Assignment 2: Scale-space blob detection

Project Report

Ahmad Rafi Faqiri(AF_2)

University of Utah
Department of Industrial and Civil and Environmental Engineering

1. Overview

In this project, I explored the blob detection using scale-space. The objective is to implement the method that localizes circular patches (aka blobs) in a given image. As discussed in class, the Laplacian filter is used to detect step-like patterns across the scale-space, and the final blob location is detected based on non-maxima suppression method.

2. Blob detection

Blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

1. Generate a **Laplacian of Gaussian filter**.
2. Build a Laplacian scale space, starting with some initial scale and going for n iterations:
    - Filter image with scale-normalized Laplacian at current scale.
    - Save square of Laplacian response for current level of scale space.
    - Increase scale by a factor k.
3. Perform **non-maximum suppression** in scale space.
4. Display resulting circles at their characteristic scales.

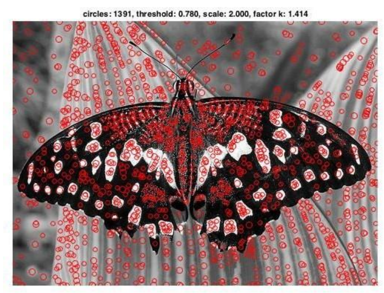
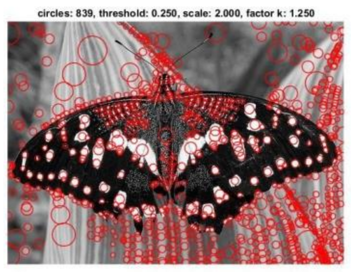
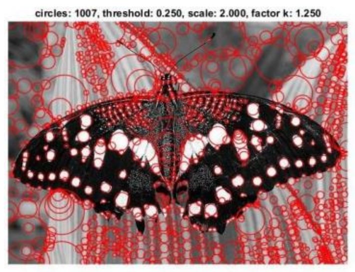
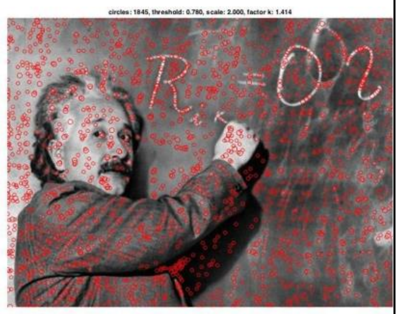
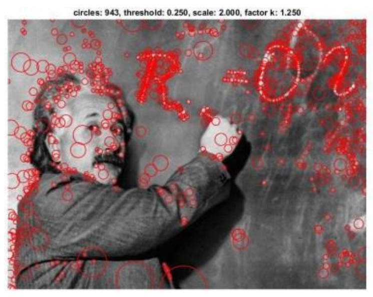
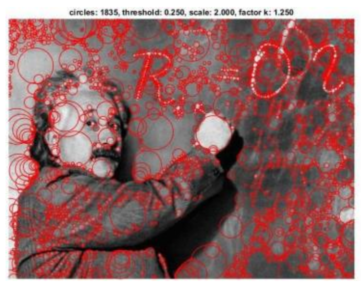
Table 1. Algorithm of blob detection

