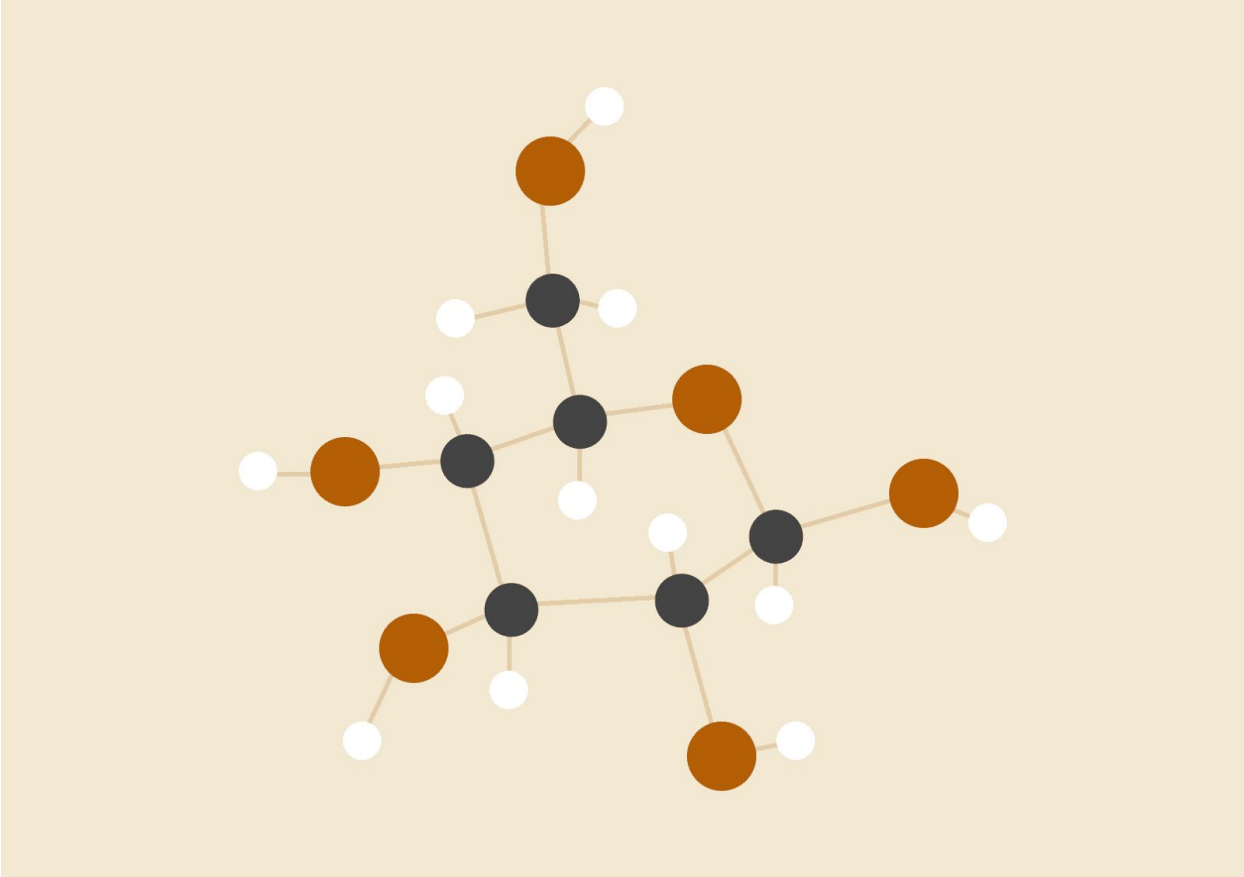
HOMEWORK 2 REPORT

Scale-space blob detection

Computer Vision and Image Processing CSE 573



Jayant Solanki

UBIT Name: jayantso
Person Number: 50246821
Fall 2017 CSE

INTRODUCTION

This report talks about the results of the application of the scale and rotation invariant "**Laplacian of Gaussian**" Filter in finding the blobs/areas of special interests in a given image, which plays an important role in object detection and tracking. This report will provide a detailed steps followed in achieving the above mentioned objective. Most of the procedures followed have been inspired from the **"Feature Detection with Automatic Scale Selection"** paper by Tony Lindeberg. Directions given in the **hw2.pdf** have been applied on the 4 test images and the results have been briefly discussed in the following sections. All the coding has been done in the MATLAB environment and proper references have been highlighted in the report.

IMPLEMENTATION

Section 1.0: A scale normalised LOG kernel has been generated using ***fspecial*** function with proper scale-space based upon the 'k' parameter whose sigma is increased k times in each iteration. In order to prevent the attenuation of the pixel response to LOG filter, kernel matrix is multiplied with the **variance (sigma^2)**. Squared value of the Filter response of the image is then preprocessed with non-maximum suppression technique to remove pixel neighbourhood with lesser value of filter response. **One thing to note is that, we can also use the *abs* function on the imfilter instead of the square function.** Initially sigma is 2 and value of k is 2^0.35 which can be adjusted for controlling the rate of growth of sigma.

Section 2.0: Above process in section 1.0 is repeated for the increasing value of the sigma and stored in a **scale-space** array whose size depends upon the number of iterations required.

Section 3.0: Since we are only concerned with the maximum response of a pixel to LOG filter, hence non-maximum suppression is done to set the pixels' values of a maxima pixel to zero whose values is less than the maxima pixel. Non-maximum suppression is done in two steps.

1. Non-maximum suppression at 2-D slices; which in turn is achieved using ***ordfilt2***

function. A matrix mask of certain dimension (must be equal to or greater than 3x3) is used with this function and non-maxima pixels are set to 0 on every scale of the scale-space array.

2. Non-maximum suppression at 3-D slices; here the characteristic scale is determined where the pixels shows maximum values across all the scales. Rest of the pixels at this positions are then turned to zero. I have used two techniques to achieve this in my code.
    a. I have looked for pixels values in every scale.
    b. Based upon David G. Lowe's paper **"Distinctive Image Features from Scale-Invariant Keypoints"**, I have looked for maximum pixel values in neighbourhood scales, for example, i-1, i and i+1 scales. This method produced better results which I will illustrate in the later sections.

