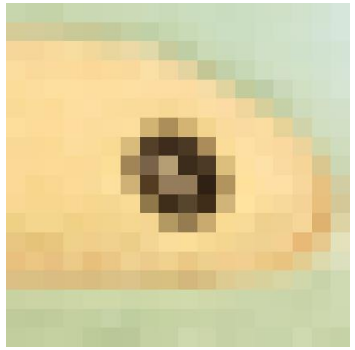


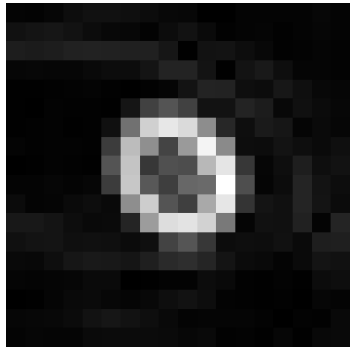
Tutorial

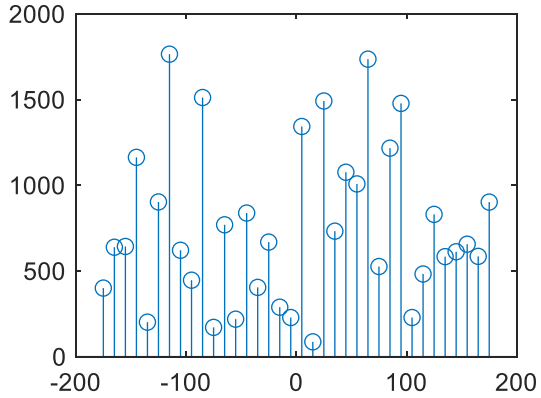
Intensity Invariant

Supplement

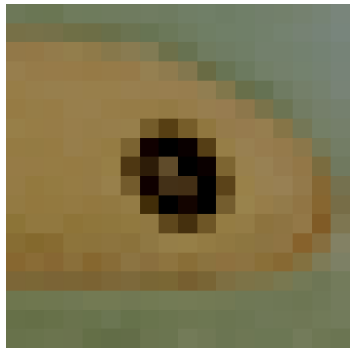
Q. Why do we use the gradient histogram?