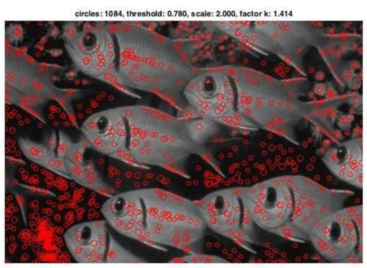
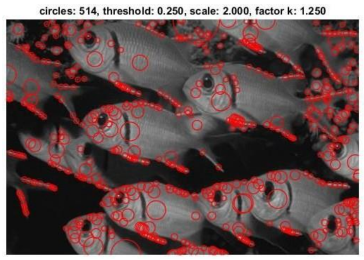
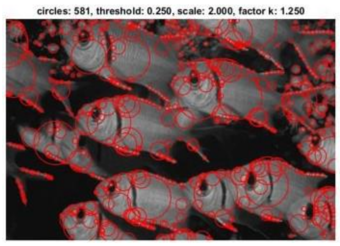
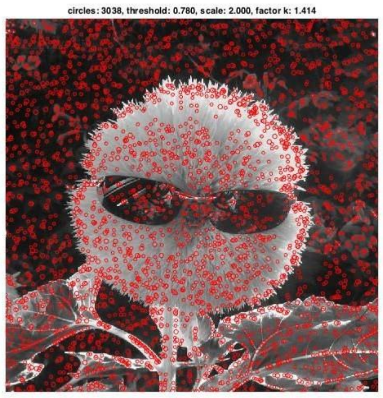
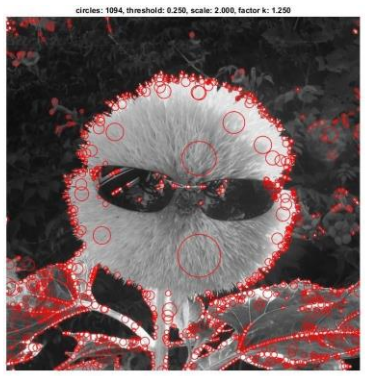
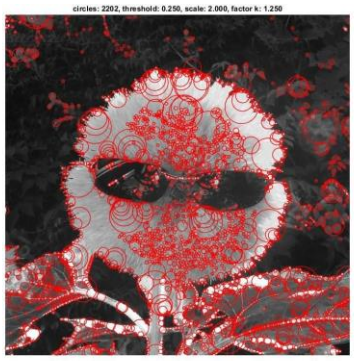
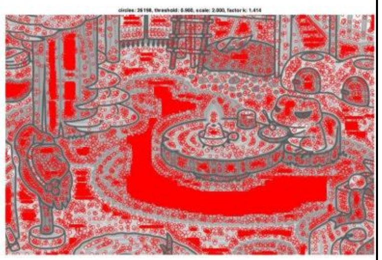
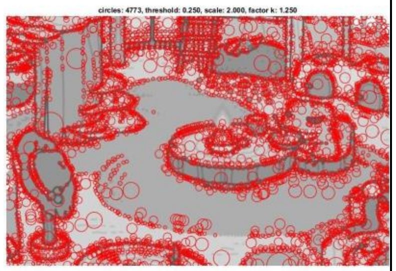
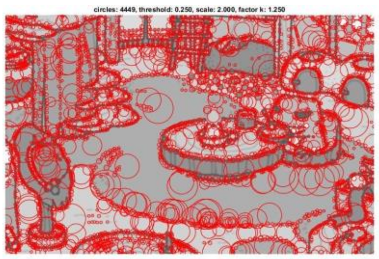
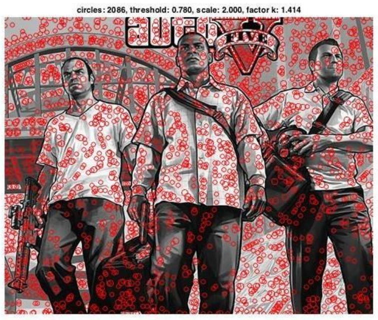
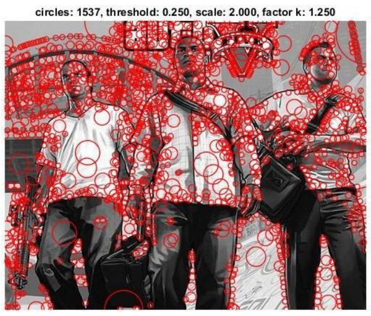
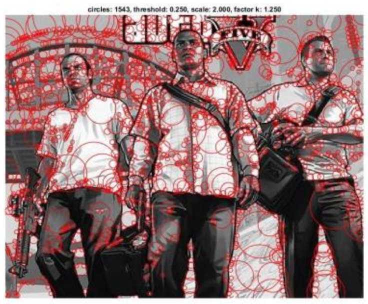
3. Output comparison

1) Parameters settings:

- k = 1.25
- scale = 2
- threshold = 0.25
- number of levels = 12

Running time is based on EWS machine. More clear output image can be found in '<u>output</u>' folder.

