CS 663 - Assignment 3

Shaan Ul Haque
180070053

Samarth Singh
180050090

Niraj Mahajan
180050069

October 2, 2020

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

Question 1

Boat Image

We used a 5*5 window for the calculation of the structure tensor at any point. The standard deviation(σ^2) used was 1.4. The value of k for cornerness measure was 0.04. While to smoothen the image we used a gaussian filter of standard deviation equal to 0.65. After all the processing we used non-max suppression to get the final cornerness measure. Results are shown below.

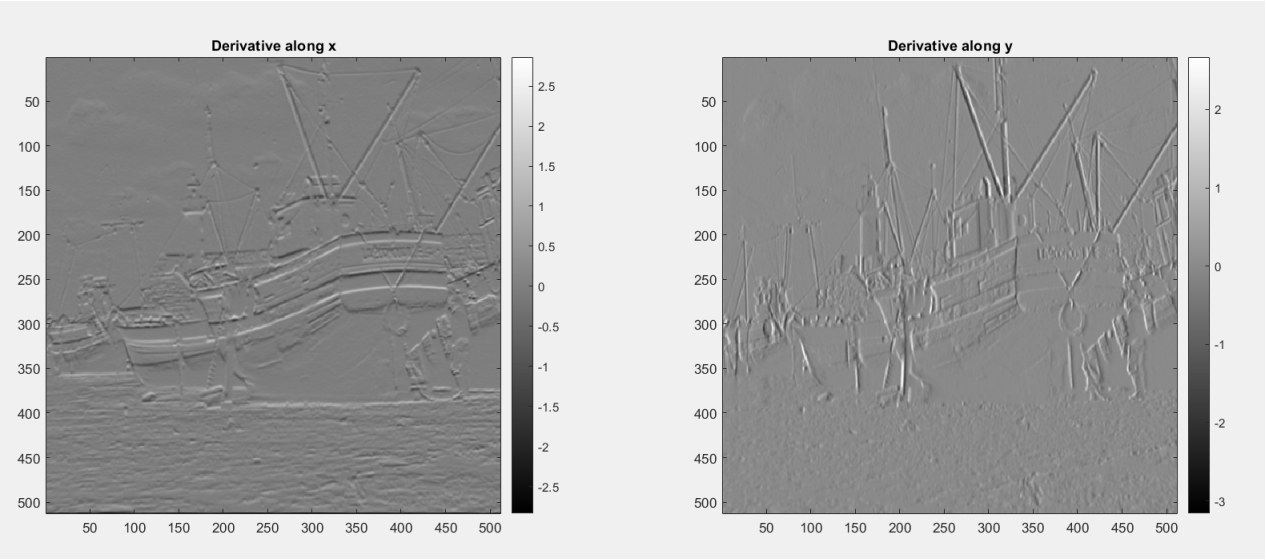

Figure 1: Derivative along X and Y

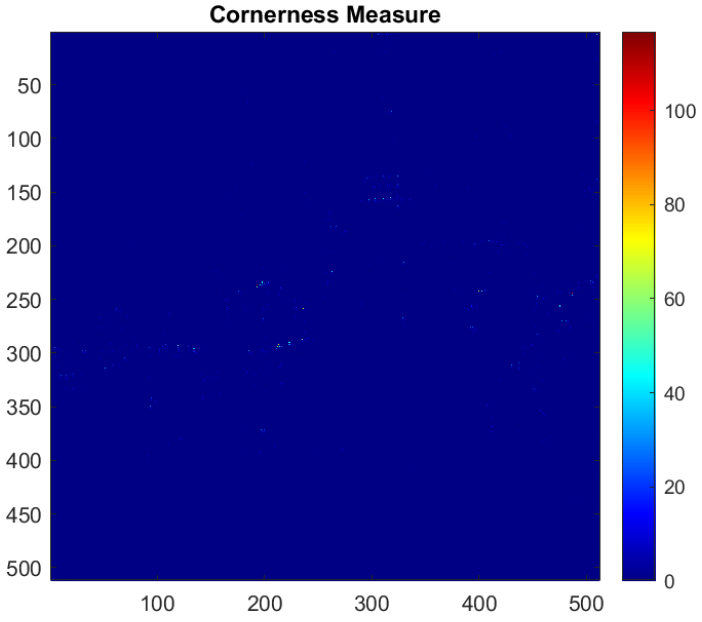

Figure 2: Cornerness measure

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