Section 4.0: Finally in this part from scale-space array I extracted coordinates of those only those pixels which have their values above a particular threshold which can be decided by the user. Magnitude of the threshold lies between and 0 and 1. Higher the magnitude lesser the number of circles/blobs drawn. One thing to note is that at characteristic scale/sigma, radius of the circle is found to be $\sqrt{2}$ **times sigma, (** $\sqrt{2} * \sigma$ **)** as illustrated in the **Figure 4.1**

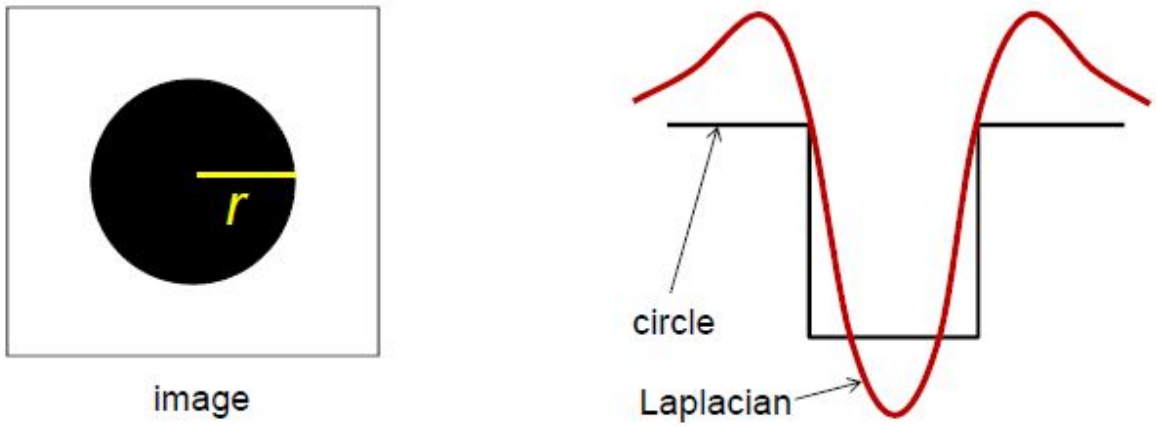


Figure 4.1 (*Courtesy Lecture-11 slides, CSE 573***)**

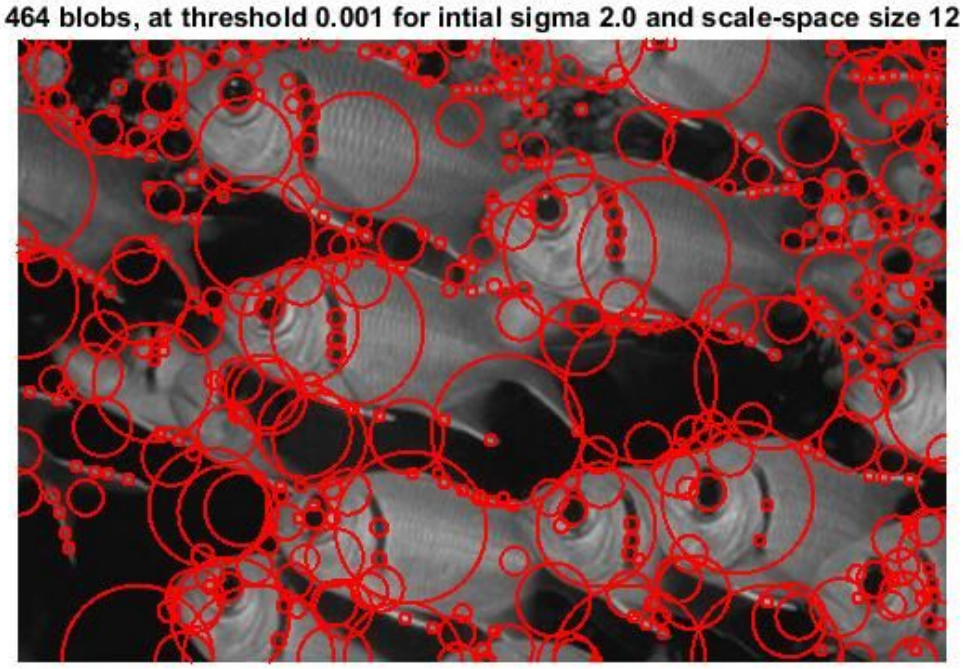
Henceforth the radius is different for every scale/sigma, bigger radius for large sigma, hence capturing the large area of interest in the image. In last I converted those pixels' coordinates into the circles with their characteristic radii and plotted them on the grayscale test image.

RESULTS

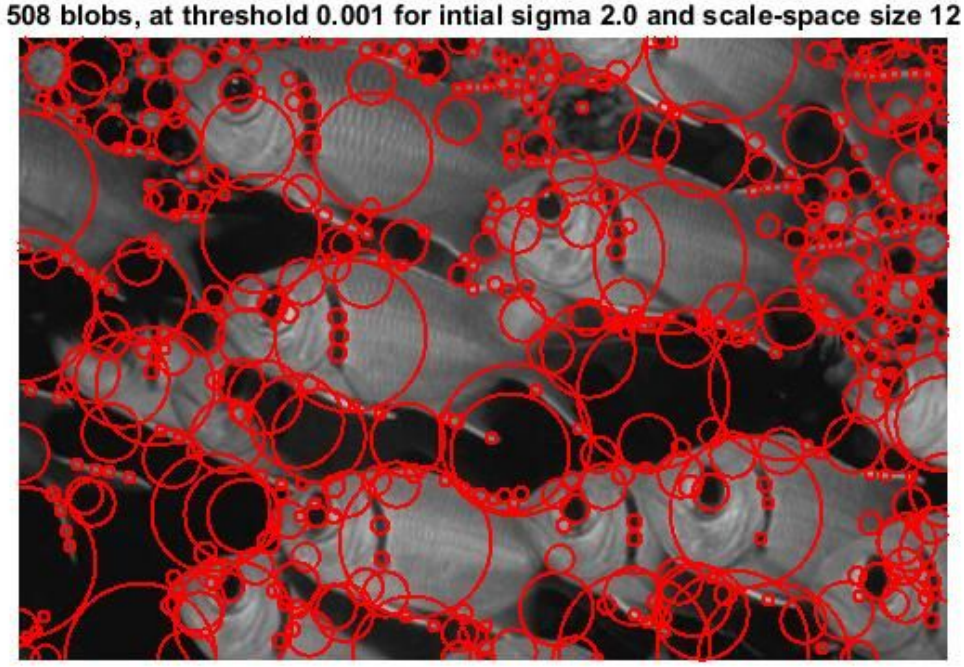
1. Outputs for each image: Image name ending in 1 or 2 denotes the method used for image filter response. 1 being efficient implementation and 2 being inefficient implementation. E.g., butterfly1 and butterfly2