Difference of Gaussian (running time not saved)	**Running Time (LoG Efficient Method)**	**Running Time (LoG Inefficient Method)**
 circles: 1391, threshold: 0.780, scale: 2.000, factor k: 1.414 butterfly.jpg	 circles: 839, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.164645 seconds (on EWS Machine)	 circles: 1007, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.957813 seconds (on EWS Machine)
 circles: 1845, threshold: 0.780, scale: 2.000, factor k: 1.414 einstein.jpg	 circles: 943, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.221423 seconds (on EWS Machine)	 circles: 1835, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 1.31354 seconds (on EWS Machine)

 circles: 1084, threshold: 0.780, scale: 2.000, factor k: 1.414 fishes.jpg	 circles: 514, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.174023 seconds (on EWS Machine)	 circles: 581, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.937453 seconds (on EWS Machine)
 circles: 3038, threshold: 0.780, scale: 2.000, factor k: 1.414 flower.jpg	 circles: 1094, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.245274 seconds (on EWS Machine)	 circles: 2202, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 1.531485 seconds (on EWS Machine)
 circles: 28198, threshold: 0.900, scale: 2.000, factor k: 1.414 frog.jpg	 circles: 4773, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.236819 seconds (on EWS Machine)	 circles: 4449, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 1.72462 seconds (on EWS Machine)
