ECE 558 FINAL PROJECT

BLOB DETECTION

Adarsh Puri|Apuri3|ECE 558 Digital Imaging Systems| Date-12/06/2019
Abhishek Ranjan Singh | Arsingh3 | ECE 558 Digital Imaging Systems | Date-12/06/2019
Rishabh Singh| Rsuhag | ECE 558 Digital Imaging Systems| Date-12/06/2019
(We would not like to be considered for extra credits)

BACKGROUND:

In computer vision, 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 like each other.

There are several motivations for studying and developing blob detectors. One main reason is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. In early work in the area, blob detection was used to obtain regions of interest for further processing. These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking. In other domains, such as histogram analysis, blob descriptors can also be used for peak detection with application to segmentation. Another common use of blob descriptor is as main primitives for texture analysis and texture recognition. In more recent work, blob descriptors have found increasingly popular use as interest points for wide baseline stereo matching and to signal the presence of informative image features for appearance-based object recognition based on local image statistics. There is also the related notion of ridge detection to signal the presence of elongated objects.

ALGORITHM:

1. Create Laplacian of Gaussian.
2. Build a Laplacian Scale Space.
3. Perform Non-Max Suppression in Scale Space
4. Display resulting circles at their characteristic scale.

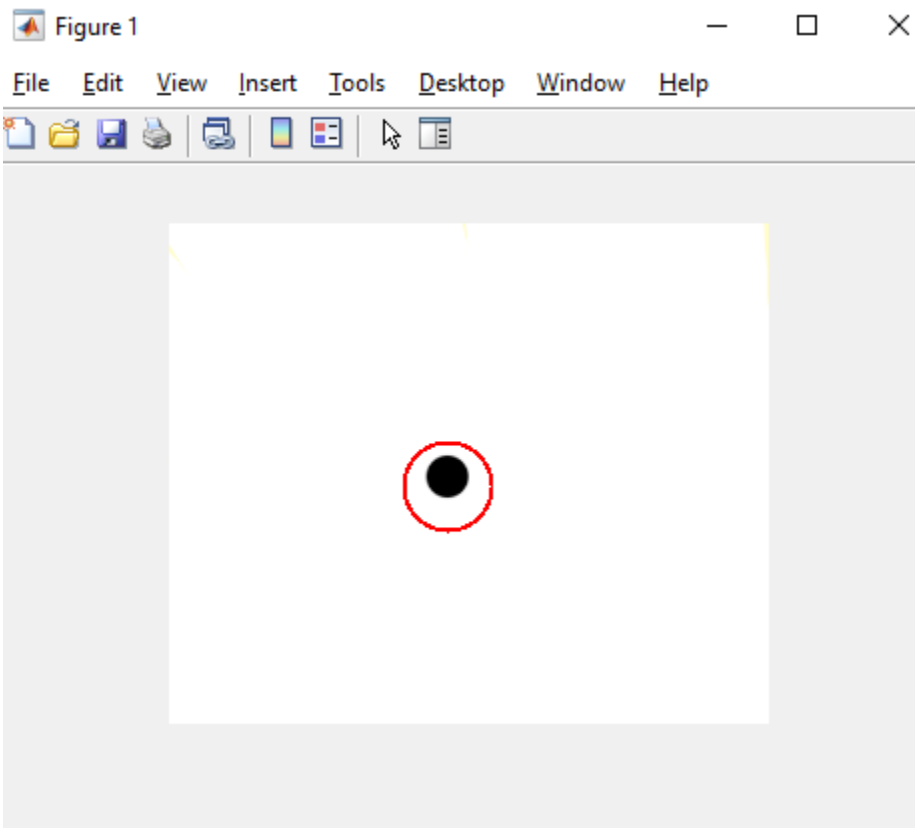
ALGORITHM IMPLEMENTATION:

Platform = MATLAB@2019Rb

1. Creating Laplacian of Gaussian:
   Functionality:
   We are creating Laplacian of Gaussian using the fspecial() function in MATLAB, which takes the input as filter size and standard deviation, including an argument ‘LoG’. The output of the function is then multiplied by variance to Normalize the LoG.
   Parameter:

The size of filter is calculated based on sigma, by the relationship filter_size = maximum of(1,6*sigma).
We experimented with values from1 to 7. For higher value of sigma more features were detected in some images but there were also some redundancies also the time required scaled exponentially with the size of sigma. For the smaller values of sigma time required reduced significantly but also some features were missing. We settled with sigma = 1.5 as for higher values of sigma there was not much to be gained and the program was sufficiently fast.

Result for Sigma = 7:



Time taken for sigma = 7

```
Warning: Integer operands are required for
> In convolve (line 5)
  In blobdetect (line 17)
  In main (line 19)

Elapsed time is 133.554513 seconds.
>>
```

2. Building a Laplacian Scale Space:
   - We are taking the scale space size and increasing it by two because the top and bottom most response will not be tested for maxima.

- For building Laplace scale, we are keeping the kernel size fixed and changing the size of image. By doing this the relative size of kernel to image changes at each iteration. This relative size is maintained to what it would have been if we changed the kernel size. By doing this we are able to reduce the time of convolution.
- We are maintaining the relative size by using image_scaling_at_layer function.
- We take an input image change its size and convolve it with our filter (LoG).
- After convolving we square the image and resize it to the original image size and store it in a 3D Matrix named scale_space for each iteration.