butterfly1.jpg

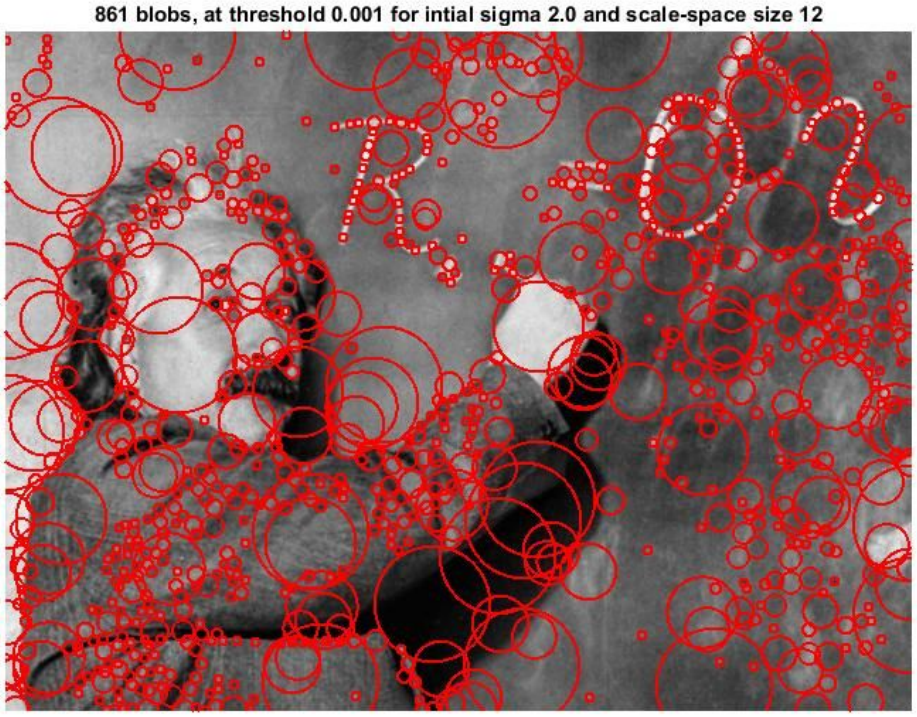
butterfly2.jpg



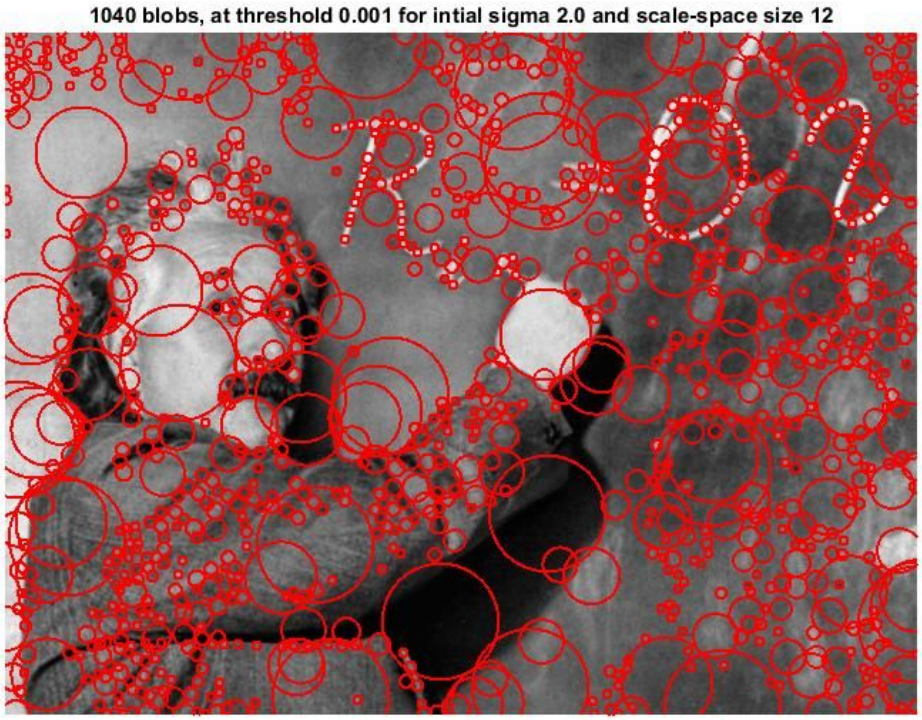
fishes1.jpg



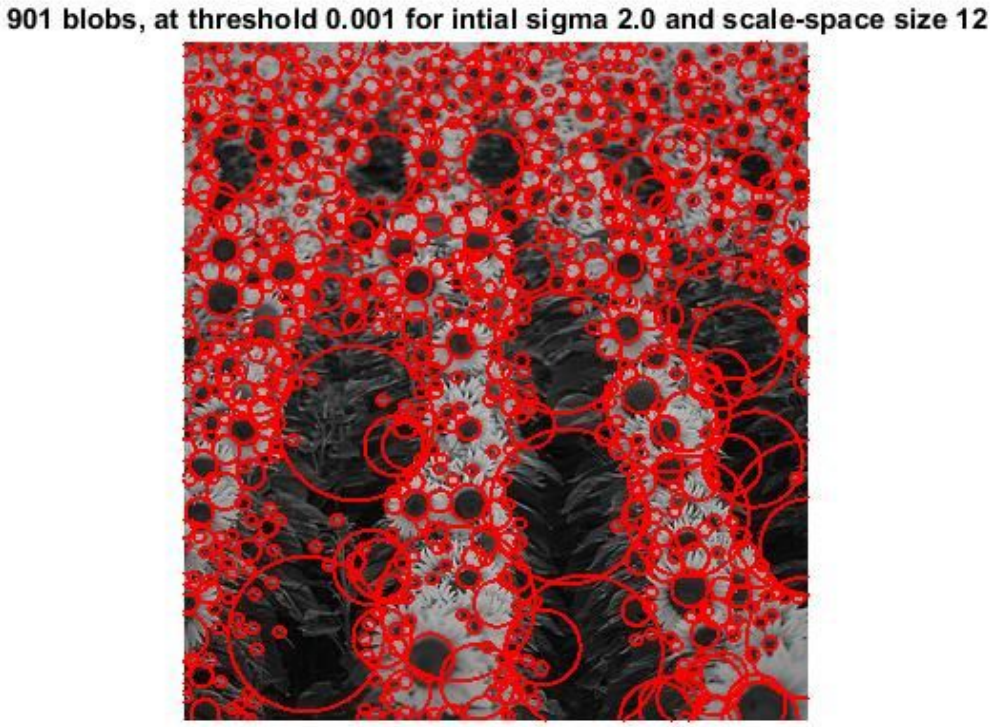
fishes2.jpg



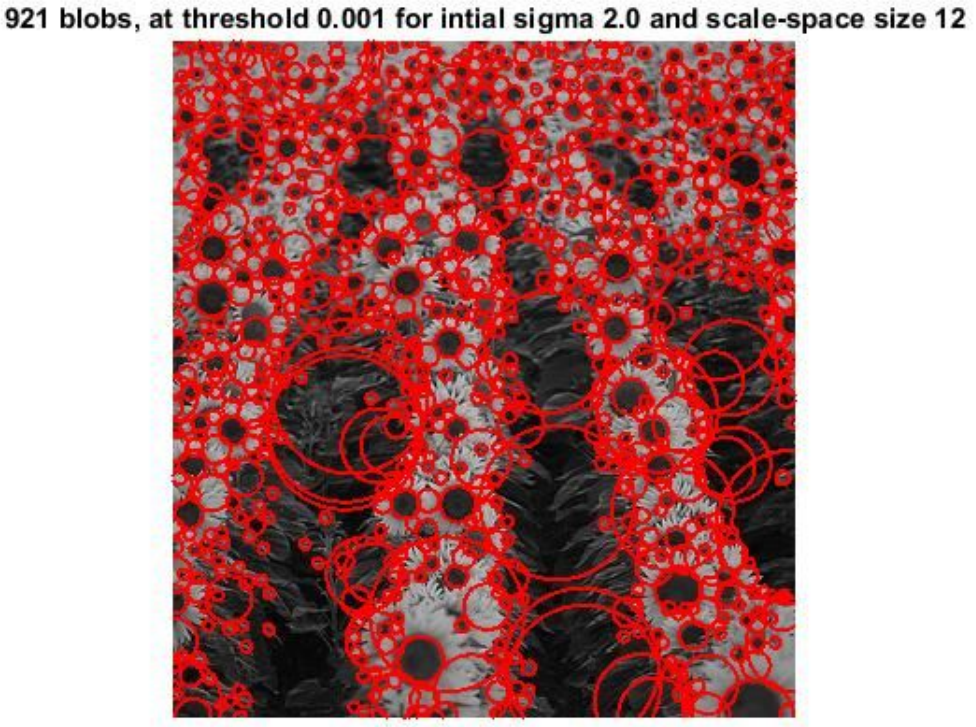
einstein1.jpg



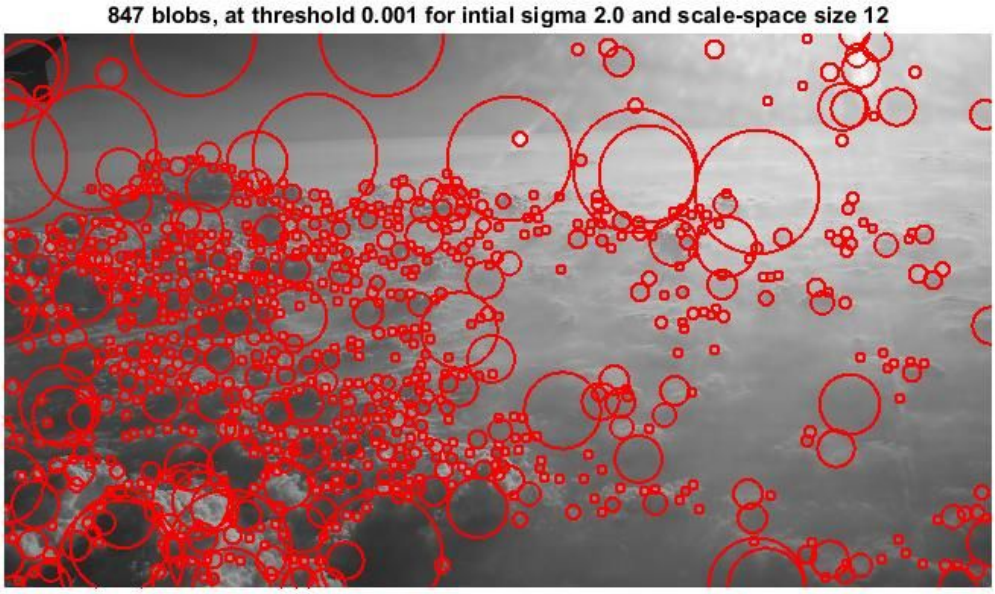
einstein2.jpg



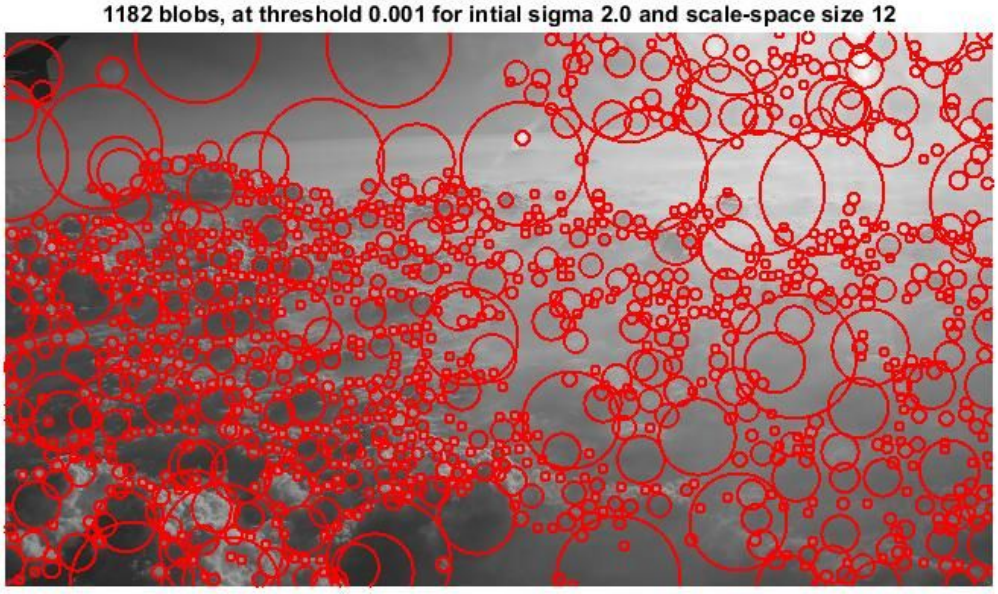
sunflowers1.jpg



sunflowers2.jpg

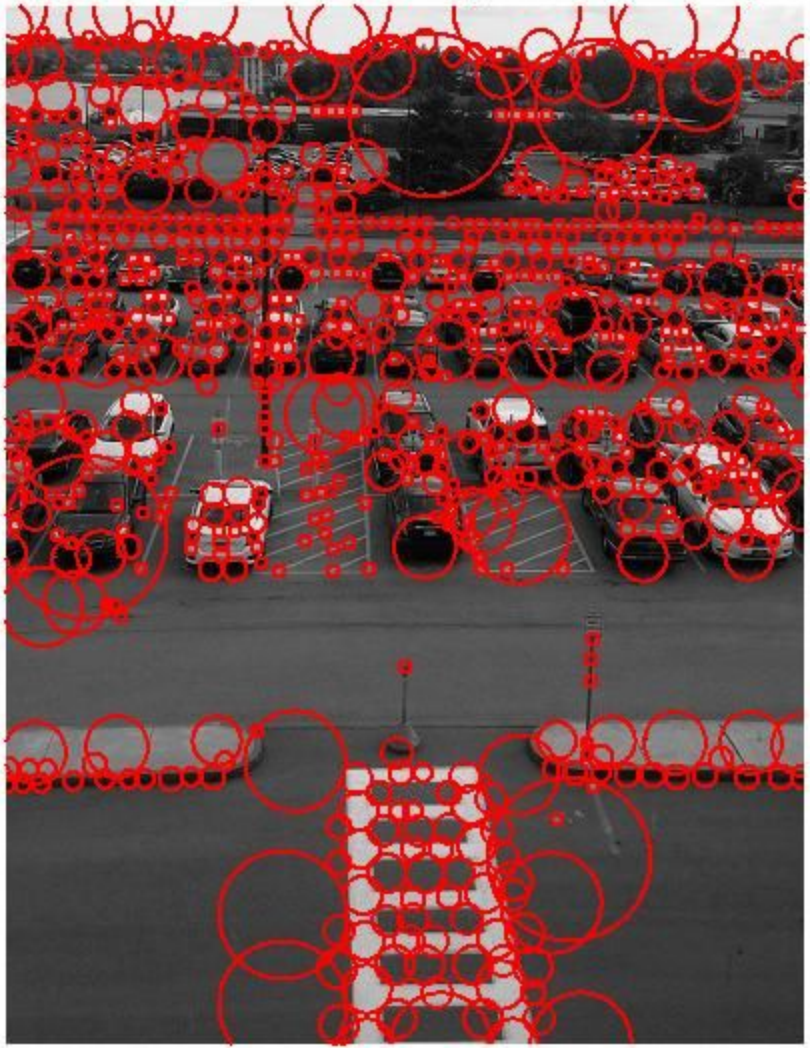


bangalore1.jpg



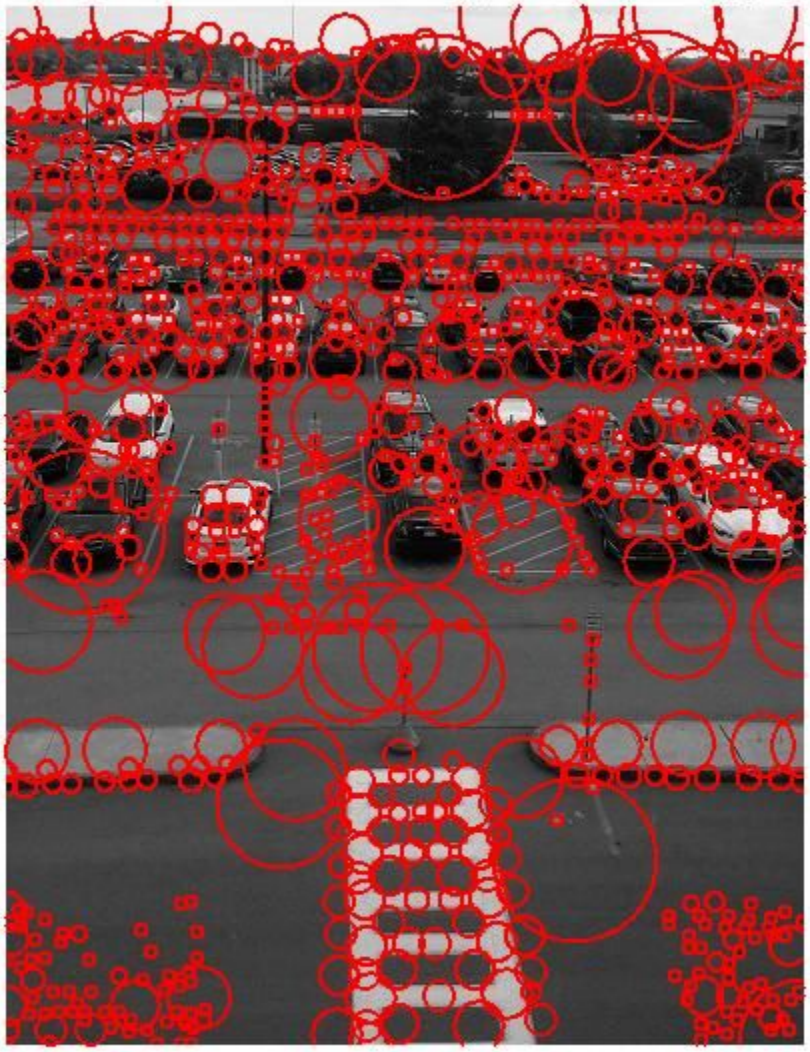
bangalore2.jpg

824 blobs, at threshold 0.001 for intial sigma 2.0 and scale-space size 12

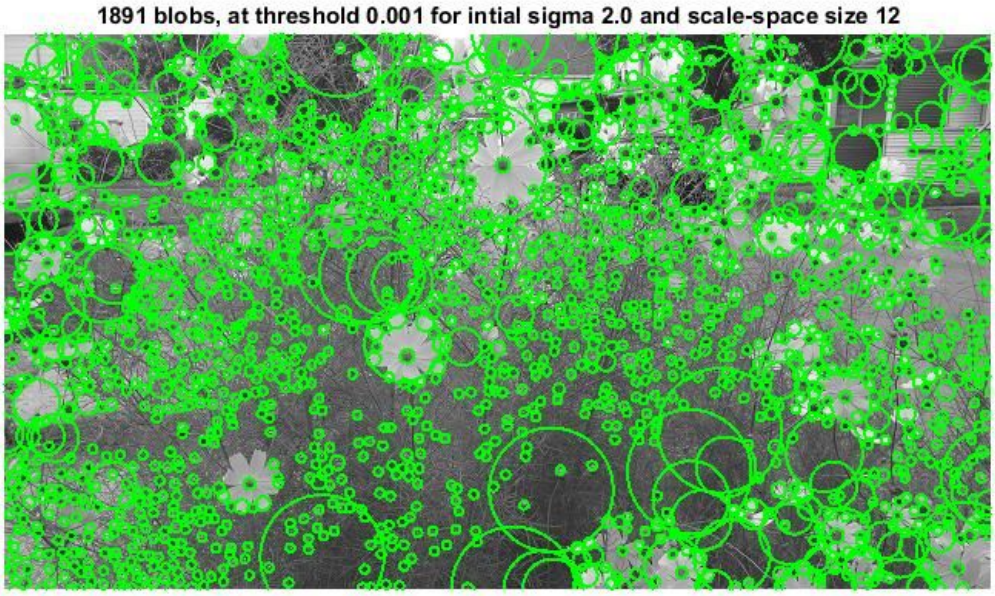


capen1.jpg

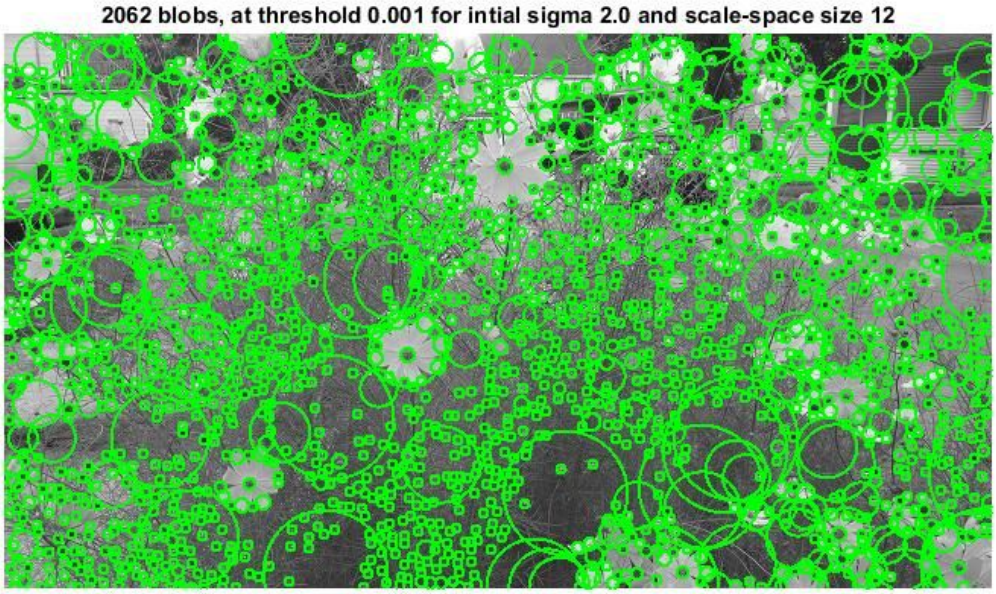
971 blobs, at threshold 0.001 for intial sigma 2.0 and scale-space size 12



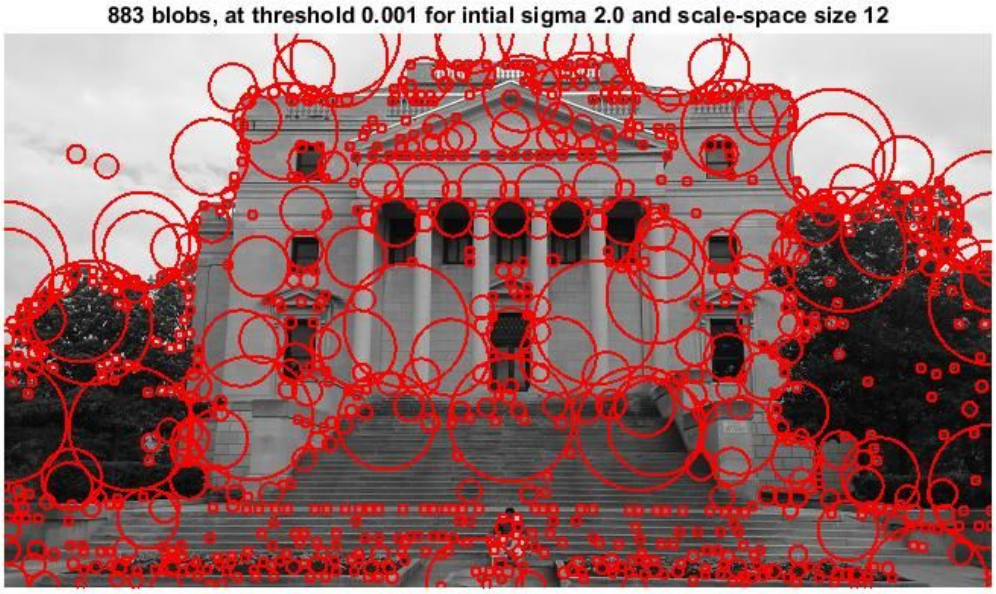
capen2.jpg



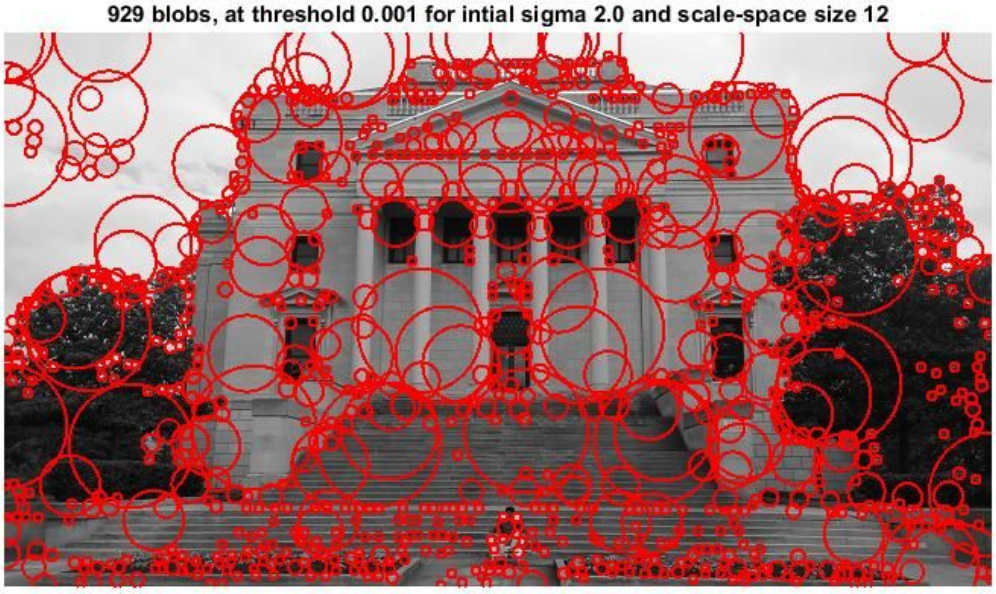
heath1.jpg



heath2.jpg



library1.jpg



library2.jpg

Comparison table

Sr. No.	image_name	initial_sigma	iterations	k	threshold	Running_time (Sec)	method	Total-Blobs
1	butterfly	2	12	2^.35	0.001	3.32	1	797
2	butterfly	2	12	2^.35	0.001	8.38	2	790
3	sunflowers	2	12	2^.35	0.001	2.35	1	901
4	sunflowers	2	12	2^.35	0.001	5.95	2	921
5	einstein	2	12	2^.35	0.001	5.74	1	861
6	einstein	2	12	2^.35	0.001	14.46	2	1040
7	fishes	2	12	2^.35	0.001	3.25	1	464
8	fishes	2	12	2^.35	0.001	8.41	2	508
9	heath	2	12	2^.35	0.001	4.45	1	1891
10	heath	2	12	2^.35	0.001	11.56	2	2062
11	library	2	12	2^.35	0.001	4.41	1	883
12	library	2	12	2^.35	0.001	11.77	2	329
13	bangalore	2	12	2^.35	0.001	4.96	1	847
14	bangalore	2	12	2^.35	0.001	11.58	2	1182
15	capen	2	12	2^.35	0.001	3.89	1	824
16	capen	2	12	2^.35	0.001	9.82	2	971

method	description
1	using efficient implementation
2	using inefficient implementation

We can clearly see that the blobs generation using implementation 1 which usage image downsizing is much faster than the implementation 2. Increasing value of sigma is the main cause of increasing the running time.