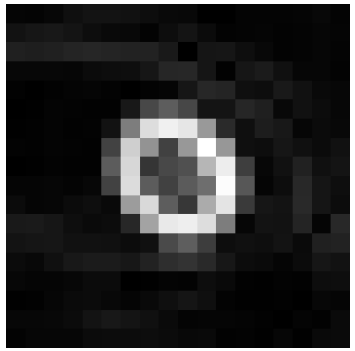


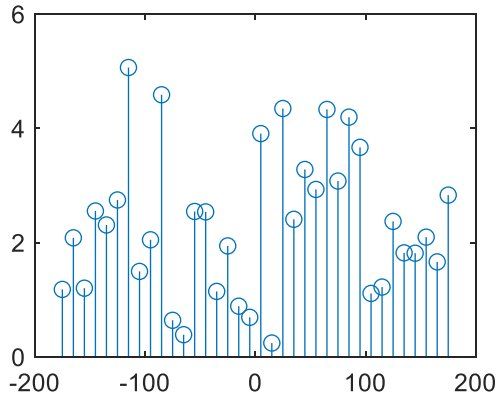


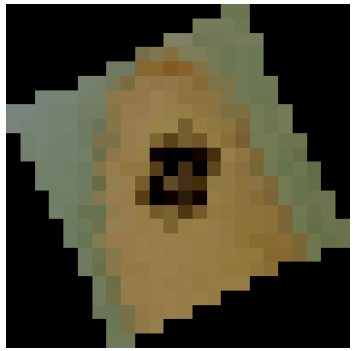


img1





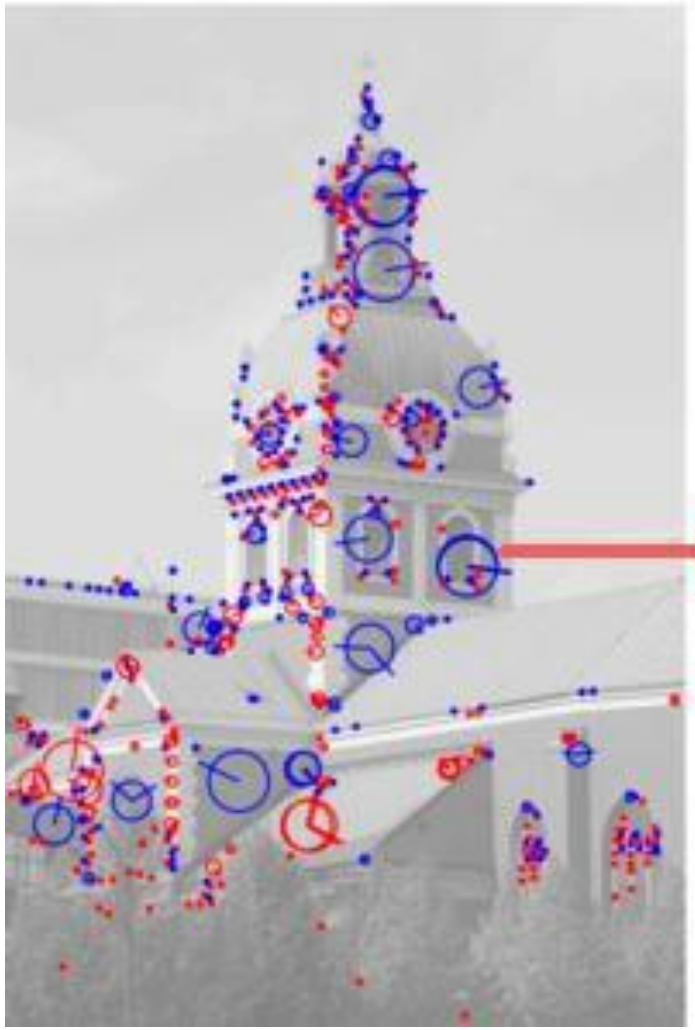




img2 = 0.8 img1 - 0.2

Tutorial

Features and Descriptors



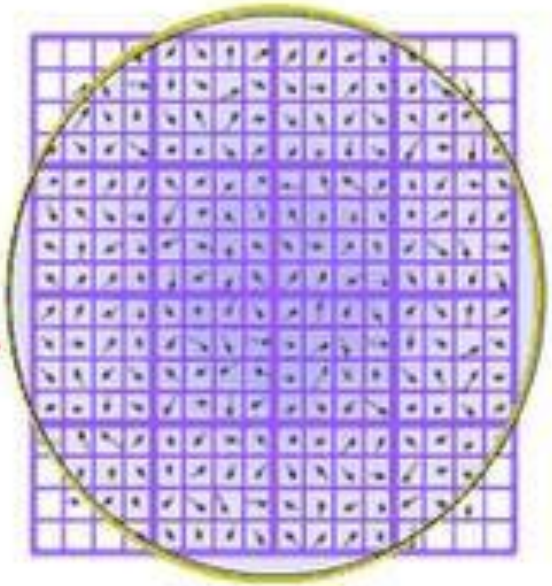
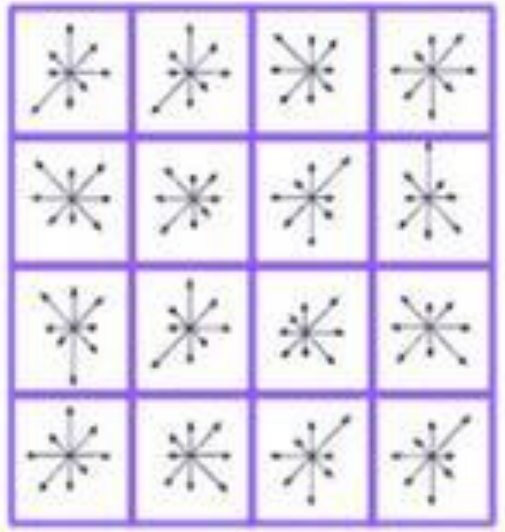


Image gradients



Keypoint descriptor

16 x 16 pixels

4 x 4 x 8 = 128

Keypoint Matching

Feature Match Process

- Each extracted feature has a 128-element descriptor vector assigned to it. Normally, it is quantized as uint8 (0~255) in each element.
- The Euclidean distances between each feature's descriptor vector in the image 1 and any of the features descriptor vectors in the image2 are computed.
- If the distance (between a feature in image 1 and its nearest feature in image 2) over the distance (between a feature in image 2 and its 2^{nd} nearest feature in image 2) is smaller than a threshold, the nearest accepted as the matching feature. Otherwise the feature in image1 does not have a match.

