ECE 558 – Digital Imaging Systems

Project 03 – Final

Submitted by –

Akhil Bukkapuram (NCSU ID: 200481004)

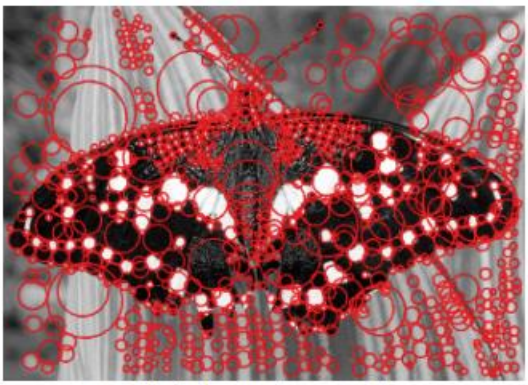
Surya Dutta (NCSU ID: 200481187)

Option 3

(We would like to be considered for extra credit)

Project Description: The objective of this project is to implement a Laplacian Blob Detector for feature tracking applications in computer vision.

An input image



A blob detection result

Algorithm Outline:

The algorithm outline for this blob detector is as follows –

- Generating a Laplacian of Gaussian filter.
- Building a Laplacian scale space, starting with some initial scale and then going for *n* iterations.
    - Filtering the image with the scale-normalized Laplacian at current scale.
    - Saving the square of the Laplacian response for the current level of scale space.
    - Increase scale by a factor k.
- Performing non-maximum suppression in the scale space.
- Displaying the resulting circles at their characteristic scales.

Generating the Laplacian of the Gaussian Filter:

The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The implementation is shown as follows. The kernel's size increases because the standard deviation is multiplied by the factor *k*. The initial value of sigma is multiplied by k^i where *i* represents the number of layers of the LoG filter.

```
#function to generate LoG kernel
def generate_LoG(sigma):
    #Adjust kernel size
    n = math.ceil(sigma*6)
    #x and y range to center the kernel
    x = np.array(range(-n//2,n//2 +1))
    x=np.reshape(x,(1,x.shape[0]))
    y = np.array(range(-n//2,n//2 +1))
    y=np.reshape(y,(y.shape[0],1))

    x_filter = np.exp(-((x**2)/(2.*(sigma**2))))
    y_filter = np.exp(-((y**2)/(2.*(sigma**2))))
    #LoG kernel
    LoG_filter = (-(2*(sigma**2)) + (x**2 + y**2) ) * (x_filter*y_filter) * (1/(2*(np.pi*sigma**4)))
    return LoG_filter
```

Building the Laplacian Scale Space:

First, the input image is normalized and then convolved with the scaled LoG kernel. We implemented the standard pixel-wise convolution in spatial domain from project 02. Prior to performing the convolution, the input image is padded with zero padding. The resulting convolved image is then squared in order to make sure that the peaks or maxima is obtained. During each convolution, the input image remains the same and only the kernel size changes (scaled by a factor k). This array of squared convolved images, which result from the convolution of the input image with the scaled LoG kernel makes up the Laplacian Scale Space. The implementation is shown as follows.

```python
#Function for convolution with appropriate padding
def conv2d(image,kernel):
    kernel = np.flipud(np.fliplr(kernel))
    output = np.zeros(image.shape)

    px = (kernel.shape[0]-1)
    py = (kernel.shape[0]-1)
    pad = np.zeros((image.shape[0] + px, image.shape[1] + py))
    #zero padding
    pad[int(px/2):int(image.shape[0]+px/2),int(py/2):int(image.shape[1]+py/2)] = image[:,:]
    #Convolution
    for y in range(image.shape[1]):
        for x in range(image.shape[0]):
            output[x, y]=(kernel * pad[x: x+kernel.shape[0], y: y+kernel.shape[1]]).sum()
    return output
```

```python
#Building the laplacian scale space
Img_conv = []
#Run for n iterations
for i in range(Layers):
    #Scaling the standard deviation in each iteration
    #Producing the LoG kernel for required standard deviation
    LoG_filter = generate_LoG(sigma * np.power(scale, i))
    #Convolve the LoG filter
    filtered_LoG = conv2d(img, LoG_filter)
    #Squaring the LoG response
    squared_LoG = np.square(filtered_LoG)
    Img_conv.append(squared_LoG)

Laplacian_space = np.array(Img_conv)
#Non Max Suppression
max_space = non_max_supp(Laplacian_space, Layers)
```