2. Above outputs are based on squared LOG responses. As mentioned earlier I have also experimented with the ***abs*** function, instead of squaring the LOG response, I have converted the negative values into the positive values using ***abs*** function of the matlab. Results are same only when the threshold value is changed, i.e., has to be increased to achieve similar pattern of the blobs. This can be explained by the fact that on squaring the pixels' values their values become much smaller, as their values is less than 1, so it decreases on squaring hence the lower level of threshold. One more change was done in the calculation of the characteristic scales in 3-D slices. As mentioned earlier, I found the characteristics scales with

two different methods. First method involved searching of the scale at global scale-space level for the maximum pixels response. Second method involved looking for the pixels maxima only at the neighbourhood scale levels (inspired from David Lowe's paper), i.e., i-1, i and i+1 scales. By my observation it looks second method is quite useful, as seen in the below pictures. Figure 5.1 is produced using 2nd method, as you one can see, the bigger circle has not dwarfed and removed the smaller circle beneath it compared to the image in Figure 5.2 produced using 1st method, there you can see bigger circle has dwarfed and removed the smaller circles. One can also notice the considerable increase in the number of blobs in 2nd method at same threshold and initial sigma values. Hence I can conclude that the 2nd method is more optimized and produces blobs/circles depicting smaller features which would have been removed by the blobs of the bigger feature produced by the 1st method. Do note that both outputs have been generated using the efficient image downsizing implementation.

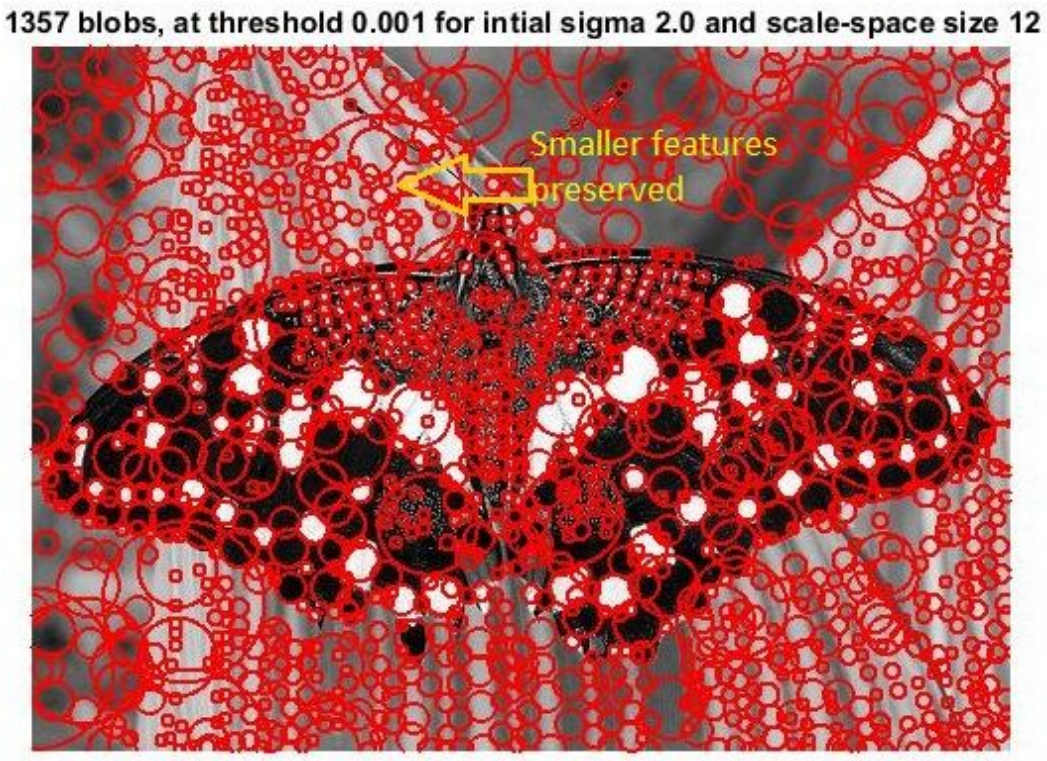


Figure 5.1

Figure 5.2

3. Smaller sigma value detect smaller features in the image whereas larger sigma values detects larger features in the image. So the value of parameter 'k' has to be chosen in such a way that the increased in the scale size is generalised enough the capture the essence of the image. I found out that **k = 2^0.35** is good enough. Also to avoid shifting of the objects I made sure that the kernel size of the LOG filter is odd and at least 5x5.

I experimented with different values of the matrix size in the `ordfilt2(A,order,domain)` function and found out that increase in the matrix mask (domain) and order size results in removal of the dense blobs. I found out that the domain of ones(7x7) with order value 49 is optimised enough to show the optimal number of blobs. Minimum has to be of ones(3x3) and domain 9. However larger size of domain matrix leads to increase in the runtime of the code.

Furthermore, threshold controls the number of circles, larger the threshold, smaller the circles/blobs count. For the image interpolation during image resizing, most effective methods were bicubic, cubic, lanczos3 and lanczos2.

HOW TO RUN THE CODE

Concerned files to look for inside the folder are:

- detectBlob.m
- custom_show_all_circles.m
- createFilter.m

Redundant files

- show_all_circles.m
- harris.m
- detectBlob2-redundant.m

detectBlob1(img, Sigma, n, threshold, color, mode, time) accepts img variable, initial sigma, threshold (0.001), color ('r') of the plot, 1 or 0 in mode for efficient or inefficient implementation, 1 or 0 for measuring time or not measuring time respectively.

To run, first read the image using imread and store it in a variable and pass it along with the other parameters in the detectBlob function. I have created a custom function for plotting the circles, which just returns the x and y coordinates of the circles calculated. Output images and original images can be found inside the output folder.

DATASHEET

1. Datasheet: https://goo.gl/fXLckT

SOFTWARE/HARDWARE USED

1. MATLAB R2017a on personal system.
2. Intel i3, 3GB Ram, Sublime Text 3 IDE.

REFERENCES

1. MATLAB Documentation
2. Hw2.pdf
3. Lecture slides

4. David G Lowe. Distinctive image features from scale-invariant keypoints.
5. Tony Lindeberg. Feature detection with automatic scale selection.
6. Sample Harris detector code.

==========END==========