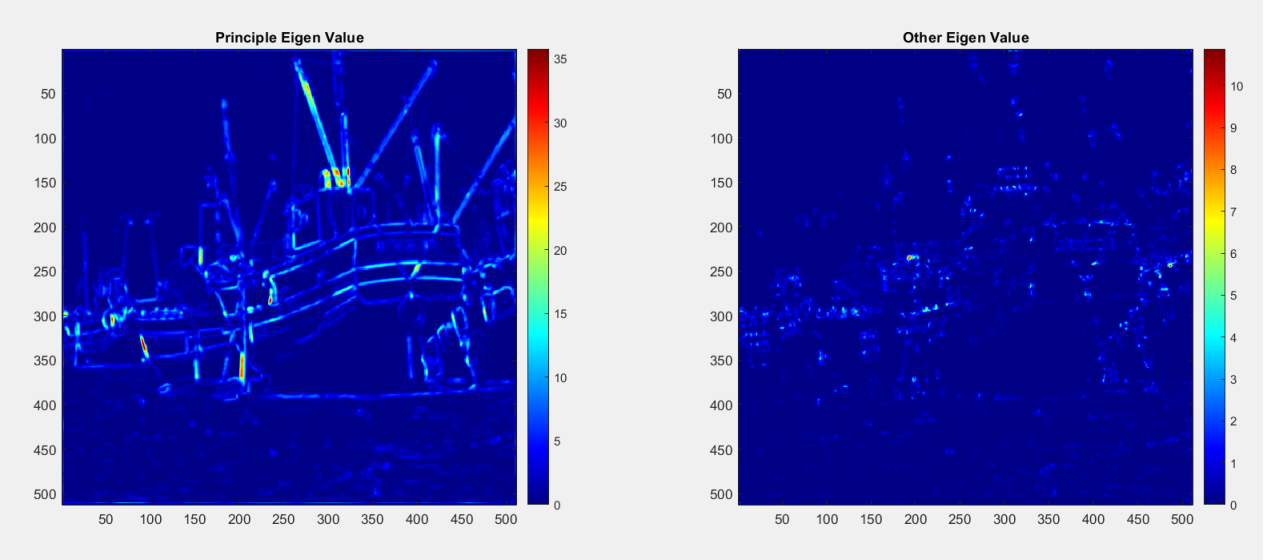

Figure 3: Eigen values

Code Usage

The main file myMainScript.m is divided into 2 parts.

- Firstly we use gaussian filter to smoothen the image. Then we use sobel filter to get the derivative along X and Y directions. Then we calculate the cornerness measure image using myHarrisCornerDetector.m function.
- The second part uses the technique of non-maximal suppression to get the final output.

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

Question 2

Part B:

Figure 2.1: Mean Shift Segmentation of **baboonColor.png**

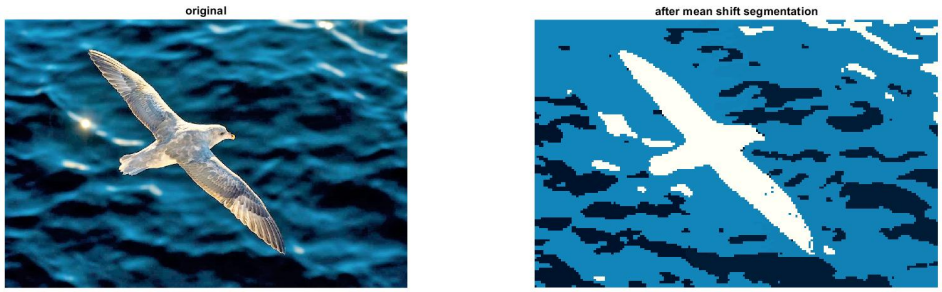

Figure 2.2: Mean Shift Segmentation of **bird.jpg**

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

Figure 2.3: Mean Shift Segmentation of **flower.jpg**

Part C:

Parameters:

- **baboonColor.png**:
  - i Gaussian Kernel bandwith for the color feature = 12
  - ii Gaussian Kernel bandwith for the spatial = 20
  - iii Number of iterations = 25
- **bird.png**:
  - i Gaussian Kernel bandwith for the color feature = 16
  - ii Gaussian Kernel bandwith for the spatial = 20
  - iii Number of iterations = 20
- **flower.png**:
  - i Gaussian Kernel bandwith for the color feature = 8
  - ii Gaussian Kernel bandwith for the spatial = 8
  - iii Number of iterations = 30

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

Question 3

3.1 : Approach to the problem

In this section, I have summarised the basic approach to the problem that I have used, along with the assumptions that I made. The essence of the problem is to separate the foreground from the background, given that the foreground is distinctively separable from the background.

Our basic approach is to generate bounding edges on the foreground using Canny's Edge Detector, and to flood fill the foreground within the detected edges. Then generate a radius matrix, (which has the smallest distance from a pixel to the foreground), and use this to perform spatially varying blurring. Here I have assumed that the foreground will be in the spatial centre of the image. Basically, we need a coordinate of our foreground to perform flood-filling. In a way, this is automatic, as our phone cameras also have a feature where we can tap on a person (in the camera app) and the camera focusses on him/her. The only input/effort that is need from us is to fine une the parameters of Canny Edge Detection. Another assumption that I made is that the foreground is a continuous blob of pixels. (More on this in Section 3.2)

3.2 : Forground Mask Generation

In this section, the exact algorithm to generate a foreground mask, given an input image is mentioned in detail, and the motivation behind all the sub parts and steps is justified.

3.2.1 Edge Detection

We perform a modified Canny Edge Detection of the input image, with the modification being, we do not apply Non Maximal Suppression. The catch here is that when we try to flood fill the image, even a tiny gap of a single pixel will flood the entire image. Hence we do away with Non Maximal Suppression, but at the cost of more noisy edges being introduced in the background.
Here are the fine tuned parameters that I used for both the images:

- **Flower:** $\sigma_g = 0.7$
- **Flower:** $\text{threshold}_{hi} = 0.315$
- **Flower:** $\text{threshold}_{lo} = 0.18$
- **Bird:** $\sigma_g = 0.7$
- **Bird:** $\text{threshold}_{hi} = 0.43$
- **Bird:** $\text{threshold}_{lo} = 0.25$

3.2.2 Performing Flood Filling on the foreground mask

Now that we have obtained the edge boundary of the input image, we want to perform a flood fill, so that we isolate the foreground from the background. It is necessary to ensure that the edges of foreground are closed, hence we keep the threshold$_{lo}$ of Canny Hysterisis Thresholding a bit lower than usual. The idea is, some extra noisy edges are acceptable, by the lack of an edge on the foreground isn't. This is, again the same reason why we did not perform Non Maximal Suppression while performing edge detection. We use matlab's $imfill$ function to perform flood filling (as this exploits multi threading of the cpu and is very fast). This flood filling is performed with the centre of the image as an initial point.

3.2.3 Removing Noisy Edges and Blobs

After the flood filling done in Section 3.2.2 we, obtain a noisy mask with many edges in the background, and some unnecessary flood filled blobs that extend to the background. In order to remove these, we perform median filtering on the image with a relatively large window size. Our assumption here is the most of the background will have 0s while most for the foreground will have 1s. So the outliers in the image which do no follow this trend get corrected by performing median filtering.

Now we have pure and continuous blobs of foreground and background. But we may have multiple blobs. For example, the mask of the flower has two blobs, one that covers the flower, and the other (smaller) that covers a part of a leaf. We need to remove blobs like these. For this we use matlab's inbuilt $bwareafilt$ function, which filters out blobs and preserves the one with the highest area.

3.2.3 Reclaiming the trimmed foreground

When we applied median filtering in section 3.2.2, some part of the foreground was lost, since the median at the peripheral points of the foreground will lie in the background. Note that the flood filling was performed on the foreground mask which is a matrix of logical bits. Hence in order to reclaim this lost/approximated area, we perform an operation quite similar to Non Maximal Suppression.