Performing Non-Max Suppression:

In non-max suppression, we consider a local neighborhood, and remove all the pixels which do not have the maximum response. This is done in two steps, where firstly, the individual layers of the Laplacian Scale Space (2D) are considered and non-max suppression is performed on them. Then, all the layers are considered (3D), and non-max suppression is done and grouped together to obtain the non-max suppression over the whole scale space. Thus, the maximum values of the pixels across the scale space are obtained along with their locations. The other values in that region are set to zero, in order to avoid overlapping of blobs. The implementation is shown as follows.

```
#Non Max Suppression
def non_max_supp(Laplacian_space, Layers):
    #Size of images
    r, c = Laplacian_space[0].shape

    #Suppression in individual layer
    supp_each_layer = np.zeros((Layers,r, c), dtype='float32')
    for i in range(Layers):
        py_layer = Laplacian_space[i, :, :]
        for j in range(r):
            for k in range(c):
                supp_each_layer[i, j, k] = np.max(py_layer[j:j+3 , k:k+3])

    #Suppression among different layers
    supp_layer_lvl = np.zeros((Layers, r, c), dtype='float32')
    #Start from 2nd bottom most layer
    for i in range(1, np.size(supp_each_layer,1)-1):
            for j in range(1, np.size(supp_each_layer,2)-1):
                supp_layer_lvl[:, i, j] = np.max(supp_each_layer[:, i-1:i+2 , j-1:j+2])
```

Displaying the resulting circles at their characteristic scales:

Once the maxima values are obtained, thresholding is done to the peaks that have values greater than or equal to 0.03. The array of blobs is hence obtained, and the radius of the circles is adjusted based on the layer. Lastly, these are plotted on the figure. The implementation is shown as follows.

```
#Detecting the maxima and thresholding
def thresholding(max_space, scale, sigma,th=0.03):
    blobs = []

    for j in range(max_space[0].shape[0]):
        for i in range(max_space[0].shape[1]):
            local_neigh = max_space[:, j:j+3 , i:i+3]
            maxima = np.max(local_neigh)
            if maxima >= th:
                layer,y,x = np.unravel_index(local_neigh.argmax(), local_neigh.shape)
                scaled_sd = np.power(scale,layer)*sigma
                blobs.append((scaled_sd,j+y, i+x))
    return blobs
```

```
#detecting the blobs
blobs =  thresholding(max_space, scale, sigma,th=0.03)
#remove duplicates
blobs = set(blobs)
#Plotting the circles on the image
fig, ax = plt.subplots()
ax.imshow(img, cmap="gray")
for blob in blobs:
    scaled_sd,y,x = blob
    ax.add_patch(plt.Circle((x, y), scaled_sd*scale, color='red', linewidth=0.3, fill=False))

ax.plot()
plt.show()
```

Results:

The input image is taken as grayscale and the intensity is scaled to between 0 and 1. The code is executed with Python 3.7 on Spyder IDE 5.2.2.

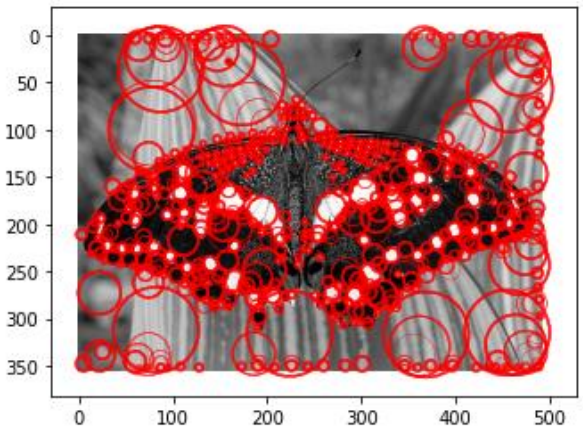
The following constants were used:

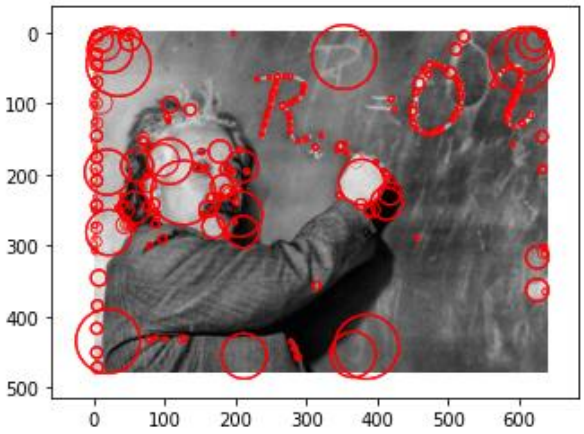
k = 1.414

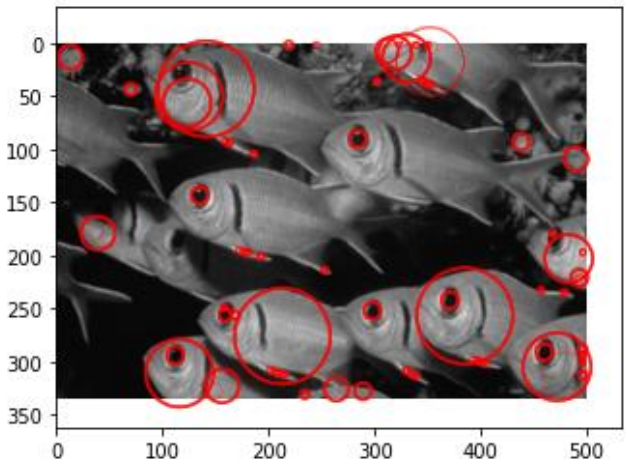
sigma = 2

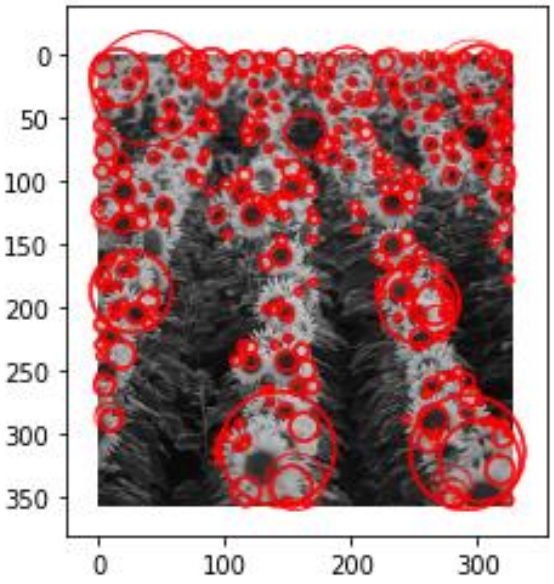
Number of layers = 9

Results of the images provided:

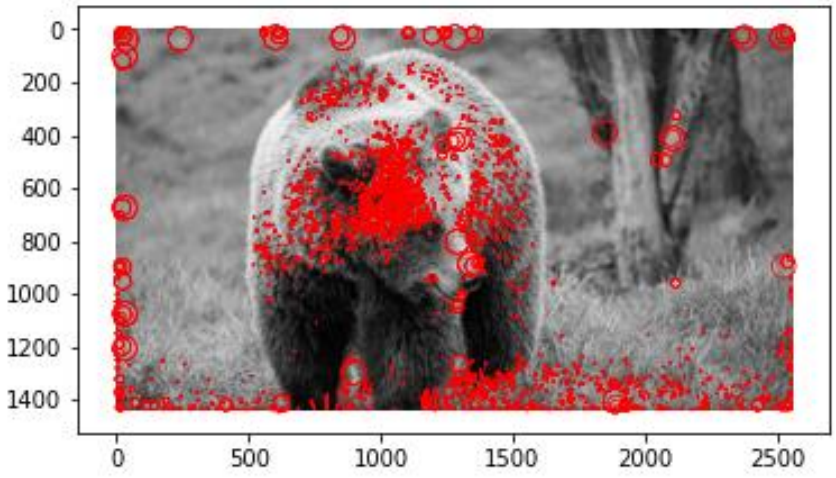


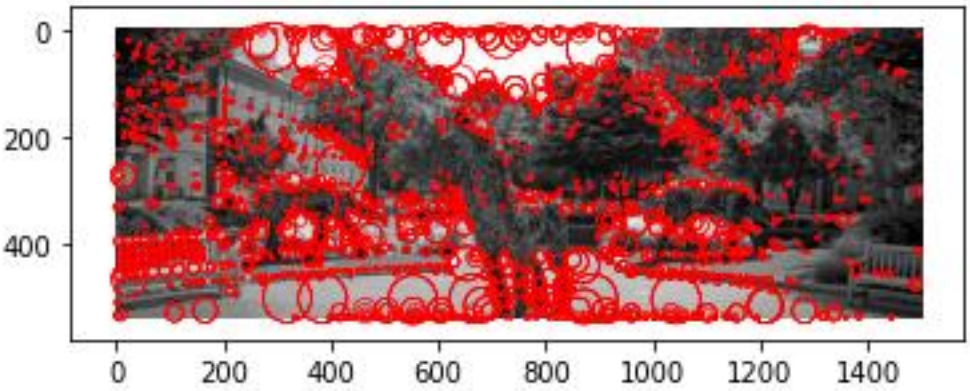


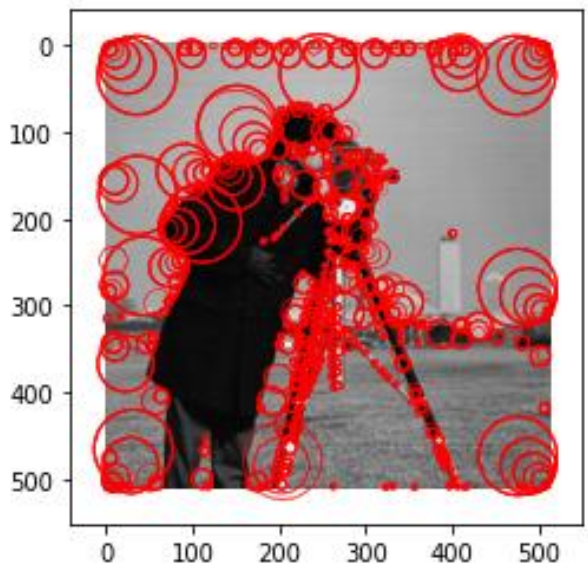


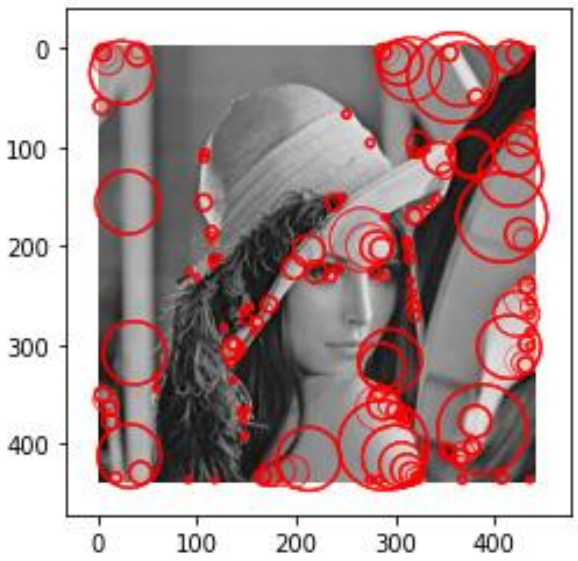


Results of select images chosen by us:









We observed that, as the size of the image increases, the program runtime also increases.

Individual Contributions:

Akhil Bukkapuram: Implementation of LoG Kernel, Non-Max Suppression, Convolution & Code Profiling.

Surya Dutta: Implementation of Laplacian Scale Space, Thresholding, Blob Plotting & Report Writing.

How to run the code:
In order to run the code and get results, make a copy of the original images and put them in the same location of the code. Run the .py file. We have used only the basic python libraries like OpenCV, Numpy, Matplotlib, and Math.

- Installation of dependencies

  Download and install Anaconda (https://www.anaconda.com/). Run the following on an anaconda prompt.
  conda create -n ece558 python=3.7 -y
  conda activate ece558
  pip3 install –upgrade pip
  pip install numpy
  pip install opencv-python
  pip install matplotlib

References:

- https://en.wikipedia.org/wiki/Blob_detection
- https://numpy.org/doc/stable/reference/generated/numpy.unravel_index.html
- https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
- https://academic.mu.edu/phys/matthysd/web226/Lab02.htm
- Lecture notes by Dr. Tianfu Wu

****