SIFT Library

F = VL_SIFT(I) computes the SIFT frames [1] (keypoints) F of the image I. I is a gray-scale image in single precision. Each column of F is a feature frame and has the format [X;Y;S;TH], where X,Y is the (fractional) center of the frame, S is the scale and TH is the orientation (in radians).

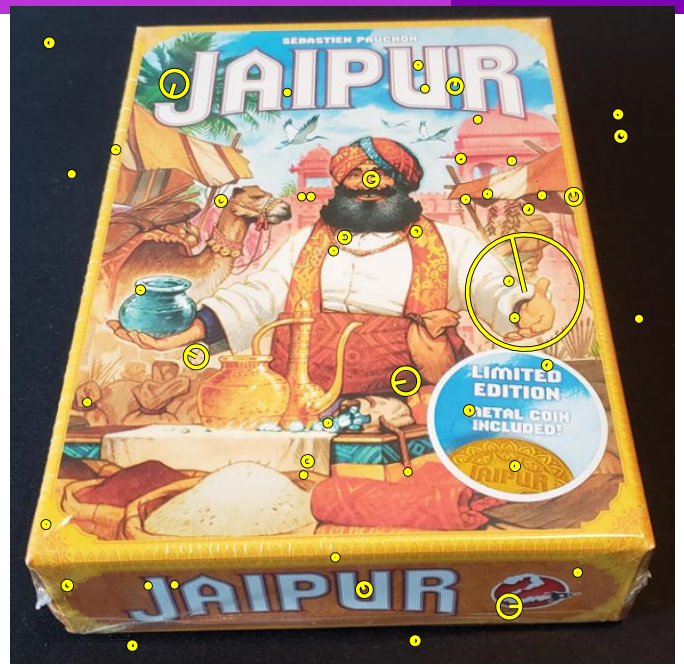
[F,D] = VL_SIFT(I) computes the SIFT descriptors [1] as well. Each column of D is the descriptor of the corresponding frame in F. A descriptor is a 128-dimensional vector of class UINT8.

VL_SIFT() accepts the following options:

http://www.vlfeat.org/overview/sift.html

VL_SIFT Demo

```
%% install vl_feat
% run('vlfeat/toolbox/vl_setup');
% vl_version verbose;

%% SIFT
clear;close all; clc;

img1o = imresize(imread('20201226_125213.jpg'), 0.3);
img2o = imresize(imread('20201226_125222.jpg'), 0.3);

img1 = single(rgb2gray(img1o));
img2 = single(rgb2gray(img2o));

figure(1);
subplot(121); imshow(img1o);
subplot(122); imshow(img2o);

[f1,d1] = vl_sift(img1);
[f2,d2] = vl_sift(img2);

figure(2);
imshow(img1o); hold on;
perm = randperm(size(f1,2)) ;
sel = perm(1:50) ;
h1 = vl_plotframe(f1(:,sel)) ;
h2 = vl_plotframe(f1(:,sel)) ;
set(h1,'color','k','linewidth',3) ;
set(h2,'color','y','linewidth',2) ; hold off;

figure(3);
imshow(img1o); hold on;
h3 = vl_plotsiftdescriptor(d1(:,sel),f1(:,sel));
set(h3,'color','g') ; hold off;

figure(4);
[matches, scores] = vl_ubcmatch(d1, d2) ;

[~, perm] = sort(scores, 'descend') ;
matches = matches(:, perm(1:30)) ;
scores  = scores(perm(1:30)) ;

figure(1) ; clf ;
```



Tutorial

Slide Credits and References

- Lecture notes: Gordon Wetzstein
- Lecture notes: Mohammad Jahanshahi
- Lecture notes: Noah Snavely
- Lecture notes: L. Fei-Fei
- Lecture notes: D. Frosyth
- Lecture notes: James Hayes
- Lecture notes: Yacov Hel-Or
- Lecture notes: K. Grauman, B. Leibe