- Iterate over all the pixels P in the foreground mask, such that the value assigned to P is 0, ie, background
- Compute the 8-neighborhood of Pixel P
- If there exist any pixel that is assigned 1 in this neigh

We go over the foreground mask, and for every pixel that is assigned 0 (ie, background), we check its 8-neighborhood. If there lies a pixel that is assigned 1 (ie foreground) in its neighborhood, we set this pixel to 1.

...We finally have our foreground mask!

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

3.2.3 Results of the foreground generation algorithm

The following are the results obtained on **flower.jpg** and **bird.jpg**.

Figure 3.1(a): Original

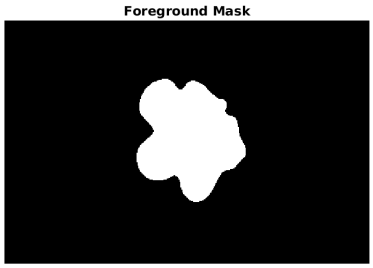

Figure 3.1(b): Mask

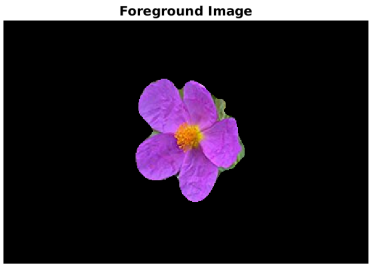

Figure 3.1(c): Foreground

Figure 3.1(d): Backgr.

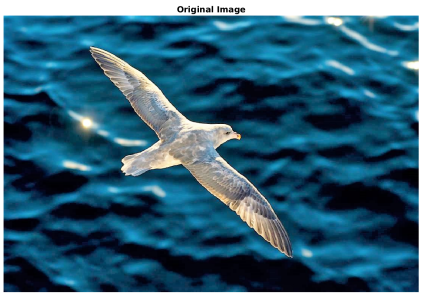

Figure 3.2(a): Original

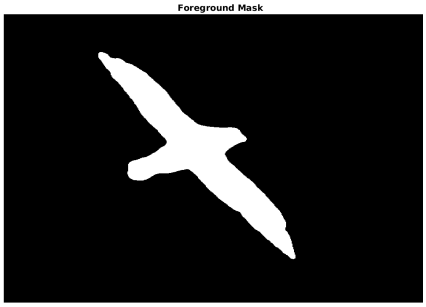

Figure 3.2(b): Mask

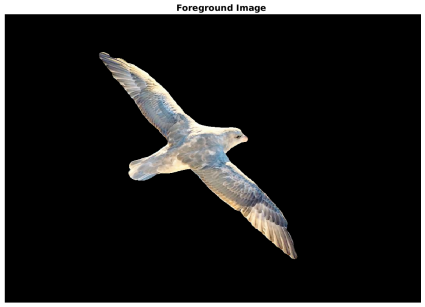

Figure 3.2(c): Foreground

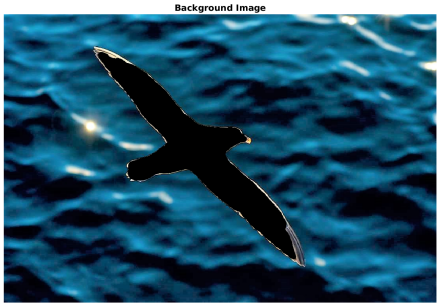

Figure 3.2(d): Backgr.

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

3.3 : Generating a Spatially Varying Kernel

Using the foreground mask that we obtained in Section 3.2, we now need to obtain a radii matrix $\mathbf{R} = [\, r_{ij} \,]$ such that $r_{i,j}$ at any pixel location i,j is the minimum distance from this pixel to any foreground pixel. We also need to cap these radii at a cutoff value (say α), where $\alpha_{flower} = 20$ and $\alpha_{bird} = 40$.

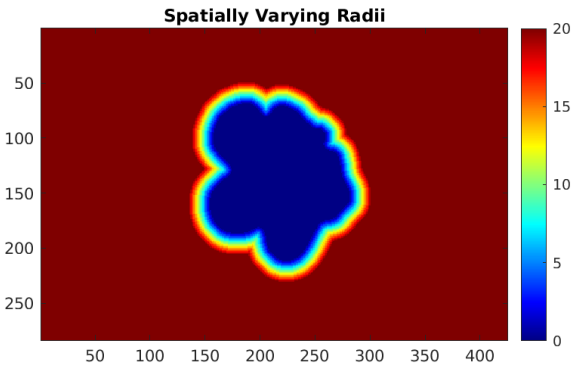

Figure 3.3(a): Flower Radii

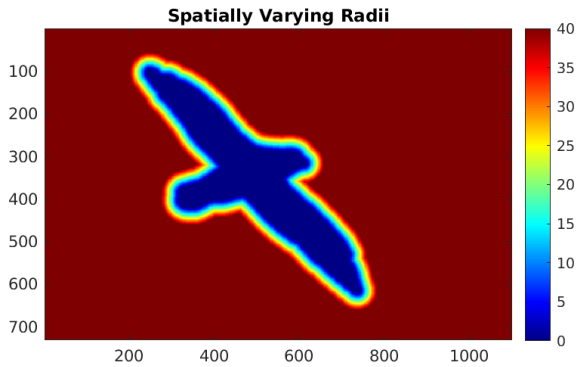

Figure 3.3(b): Bird Radii

3.4 : Generating Circular Kernel Filters

In order to perform background blurring, we need circular filters (of varying radius, of course). The corresponding filters at different radii, for flower.jpg and bird.jpg respectively are displayed below.

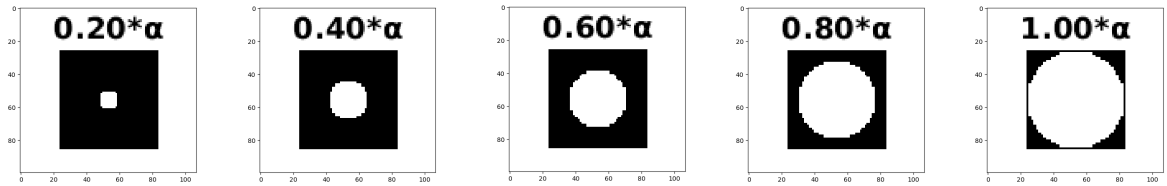

Figure : flower

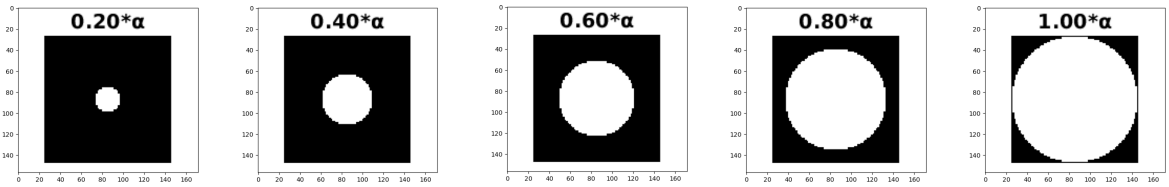

Figure : bird

Shaan Ul Haque(180070053)
Samarth Singh (180050090)
Niraj Mahajan (180050069)

3.5 : Final Results

Using the foreground maks obtained in Section 3.2, the radii matrices obtained in Section 3.3, and the Circular Filters obtained in Section 3.4, we now finally can perform background blurring on our input images. The results are as follows:

Figure 3.6(a): Original Flower

Figure 3.6(b): Blurred Flower

Figure 3.7(a): Original Bird

Figure 3.7(b): Blurred Bird

3.5 : Usage of Code

- The main file **myMainScript.m** takes about 170 seconds to execute - 6 seconds for flower.png and about 164 seconds for bird.png.
- The main script has all the parameters that are fine tuned for Canny Edge Detector. There is no other manual human input required, and rest of the process is completely automatic.
- The main script calls the **workOnImage.m** function, which appropriately processes each image and saves the outputs as .png
- The function **cannyEdgeDetector.m** implements the Canny algorithm, without Non Maximal Suppression (commented out).
- The function **getForeground.m** takes as input the original image and the edges obtained, and generates the foreground mask.
- The function **getKernel.m** generates a circular filter of a specified radius.
- The function **getRadiusMask.m** takes as input the foreground mask, and generates the radii matrix as specified in Section 3.3
- The function **mySpatiallyVaryingKernel.m** takes as input the foreground mask, input image, and radii matrix, and performs background blurring on the input image.