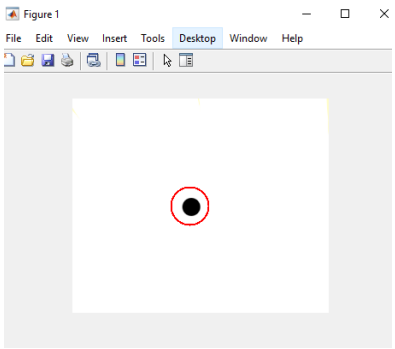
3. Performing Non-Max Suppression in Scale Space:
   - We first take a LoG response from scale space and apply the 2d non-max suppression.
   - For 2d-Non-Max suppression we are using the method suggested by Harris.
   - We have set the threshold to 999 after doing various trials.
   - We do this for all the LoG responses in the Laplacian Scale space and save the coordinates of maxima in a cell matrix.
   - Now we apply the Non-Max suppression at each scale space in 3-d Laplacian response to remove redundancy of repeating coordinates, hence, the features.
   - If maximas are detected at the edge of the image we are not using them for the blob features.
   - After doing this we save the radius of the blob at maxima coordinates which is relative to our size of image for each layer in scale space.
4. Displaying the resulting circle at their characteristics space:
   - Using the function viscircles() we are drawing the features on the input image.

RESULTS:

First image: Dot

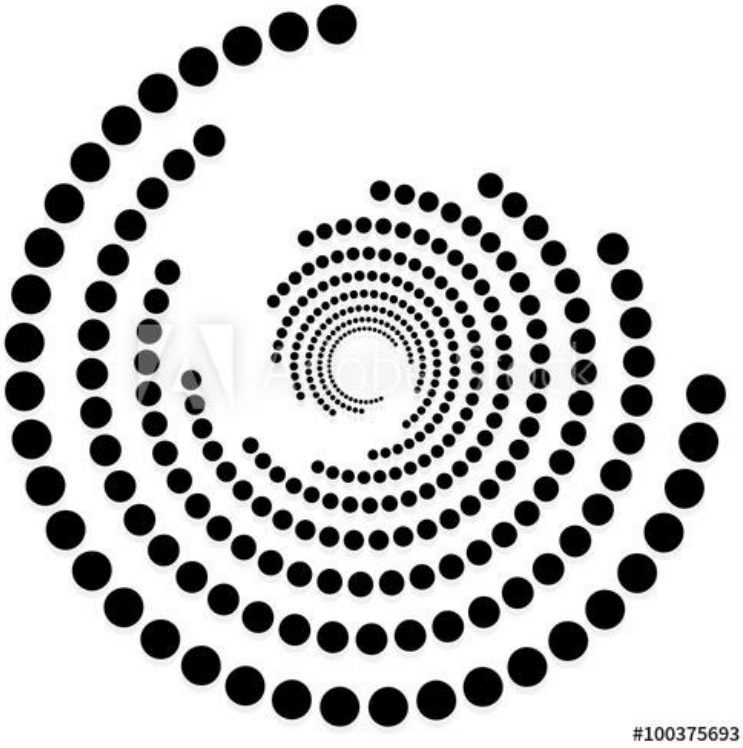
Input:

Output:

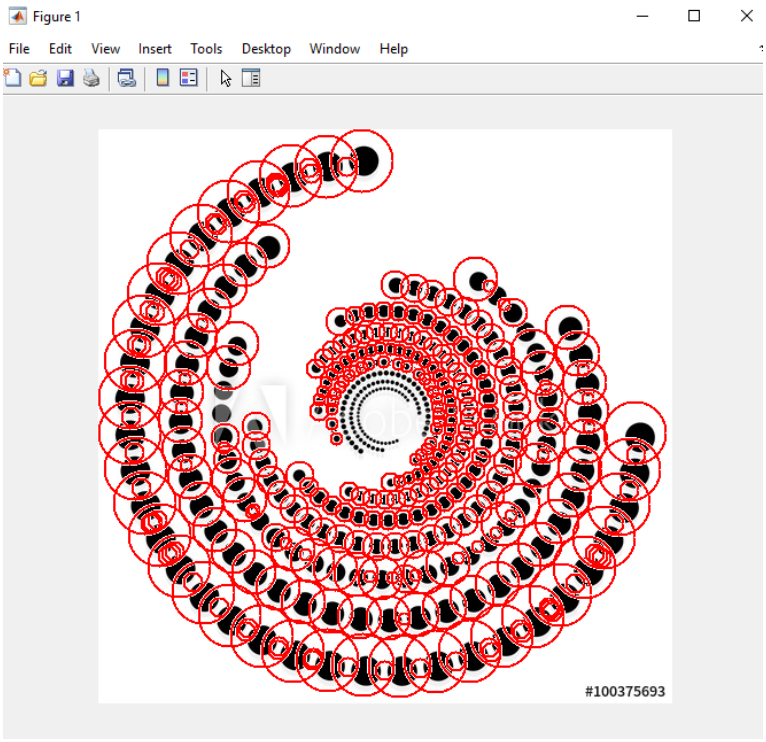


Second Image: Blob

Input:

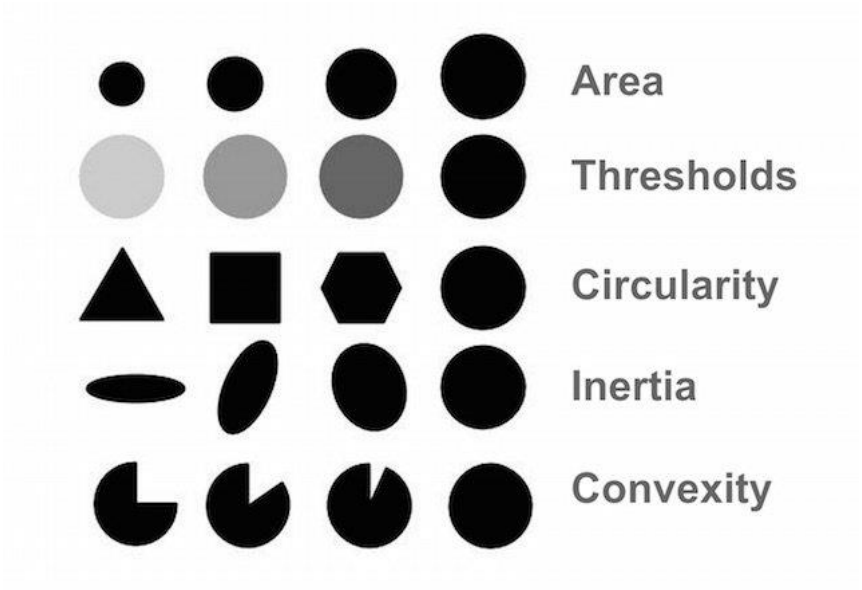


Output:

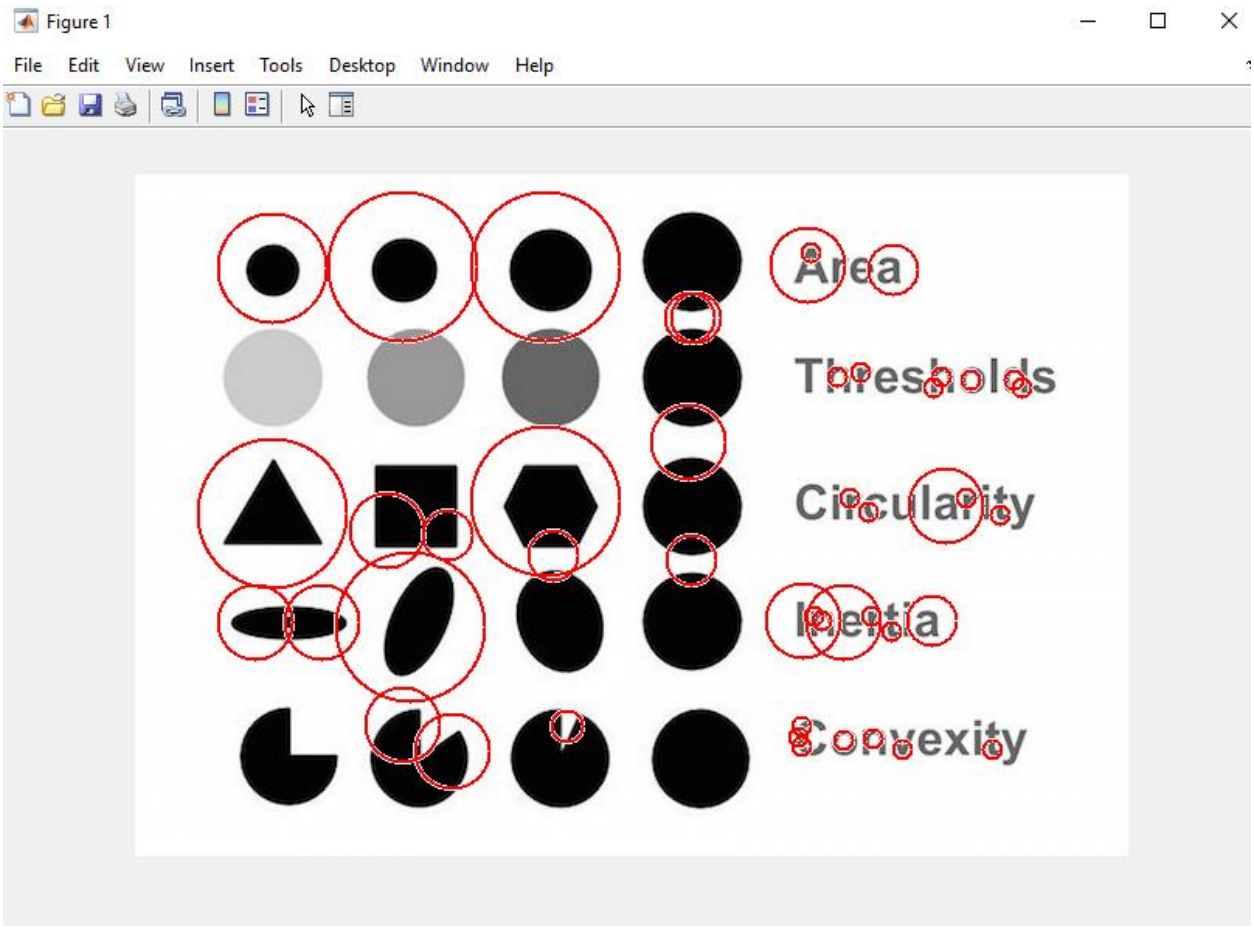


Third Image: Blob Test

Input:



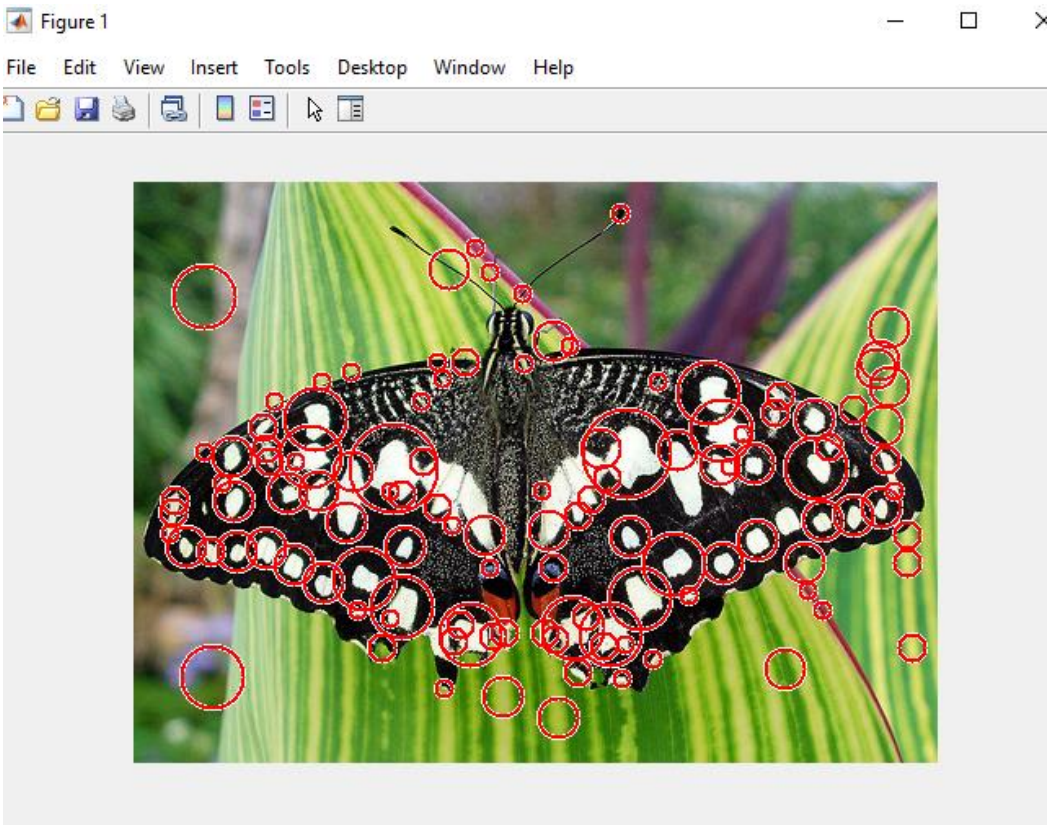
Output:



Fourth Image: Butterfly

Input:

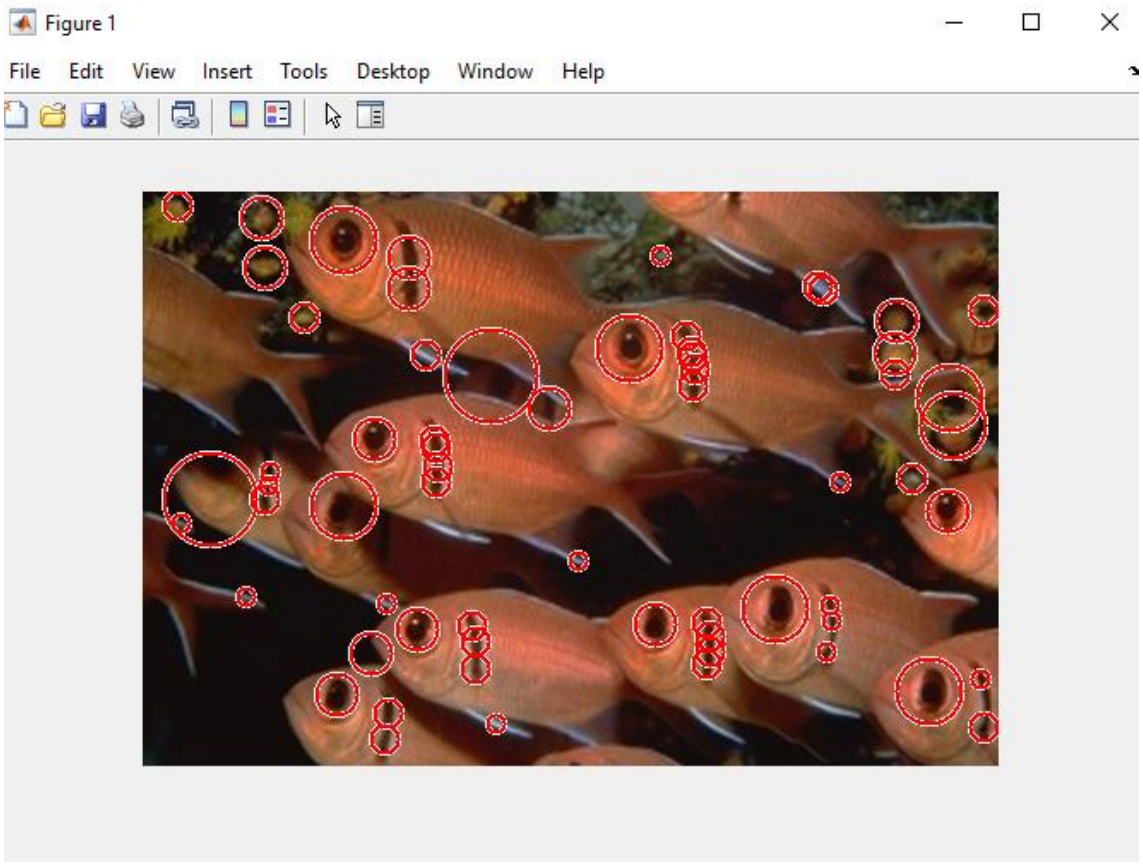
Output:



Fifth Image: Fishes

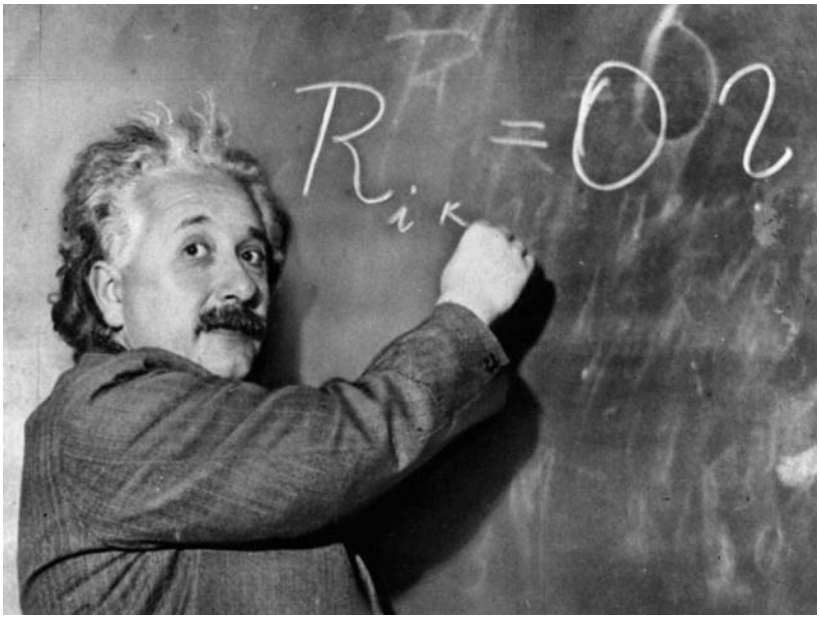
Input:

Output:

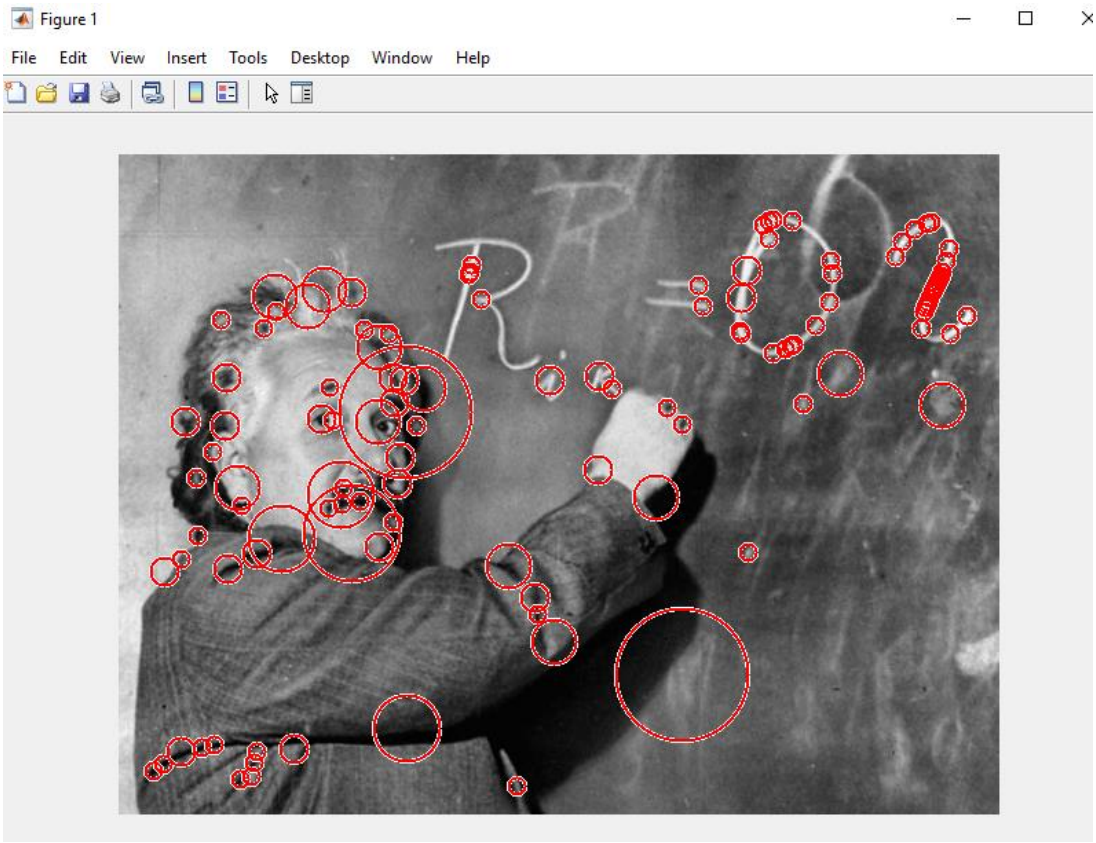


Sixth Image: Einstein

Input:



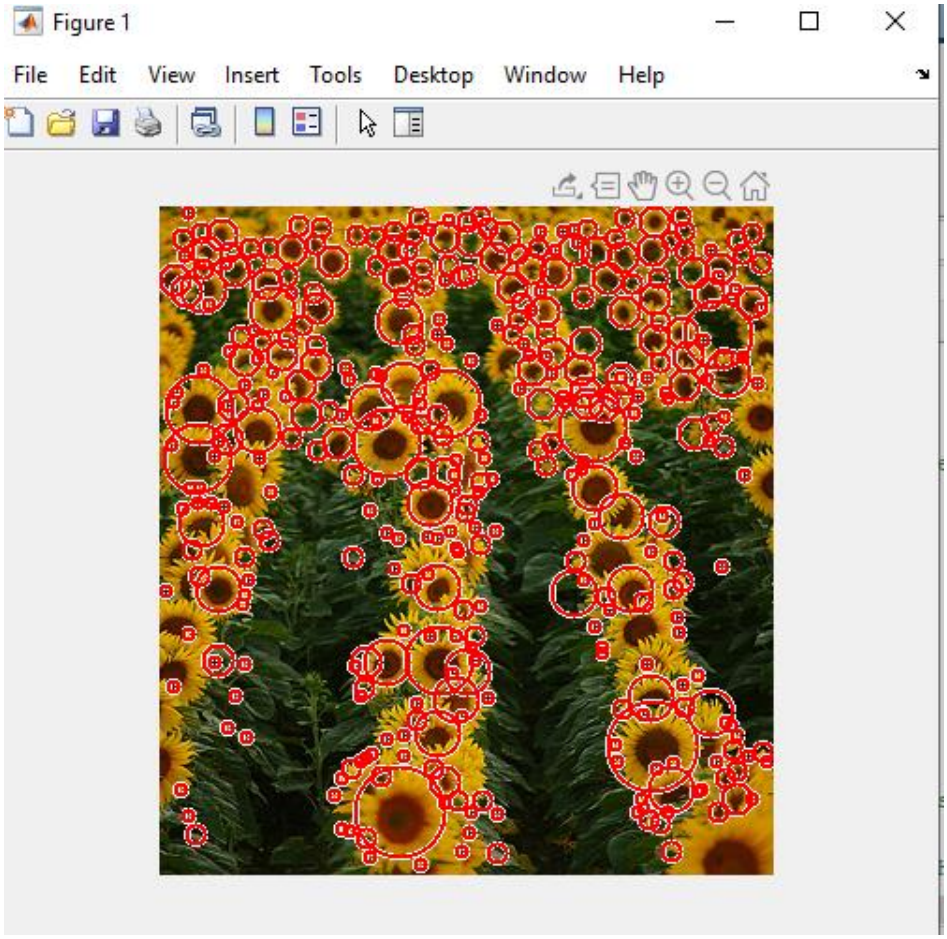
Output:



Seventh Image: Sunflower

Input:

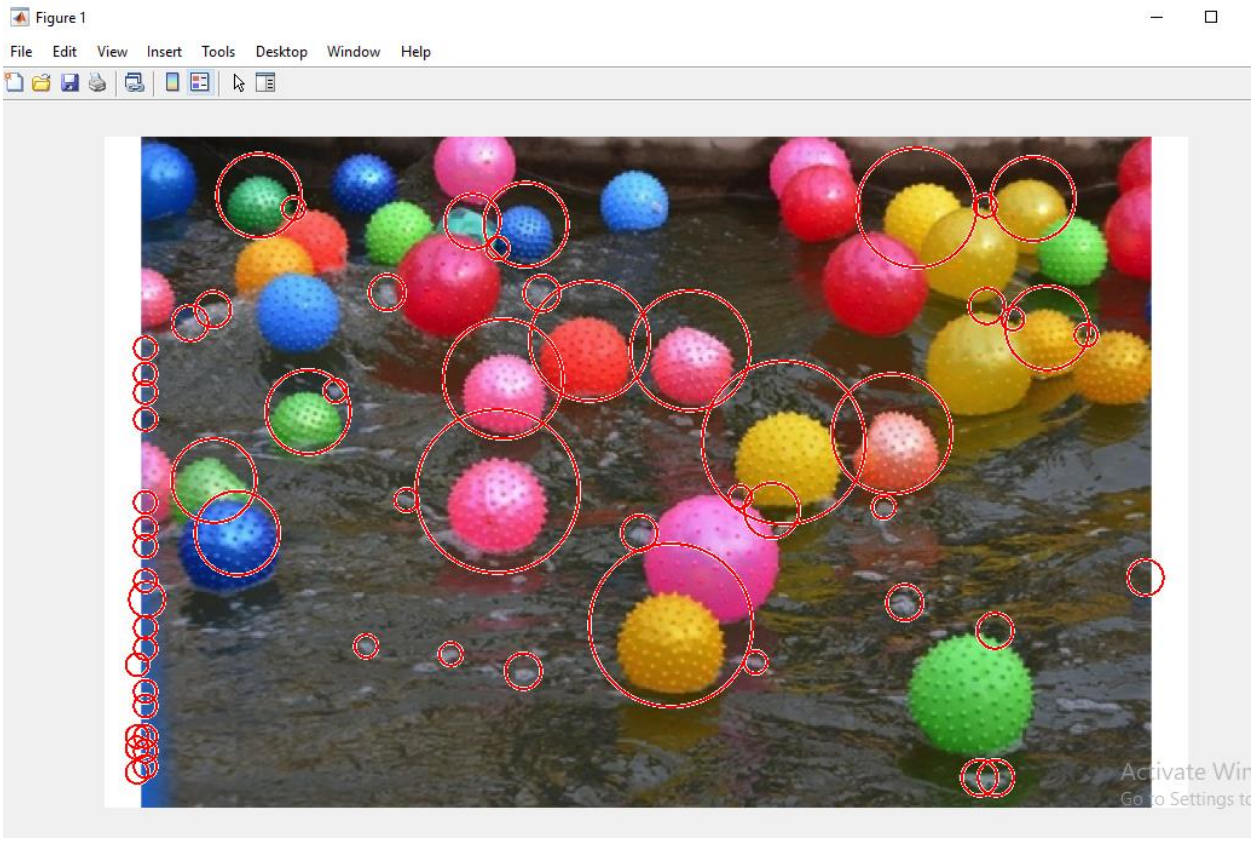
Output:



Eighth Image: Playball

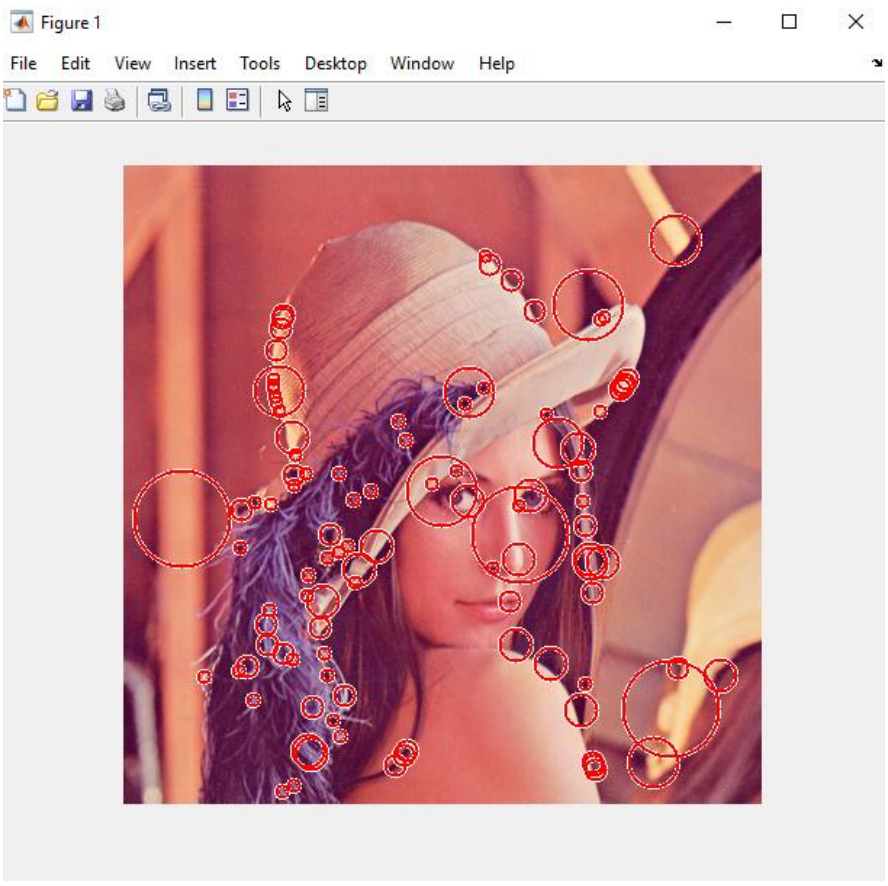
Input:

Output:



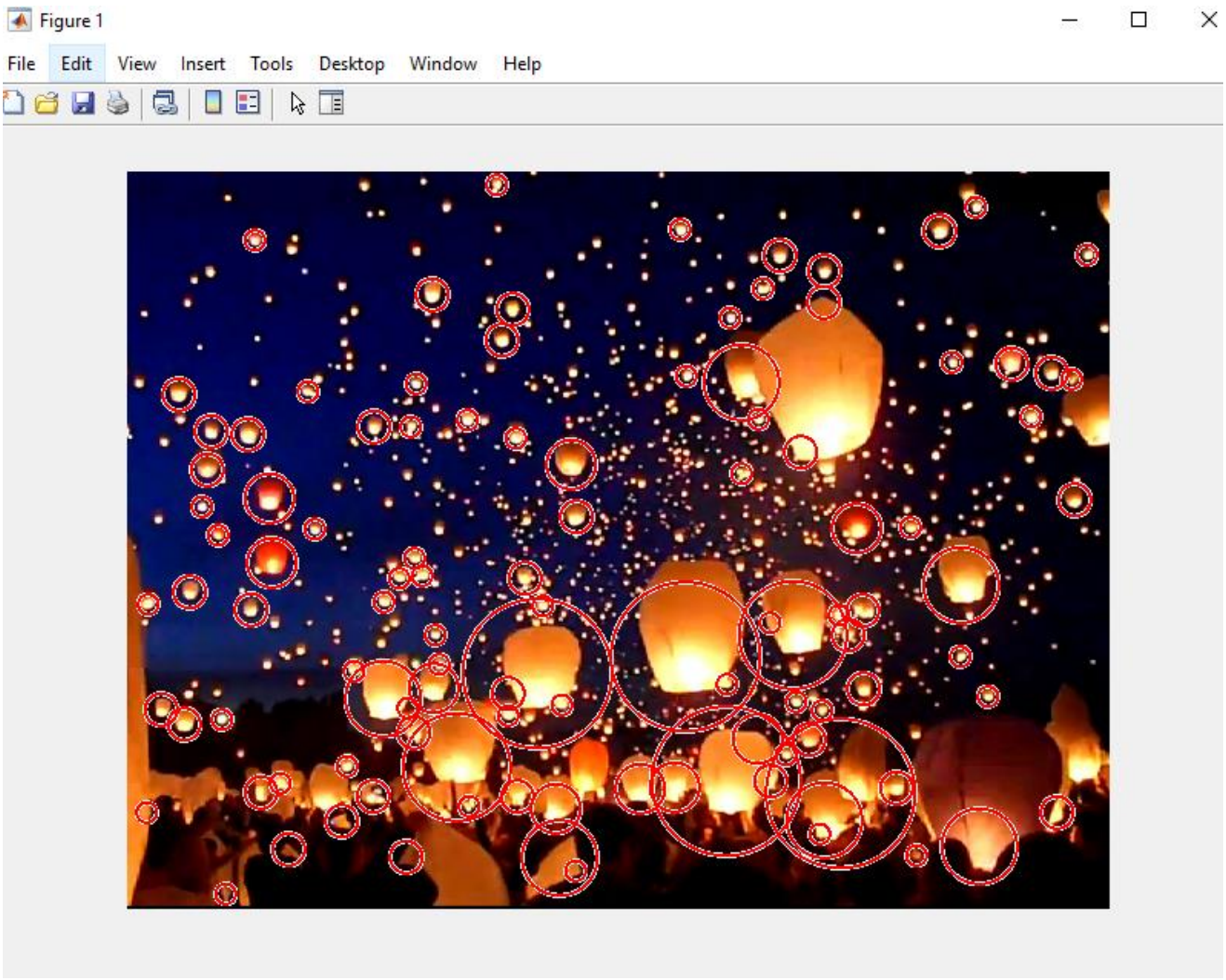
Ninth Image: Lena

Output:



Tenth Image: Lanterns

Output:



APPENDIX:

To Run the code open main function and run it.

main Function:

```
clear all;
close all
clc
[file1,path1,~]=uigetfile('*.png; *.jpg; *.jpeg'); % select a file from drive
if isequal(file1,0)
   disp('User selected Cancel')     % if user cancel selection
else
   disp(['User selected ', fullfile(path1, file1)]) %else if user makes a file
selection
end
im=imread(horzcat(path1,file1));        %Reading the image
img=double(rgb2gray(im));       %taking float value of image data
[m,n]=size(img);            % taking the size of image
initial_value=1.8;     %the initial value in which I will vary my kernel size relative
to image
end_value=20;       %the end value in which I will vary y kernel size relative to
image
scale_space=6;      %the scale space steps to take
sigma = 1.5; %value of standard deviation to be input as seen in Harris blob detector
%sigma=1; %2; %3; %4; %;5 %6; %7 ; % differet sigma test runs
tic                 %to start counting clock for speed of algo
[coordx,coordy,radii]=blobdetect(img,sigma,initial_value,end_value,scale_space);
%blobdetect function to find blobs
app=[];         %empty array to save blobs x and y coordinates
imshow(im); hold on;            %display image
app(:,1)=coordx;
app(:,2)=coordy;
radii=fix(n*radii*0.5);
viscircles(app,radii,'Color','r','LineWidth',1.5);  %draw circle function
toc                     % stop the counting clock
```