 circles: 2086, threshold: 0.780, scale: 2.000, factor k: 1.414 gta5.jpg	 circles: 1537, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.182611	 circles: 1543, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.962624

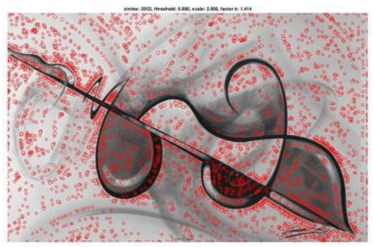
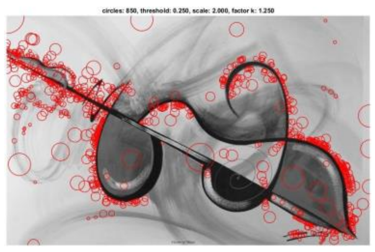
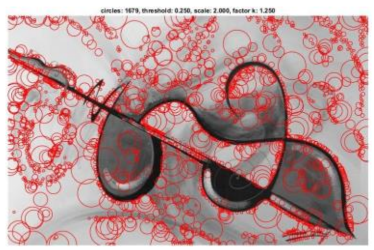
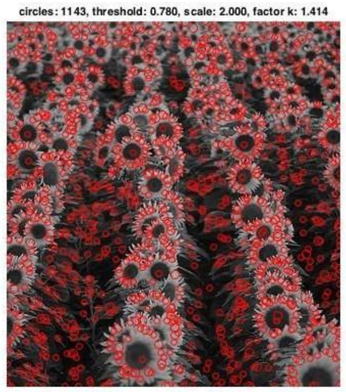
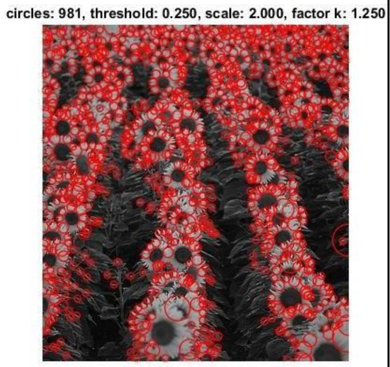
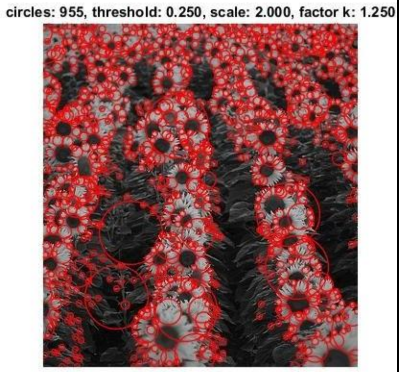
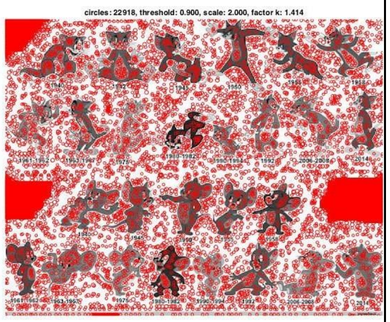
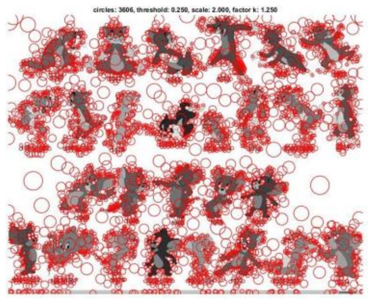
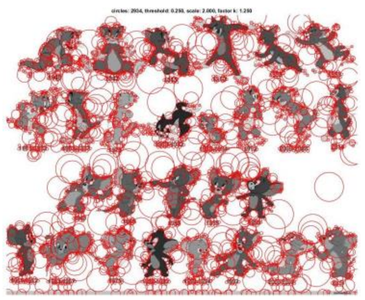
	seconds (on EWS Machine)	seconds (on EWS Machine)
 circles: 2952, threshold: 0.900, scale: 2.000, factor k: 1.414 music.jpg	 circles: 850, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.253658 seconds (on EWS Machine)	 circles: 1679, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 1.811355 seconds (on EWS Machine)
 circles: 1143, threshold: 0.780, scale: 2.000, factor k: 1.414 sunflowers.jpg	 circles: 981, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.150905 seconds (on EWS Machine)	 circles: 955, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.584322 seconds (on EWS Machine)
 circles: 22918, threshold: 0.900, scale: 2.000, factor k: 1.414 tnj.jpg	 circles: 3606, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 0.260623 seconds (on EWS Machine)	 circles: 2934, threshold: 0.250, scale: 2.000, factor k: 1.250 Elapsed time 1.913412 seconds (on EWS Machine)

Table 2. Comparison of running time of efficient/inefficient approach

4. Implementation

1) Gif animation of filter response

I used MATLAB to generate a series of Gif pictures to reveal the filter response of original image after convolving with the Laplacian of Gaussian Filter. The image is more and more ambiguous and also the edges are revealed more obvious.

This is the link to all of GIFs I generated using all of the data images:
https://github.com/Zhenye-Na/cs543/tree/master/assignments/assignment2/gif

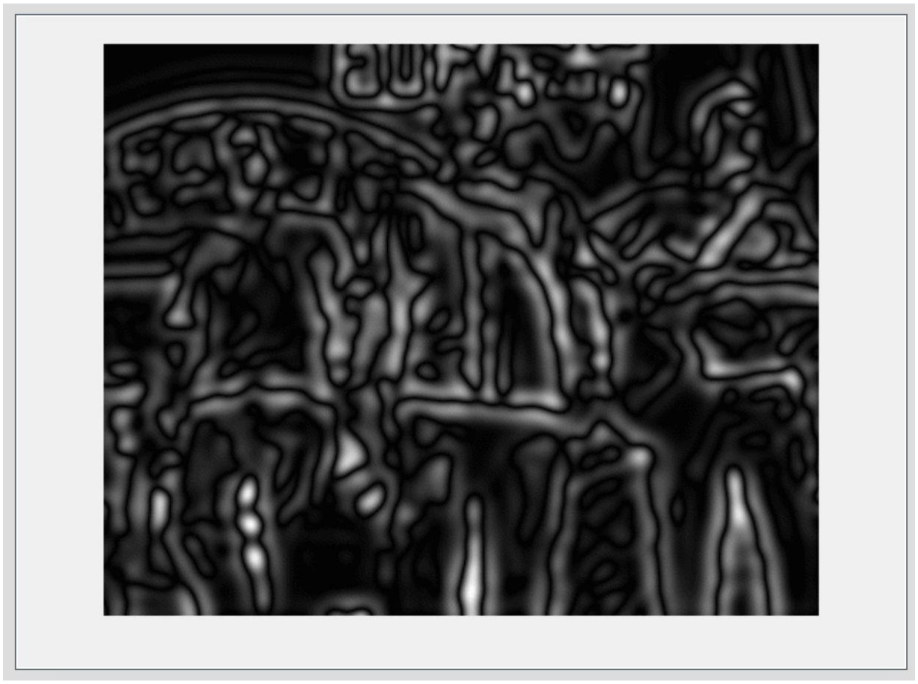


Figure 2. Filter response of gta5.jpg

5. Explanation of choosing parameters

1) Scale (sigma)

Greater sigma results in smaller blobs are missed from the image. So according to the images, I used ‘sigma = 2’ at first and according to different images and adjusted sigma respectively for different images. Because some of the images have more blobs than others. Furthermore, the size of images is another factor influenced the choice of parameters. In the extra credit part, initial sigma is sqrt(2).

2) Levels

I used 12 levels in scale space pyramid in both efficient and inefficient approaches. The number of levels can affect the final result and running time.

3) Factor k

According to instructions of this project, I used ‘k = 2’ in upscaling and downsampling approaches. In the extra credit part 1, according to the paper I used ‘k = sqrt(2)’ to initialize.

4) Threshold

Threshold I settled for all the images is 0.25. This parameter varies from images to images. There are more blobs in some of the images and other images has fewer blobs which visually distinguished. Furthermore, the size of images vary so according to these factors I selected the parameter which is the ‘best’ for images separately.

5) Optimize parameters

Grid search

6) Optimal

- k = 1.25
- scale = 2
- threshold = 0.25~0.35
- number of levels = 12~13

6. Discussion

1. Downsampling and upscaling the image instead of applying filters of increasing kernel size does improve the speed of the program.

2. The reason for the speed up is as the filter kernel size remains constant and the image size decreases the number of computations decrease, increasing the speed, while in the case of

increasing the scale of the filter coupled with the increase in the kernel size only increases the computations.

3. In my opinion it should be odd. When the width is odd, the pixel on which the filter is applied will align directly with the center of the filter mask. In case of even width there will not be a center in the mask perse and thus any pixel on the in filter mask might have to be chosen as center pixel.

4. The interpolation method used was the default ‘bicubic’ interpolation. Interpolation methods bicubic, cubic, and lanczos3 showed indistinguishable results. Hence I used the default bicubic interpolation. However, Bicubic interpolation can produce pixel values outside the original range.

5. Folder structure is as follows:

```
├── data
├── gif
├── output
│   ├── dog
│   ├── downsampling
│   └── upscaling
├── sample_output
└── src
    ├── DoG.m
    ├── Gaussian_filter.m
    ├── affine_transf.m
    ├── downsampling.m
    ├── edge_rm.m
    ├── edge_rm.m~
    ├── harris.m
    ├── show_all_circles.m
    ├── show_ellipse_circles.m
    ├── show_filter.m
    └── upscaling.m
```

This project demonstrates Laplacian of Gaussian filter which using corresponding algorithms can help us detect features at multiple scales. The scale-space (pyramid) representation is another powerful tool for segmentation and feature detection. The efficient approach which down-sample the images is much more efficient than up-size the filter size. The result of these two approaches is kind of different.

7. Extra Credit

1) Differences of Gaussian pyramid

More details can be found in '<u>DoG.m</u>'

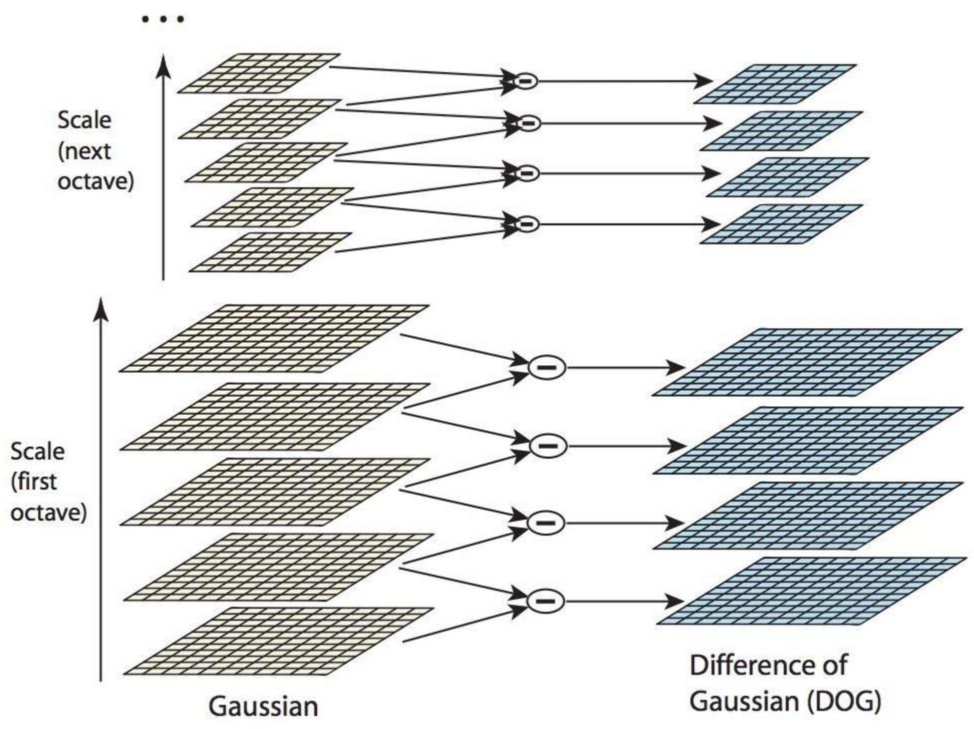


Figure 1: For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

Difference of Gaussian is much more efficient that Laplacian of Gaussian when we used to detect blobs and is kind of approximation of LoG. However, we have a distinct framework for this approach.

First, we have a new concept 'octave'. We use series of Gaussian filters to convolve with the image and subtract the adjacent ones. Then we resize the image to 'half-half' of the original size and use the median sigma of the previous octave as the first sigma in the next octave.

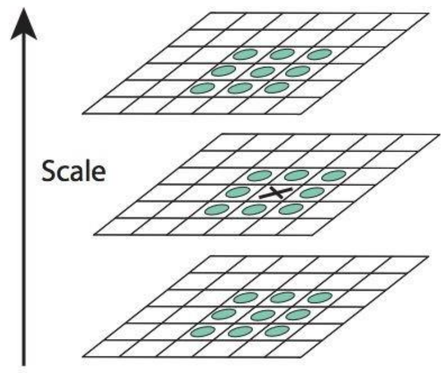


Figure 2: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

Non-maxima suppression is also a little different from the normal approach. This method need compare the specific pixel point with its 9+9+8 neighbors. Also set a threshold for suppression.

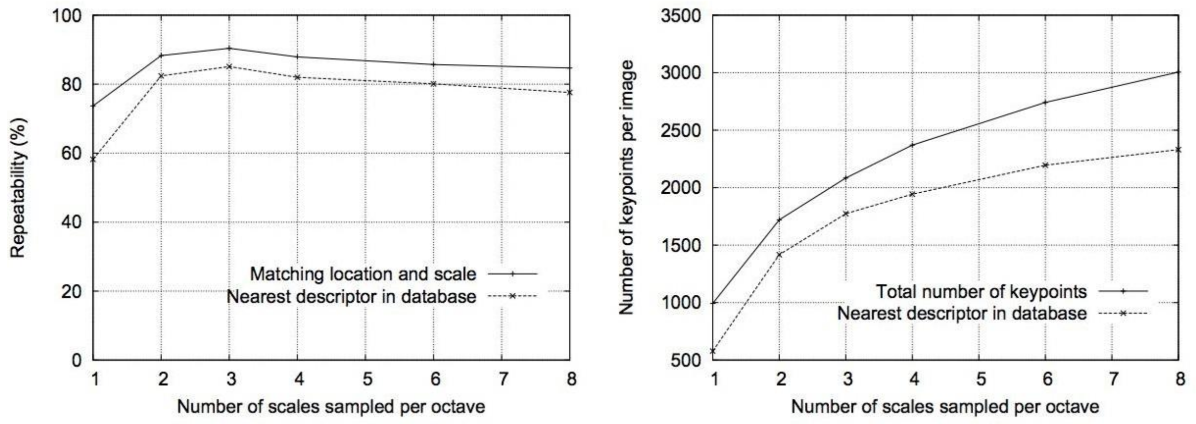


Figure 3: The top line of the first graph shows the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The lower line shows the percent of keypoints that have their descriptors correctly matched to a large database. The second graph shows the total number of keypoints detected in a typical image as a function of the number of scale samples.

According to the paper, I set the number of scales to 3 (peak in Figure 3 left side) and 3 octaves.

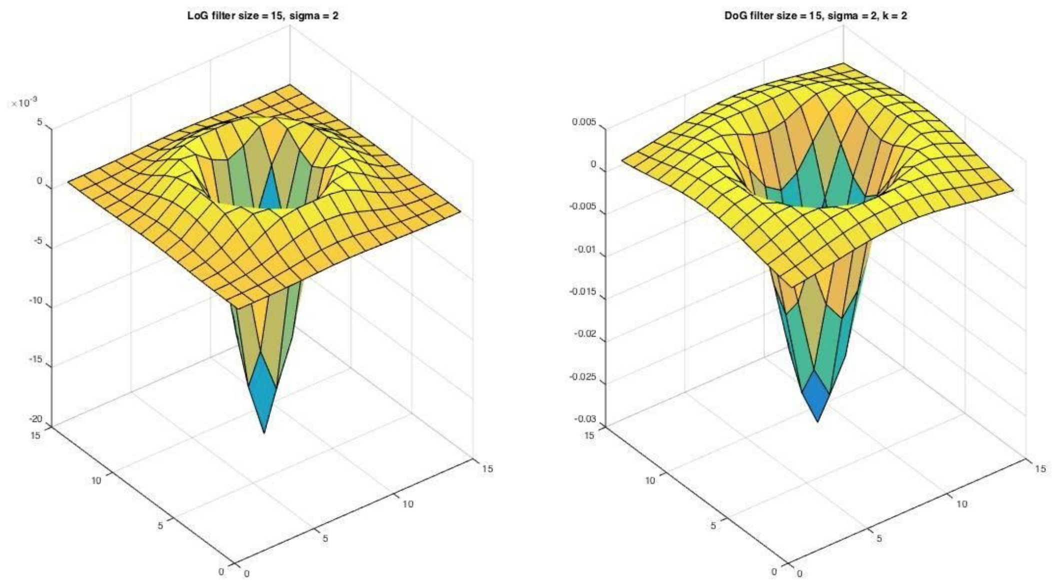


Figure 3. Comparison of DoG with LoG

2) Affine Transformation

According to the slides, we need find the relative shape of the second moment ellipse, but not the absolute scale (i.e., the axis lengths are defined up to some arbitrary constant multiplier). I computed the second moment matrix of each pixels and compute the eigenvalue of the second moment matrix. However, the plotting of the ellipse is not so correct. Furthermore, the algorithm need many 'for' loops. It need more specific algorithm to compute it.

More details in 'affine_transf.m'

3) Edge remove

Laplacian of Gaussian filter has a strong response along edges, we can observe this in the output file of LoG. So I implemented an additional thresholding step which computes the Harris response at each scale space of the filter response of LoG and rejects the regions that have only one dominant gradient orientation (i.e., regions along edges by using '$R < 0$').

More details in 'edge_rm.m'