blobdetect function:

```
function
[max_y_coordinate,max_x_coordinate,max_radius]=blobdetect(image,sigma,a,b,scale_size)
    [m,n]=size(image);          %size of image is saved in m and n
    scale_space=zeros(m,n,scale_size);      %scale space vector is initialised(3d
vector)
    max_occurs = zeros(m,n,scale_size);         % local variable to find maximas at
each 2d scale_space
    two_dimension_maximas=[];           % the 2dimensional maximas are saved here
    max_x_coordinate=[];                % coordinates of x,y and radii where maximas
occur.
    max_y_coordinate=[];
    max_radius=[];
    filt_size= max(1,fix(6*sigma));            % taking filter size as seen in Harris
blob detector.
    normalised_log=(sigma.^2)*fspecial( 'log',filt_size,sigma );    %using fpscial
function to get laplaciun of gaussians.
    %also normalised by multiplying
    scale_size = scale_size + 2;  % scale space taken 2 extra because the top and last
responses will not be tested at maximas.
    % generating the Laplacian of gaussian response at different scales
    for i=1:scale_size
        new_image_size = filt_size /(image_scaling_at_layer(i,a,b,scale_size));
%image scaling factor relative to kernal size
```

```
        upscaled=imresize(image,new_image_size/n, 'cubic');         %using new size of
image and resizing
        log_response=(convolve(upscaled,normalised_log));        %convolving two
inputs
        log_response=log_response.^2;                   %square of Laplacian response
for current level of scale space.
        log_response= imresize(log_response,[m n],'cubic');            % resize back
to save the image for same coordinates scale
        scale_space(:,:, i) =log_response;              % appending in 3d matrix
    end
    %finding the local 2 dimensional maxima for same scale space at same layer
    %Harris code 2d non mux suppression
    i=i-scale_size+1;
    while(i<=scale_size)
        current_image = scale_space(:,:,i);
        threshold = 999;       %doing image dilation and matching it with a threshold
        window_size=3;      %window size for maxima checking
        aa=ordfilt2(scale_space(:,:,i), window_size^2, ones(window_size)); % grey
scale dilation
        max_occurs(:,:,i) = (scale_space(:,:,i)==aa)&(scale_space(:,:,i)>=threshold);
%finding for maxima
        [r,c]=find(max_occurs(:,:,i));         %getting row and column of those
maximas
        two_dimension_maximas= [two_dimension_maximas, {[r,c]}];    %saving in a cell
matrix or list
        i=i+1;
    end
    for layer=2:(scale_size-1)              %now finding the maximas at each scale
space in 3D
        temp_layer= two_dimension_maximas(layer);
        temp_coordinates=temp_layer{1};
        temp_coord_size= size(temp_coordinates);
        radius = image_scaling_at_layer(layer,a,b,scale_size);
        filter_size = fix(n*image_scaling_at_layer(layer,a,b,scale_size)*(1/2));
        for i=1 : temp_coord_size(1)
            temp_point= temp_coordinates(i,:);
            x = temp_point(1);
            y = temp_point(2); % if the maximas are detected at the edge of image , we
will not use them for blob features
            if(x<=filter_size)||(x>m-filter_size)
                continue;
            end
            if(y<=filter_size)||(y>n-filter_size)
                continue;
            end
            local_maxima = scale_space(x,y,layer); % local maxima at the current scale
space
            upper_values = [scale_space(x-filter_size:x+filter_size, y-
filter_size:y+filter_size,layer+1) ]; %window in scale space above current
            upper_maxima = max(upper_values(:)); %maxima above current scale space
            lower_values = [scale_space(x-filter_size:x+filter_size, y-
filter_size:y+filter_size,layer-1) ];    %window in sale space below current
            lower_maxima = max(lower_values(:)); %maxima below current scale space
            if( (local_maxima > upper_maxima) && (local_maxima > lower_maxima))
                max_x_coordinate = [max_x_coordinate, [x]];     %if current is more
than both then the coordinates of x,y and radii are appended
                max_y_coordinate = [max_y_coordinate, [y]];
                max_radius = [max_radius, [radius]];
            end
        end
    end
```

Convolve Function:

```
function [f]= convolve(f,ker)
[m,n,r]=size(f);        %size of image
    [x,y]=size(ker);        % kernel size
    pad_rgb=zeros(m+x-1,n+y-1,r);
    pad_rgb((x+1)/2:m+((x-1)/2),(y+1)/2:n+((y-1)/2),:)=f(1:m,1:n,:);    %padding
depending upon kernel size
    [m,n,r]=size(pad_rgb);
    f=zeros(m-x+1,n-y+1,r);
     for j=1:n-y+1          %convolving
         for i=1:m-x+1
            temp=pad_rgb(i:i+x-1,j:j+y-1,:);
            temp=temp(:,:,:).*ker;
            bb=sum(sum(temp));
            f(i,j,:)=abs(bb);
         end
     end
```

image_scaling_at_layer function:

```
function percent = image_scaling_at_layer(i, start_perc, end_perc, scale_step_size)
    step_size = (end_perc - start_perc) / (scale_step_size-1); % function which will
generate image size relative to kernel size.
    percent = (((i-1) * step_size) / (end_perc - start_perc))^2 * (end_perc -
start_perc) + start_perc;
    percent = percent / 100;            % image scaling percentage level( how much
will be scaled relative to kernel)
```

REFERENCES:

1. Lecture Slides
2. Digital Image Processing, 4th edition Gonzalez and Woods Pearson/Prentice Hall © 2018
3. https://en.wikipedia.org/wiki/Blob_detection
4. https://matlab.fandom.com/wiki/FAQ#How_do_I_create_a_circle.3F
5. http://www.cs.utah.edu/~manasi/coursework/cs7960/p1/project1.html
6. https://www.kixor.net/
7. http://www.sci.utah.edu/~weiliu/class/aip/p1/
8. https://en.wikipedia.org/wiki/Corner_detection
9. https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf