CE4003-COMPUTER VISION
LAB REPORT, AY 2015-16 Semester 1

CHAITANYA MALAVIYA, U1221016E

LAB 1

Experiment 2.1 : Contrast Stretching

Source Code:

```
%Part a:

>>Pc = imread('mrt-train.jpg');  %Pc stores the image
>>whos Pc;
>>P = rgb2gray(Pc);  %Conversion from rgb to grayscale


%Part b:

>>imshow(P)


%Part c:

>>min(P(:))

>>max(P(:))


%Part d:


>>J=immultiply(imsubtract(P,double(min(P(:)))),(255/double(max(P(:))-
>>min(P(:)))));  %Code for contrast stretching

>>min(J(:)), max(J(:));  %Finding min and max intensity of image after
contrast stretching

%Part e:


>>imshow(J);
```

Outputs:

```
>> whos Pc;
  Name         Size                Bytes  Class     Attributes

  Pc         320x443x3           425280  uint8

>> min(P(:))

ans =

   13

>> max(P(:))

ans =

  204

>> J=immultiply(imsubtract(P,double(min(P(:)))),(255/double(max(P(:))-min(P(:)))));
>> min(J(:))

ans =

    0

>> max(J(:))

ans =

  255
```

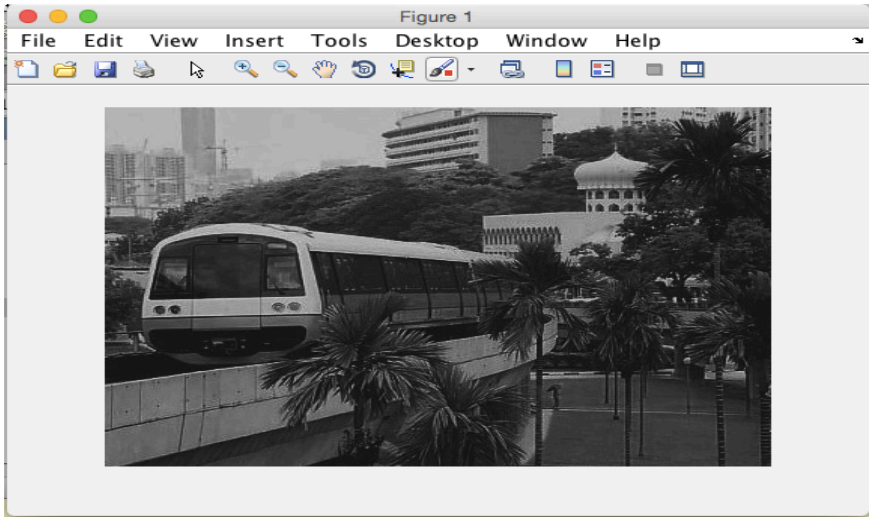

Figure 1: Image in grayscale mode

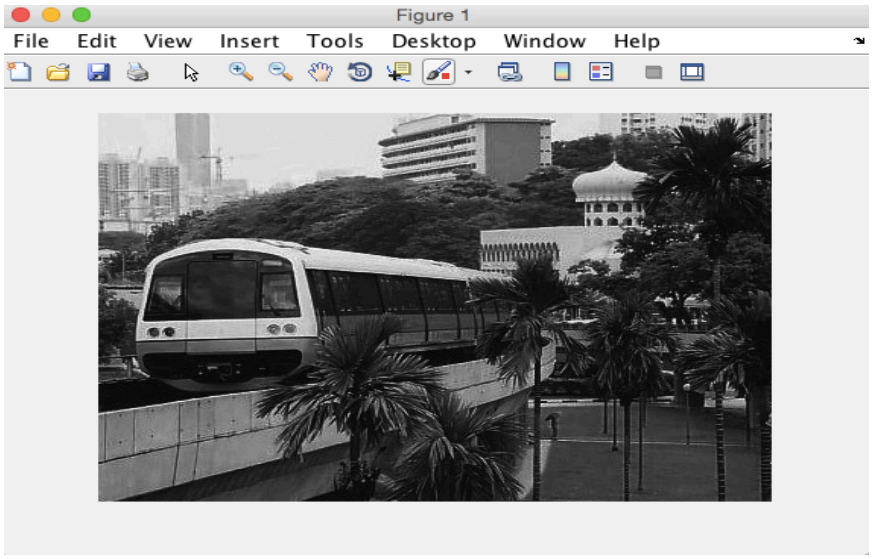

Figure 2: Image after contrast stretching

Experiment 2.2 : Histogram Equalization

Source Code:

```
%Part a:

>>imhist(P,10);

>>imhist(P,256);

%Part b:

>>P3 = histeq(P,255);

>>imhist(P3,10);

>>imhist(P3,256);

%Part c:

>>P4 = histeq(P3,255);

>>imhist(P4,10);

>>imhist(P4,256);
```

Outputs and Explanations:

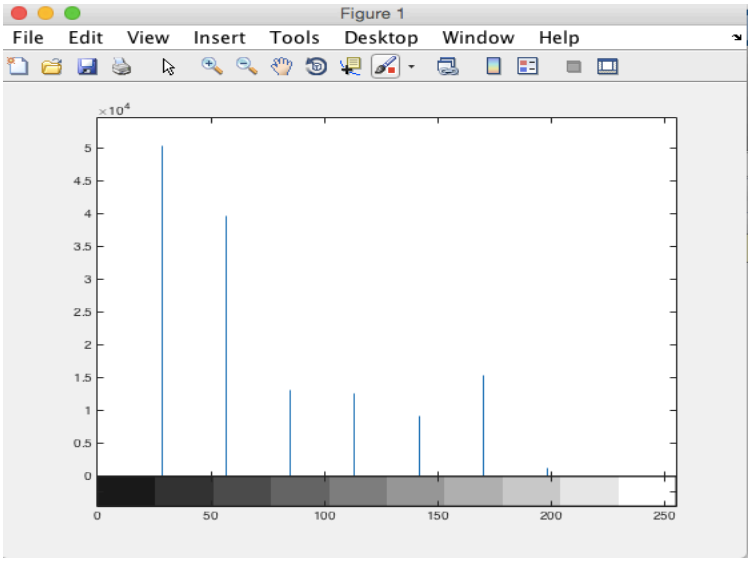

Figure 3 : Original Histogram with 10 bins

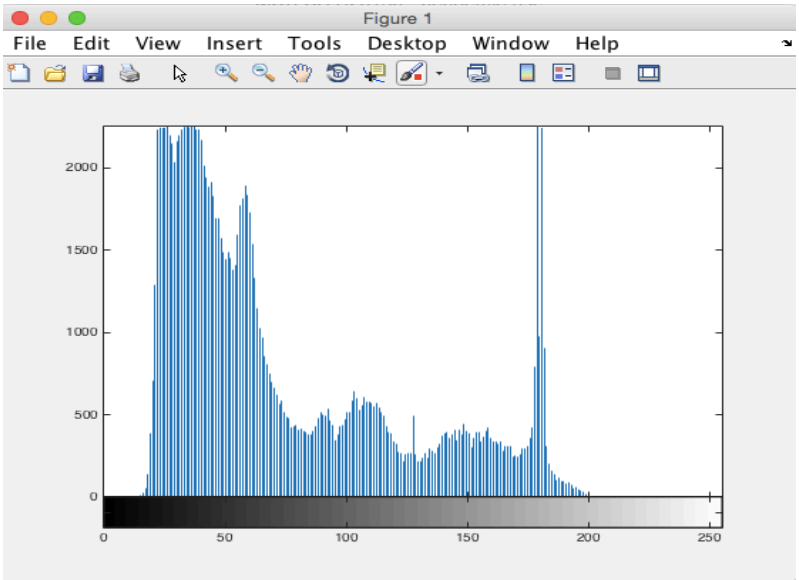

Figure 4 : Original Histogram with 255 bins

Difference between the 2 histograms:

The histogram for an image plots the number of pixels that belong to different intensity values of the image, which can range from 0 to 255. When we choose n bins to plot the histogram, this range of intensity values is divided into n blocks. In other words, the number of bins used is the level of quantization of intensity values. The range of intensities that each bin covers increases as the number of bins increase.

Therefore, the histogram with 10 bins divides the number of image pixels into intensity ranges of 0-24, 25-49, and so on. Whereas, the histogram divides the pixels into intensity ranges of single values 0, 1, 2, 3..255 each. Since the intensity range of each bin in the 10-bin histogram is larger, more pixels fall into each interval whereas lesser pixels fall into each interval in the 255-bin histogram.

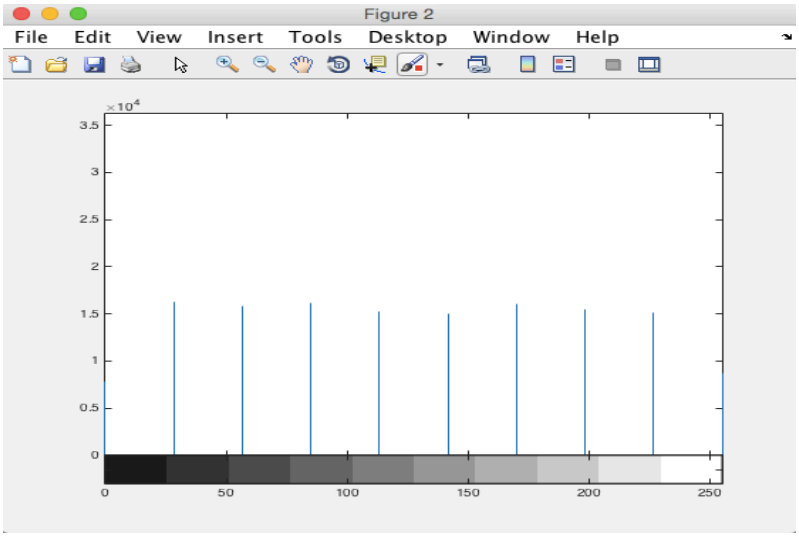

Figure 5 : Image histogram (10 bins) after histogram equalization

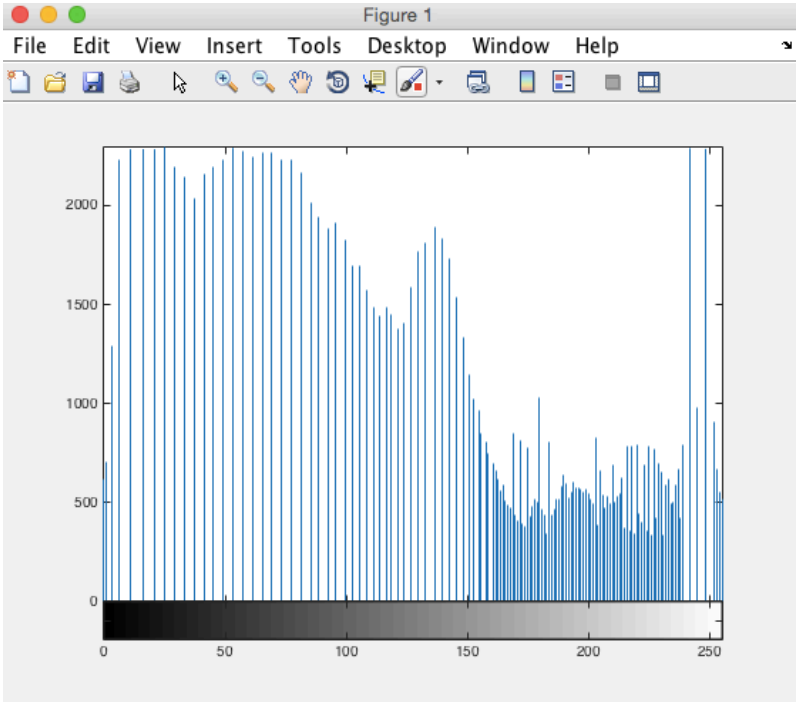

Figure 6 : Image histogram (10 bins) after histogram equalization

Yes, the histograms are equalized but not exactly flattened. In the 10 bin histogram, the intensity ranges have considerably flattened but still not exactly uniform. After equalization, many pixels have moved to the middle and high intensity range, stretching out the intensity range.

Similarity: It can be seen that in both the histograms, the most frequent intensity values are more spread out throughout the histogram after equalization. In doing so, the global contrast of the image is enhanced.

Difference: The 10-bin histogram is much flatter than the 255-bin histogram because it plots the number of pixels for a wider range of intensity values. From the 255-bin histogram, it can be seen that less number of pixels but with short intervals are present in the high intensity range while high number of pixels with long intervals are present in the low intensity range. This results in a relatively flatter histogram view in the 10-bin histogram.

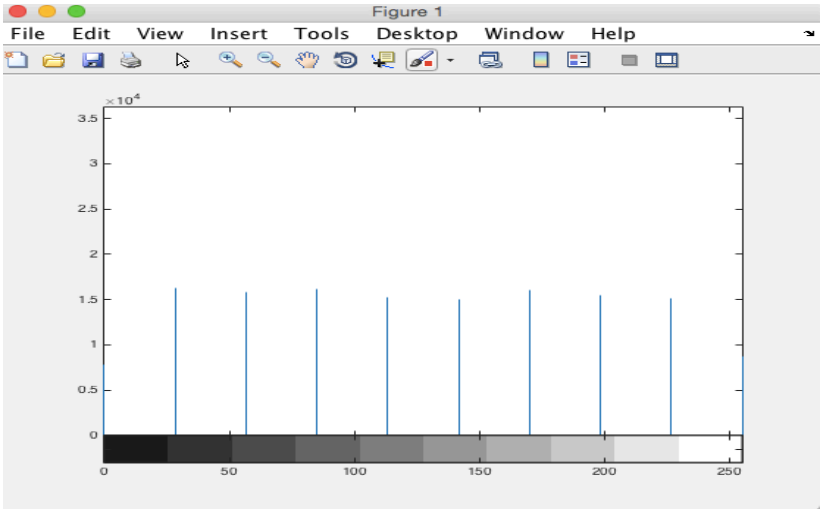

Figure 7 : 10-bin histogram after equalizing again

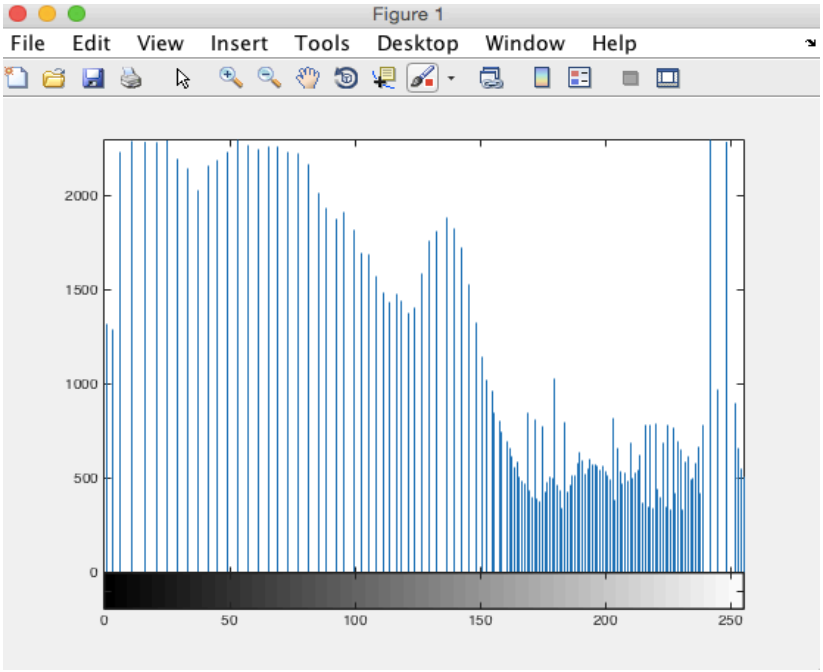

Figure 8 : 255-bin histogram after equalizing again

After we rerun the histogram equalization, the histogram does not become any more uniform. In fact, there is **no change to the histogram**. To explain why, lets understand how histogram equalization works. The process involves mapping the original histogram distribution to another distribution, that is more uniform and more spread out. For a continuous distribution, this remapping is done by a cumulative distribution function, given by, $H'(i)= \sum H(j)$, where j runs from 0 to i, for a histogram given by H(i). We then normalize H'(i) from 0-255 to cover the entire range of intensity values. If the histogram H is regarded as a continuous probability density function giving distribution of the intensity levels, the equalized probability densities are an equal fraction of the maximum number of intensity levels in the image. Thus, an image that is transformed using its cumulative distributive function would give an output distribution that is flat. Reapplying the equalization would make no difference as the cumulative distribution function would have no effect on the histogram, as the distribution is already flattened.

Experiment 2.3 : Linear Spatial Filtering

Source Code:

```
%Part a:

sigma1 = 1.0;
sigma2 = 2.0;
[x y] = meshgrid(-2:2,-2:2);
F1=(1/(2*pi*sigma1^2))*exp(-((x.^2+y.^2)/(2*sigma1^2))); %Creation
of filter1
F1=F1./sum(F1(:));%Normalization
F2=(1/(2*pi*sigma2^2))*exp(-((x.^2+y.^2)/(2*sigma2^2))); %Creation
of filter2
F2=F2./sum(F2(:));
figure('name','Filter 1');
mesh(F1);
figure('name','Filter 2');
mesh(F2);

%Part b:

I = imread('ntugn.jpg');
figure('name','Original Image with Gaussian Noise');
imshow(I);

%Part c:

G1 = uint8(conv2(I,F1)); %Convolution with original image
figure('name','Image1 after using Filter 1');
imshow(G1);
G2 = uint8(conv2(I,F2));
figure('name','Image1 after using Filter 2');
imshow(G2);


%Part d:

J = imread('ntusp.jpg');
figure('name','Original Image with Speckle Noise');
imshow(J);

%Part e:

H1 = uint8(conv2(J,F1)); %Convolution with original image
figure('name','Image2 after using Filter 1');
imshow(H1);
H2 = uint8(conv2(J,F2));
figure('name','Image2 after using Filter 2');
imshow(H2);
```

Outputs and Explanations:

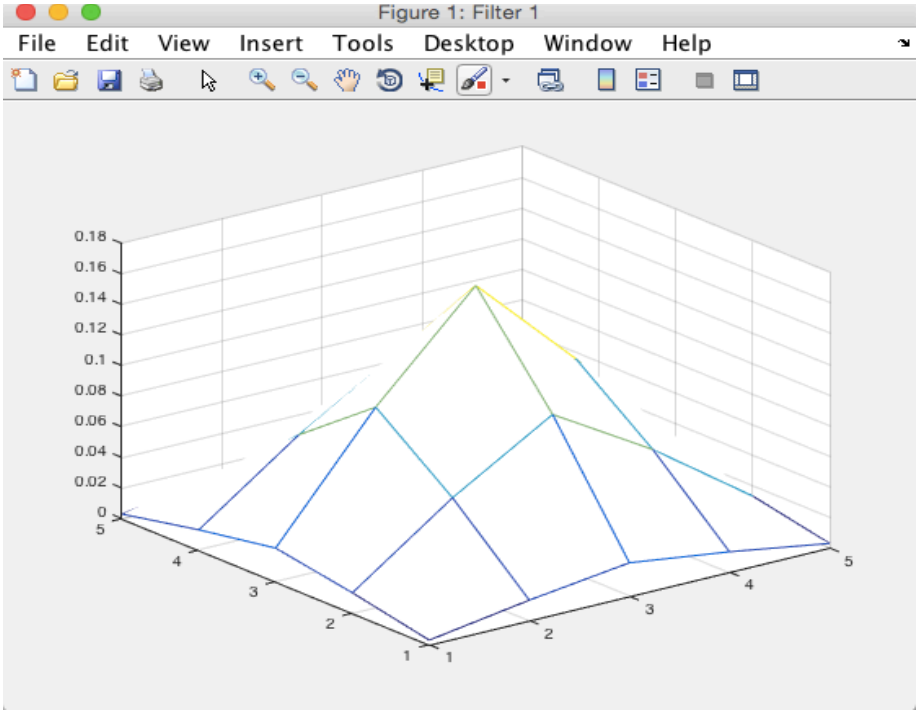

Figure 9 : 3D Mesh of Filter 1 with Sigma=1.0

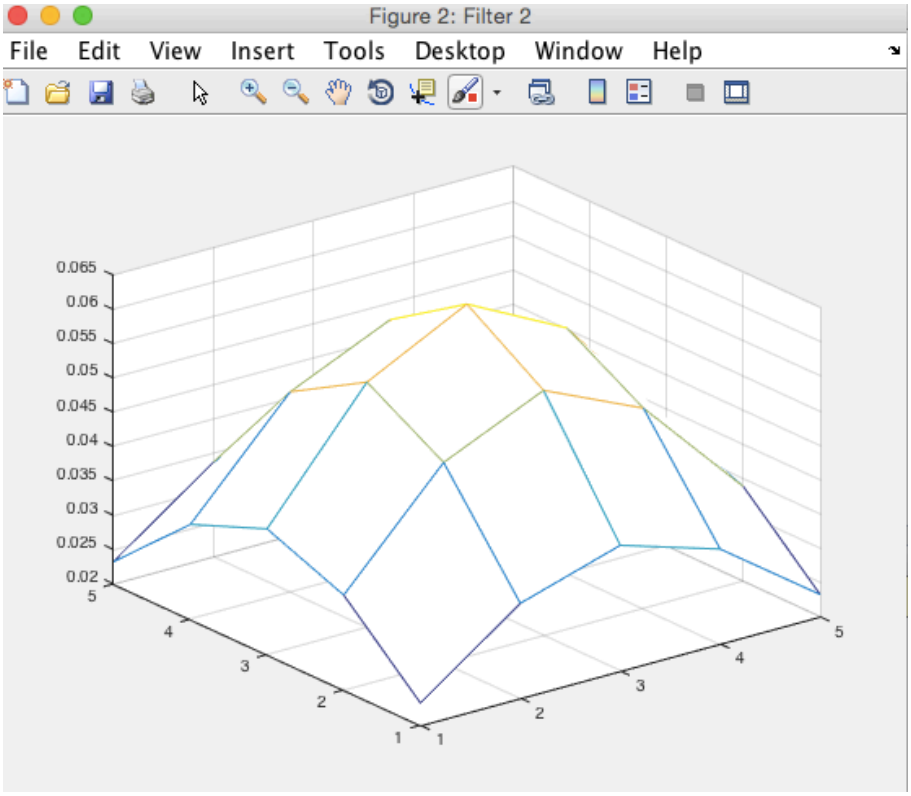

Figure 10 : 3D Mesh of Filter 2 with Sigma=2.0

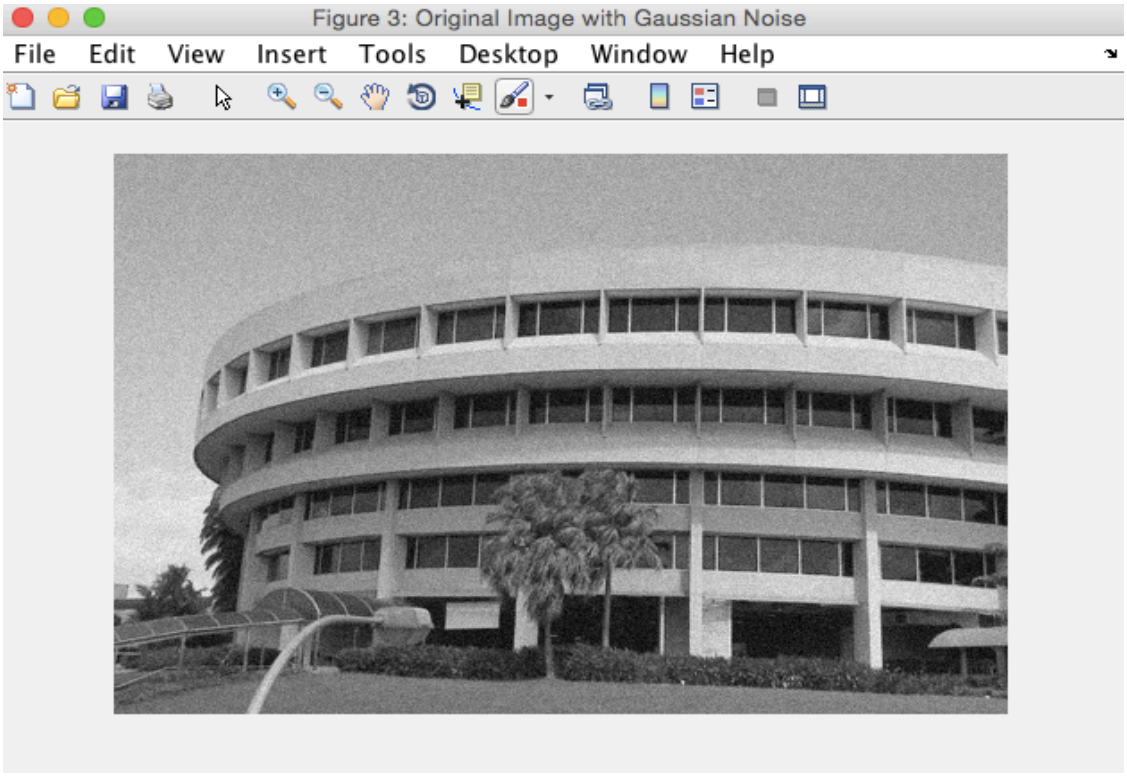

Figure 11 : Original Image with additive white gaussian noise

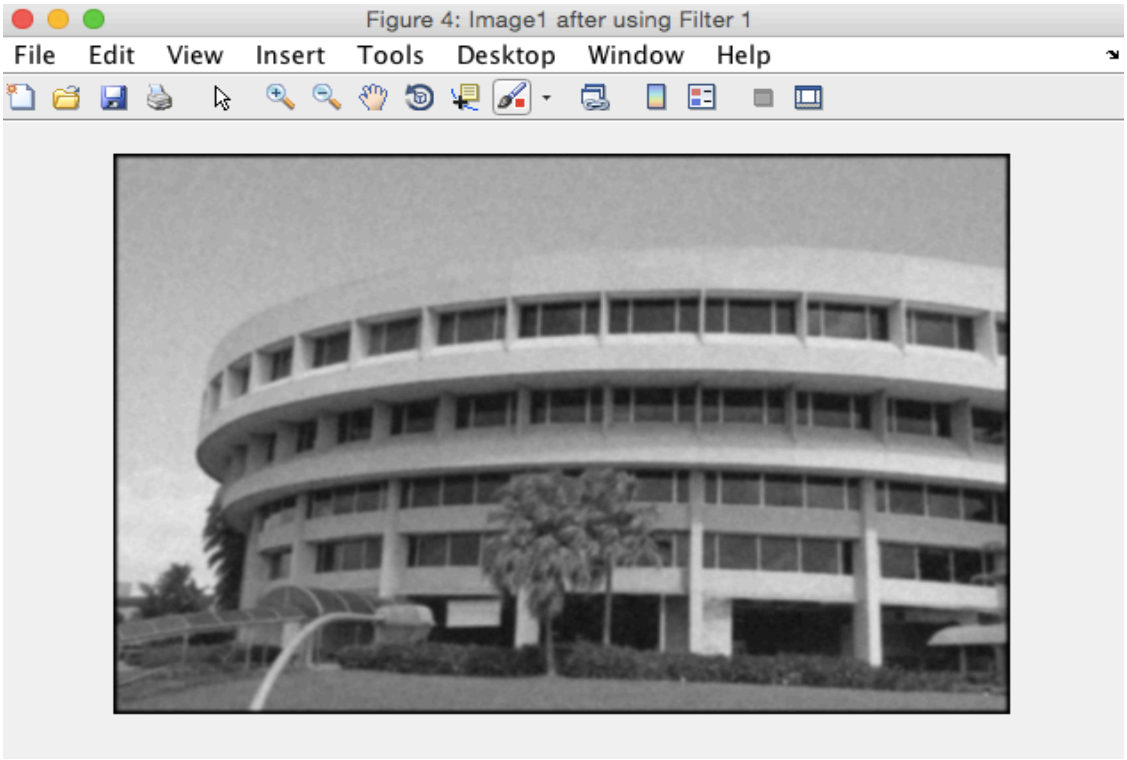

Figure 12 : Image1 filtered using Filter1

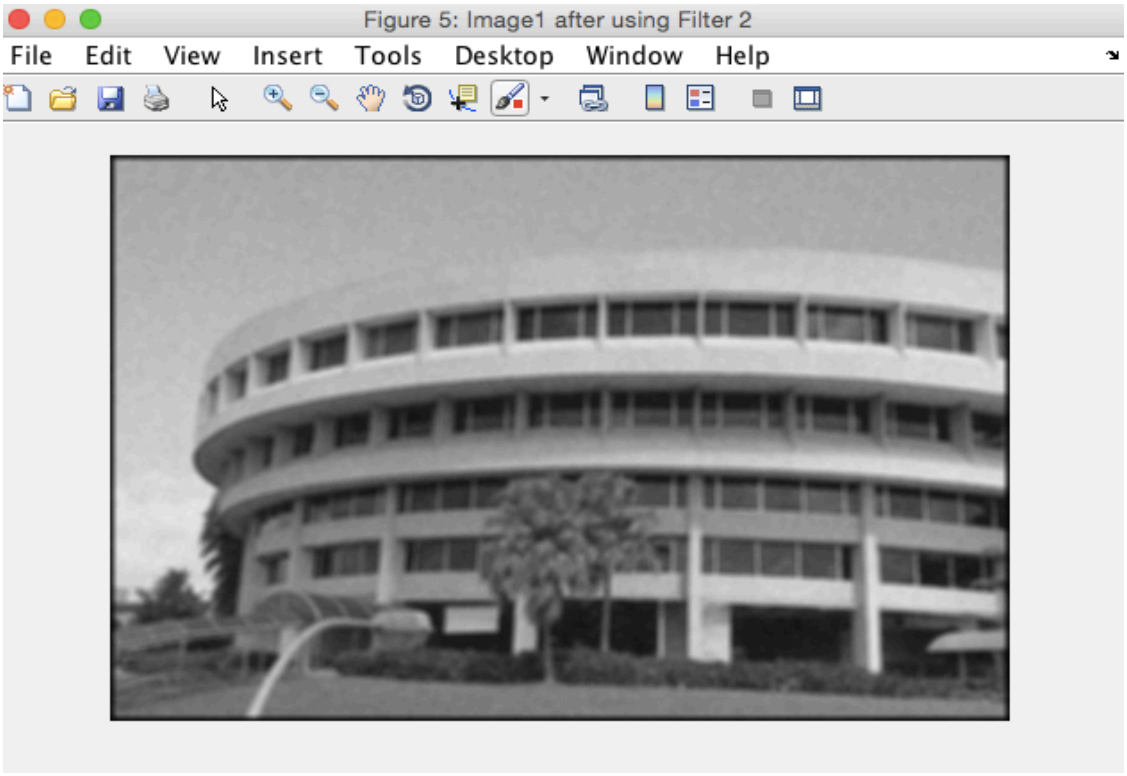

Figure 13 : Image1 filtered using Filter2

From the above output , we can notice that both the filters remove Gaussian noise to a certain extent but they also cause blurring of the image.

Using filter1 with sigma=1.0 reduces the Gaussian noise in the image, but filter2 with sigma=2.0 gives even better results in removing Gaussian noise. However, using filter2 also causes a lot of blurring of the image. This is because higher value of sigma reduces Gaussian noise but increases blurring of the image.

The tradeoff we observe here is that using filter1 reduces Gaussian noise to a lesser level than filter2 but introduces lesser blurring of the image than filter2. Not using a filter at all would let the additive Gaussian noise to be retained by the image. Here, filter1 proves to be a good balance in reducing Gaussian noise and also not blurring the resultant image too much.

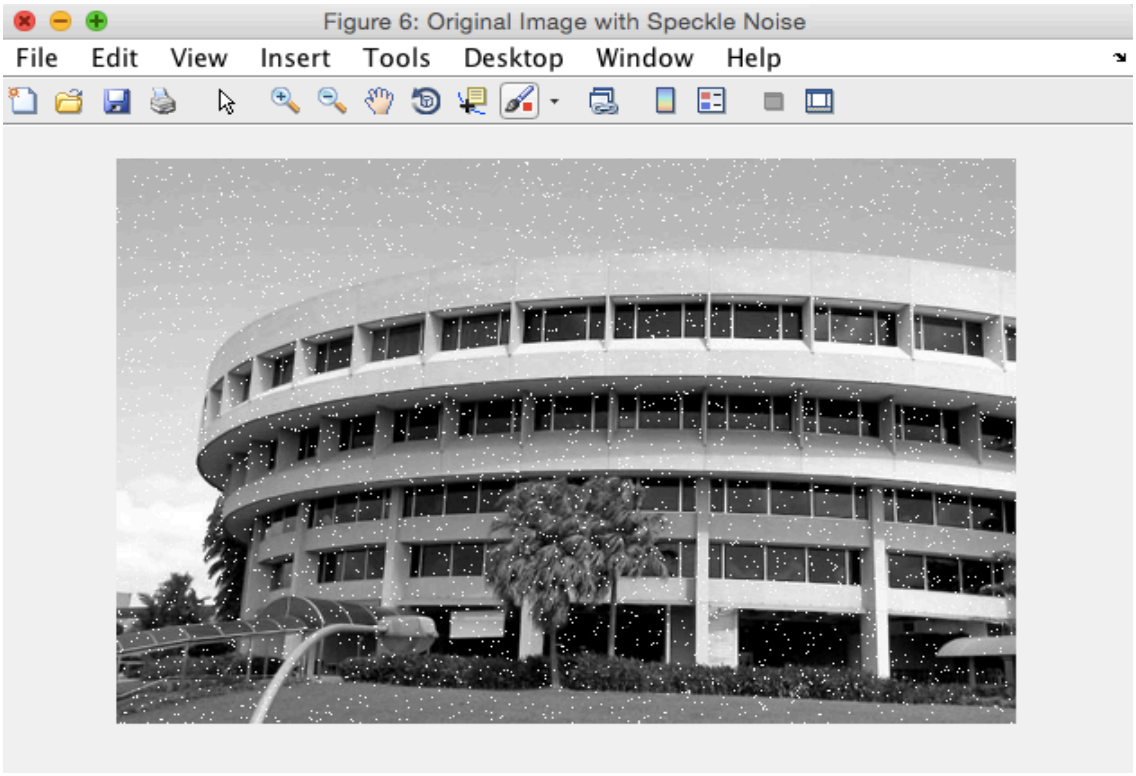

Figure 14 : Original Image with speckle noise

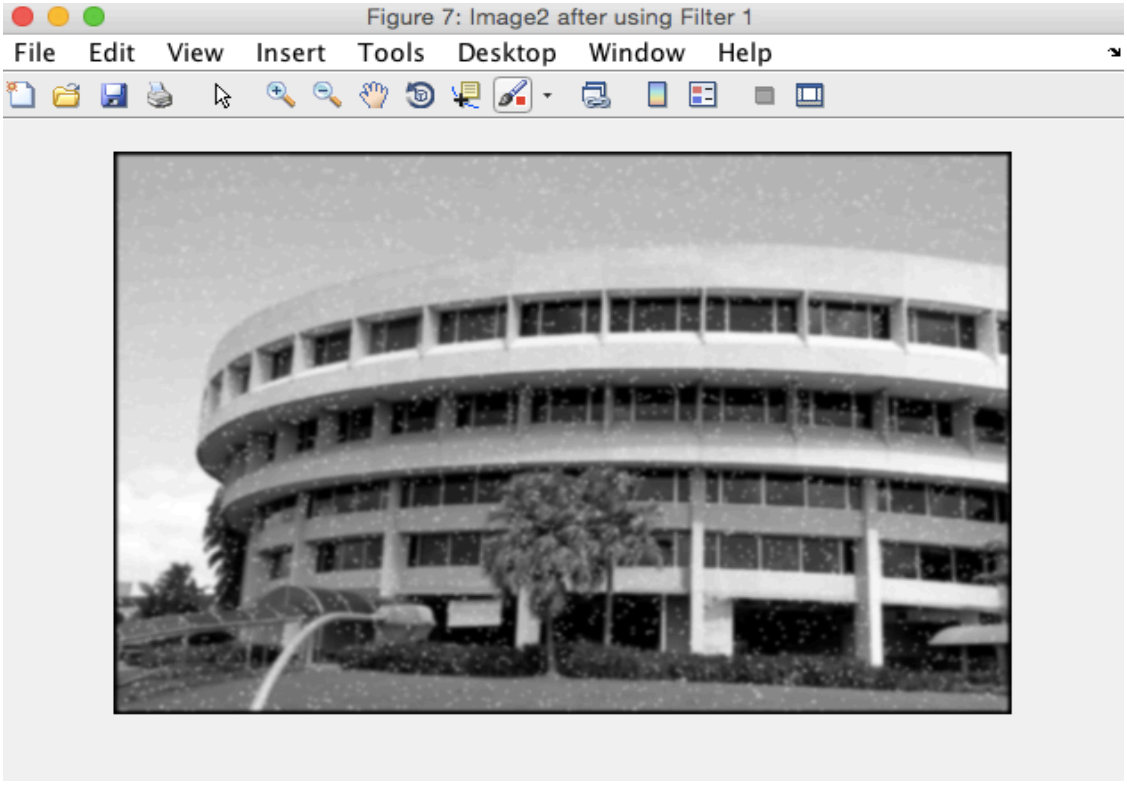

Figure 15 : Image2 filtered using Filter1

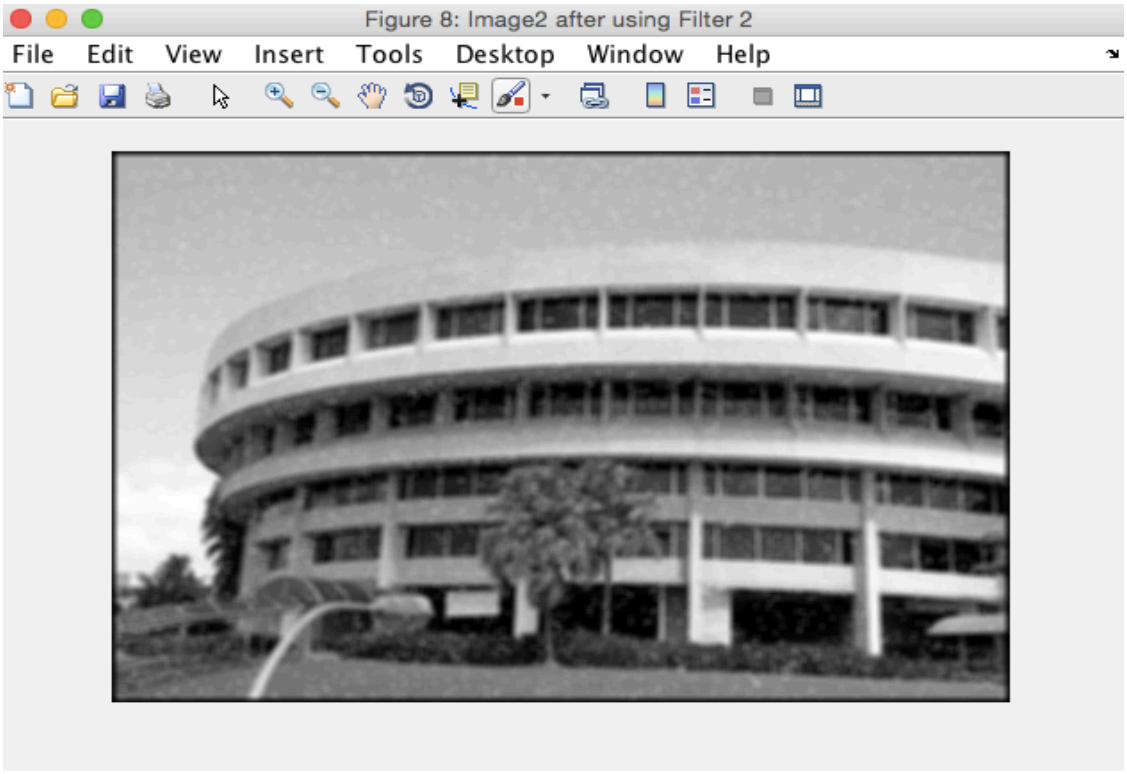

Figure 16 : Image2 filtered using Filter2

From the above images, it can be seen that both the filters reduce speckle noise from the image to a certain extent but not completely. Filter2 introduces blurring of this image due to higher sigma as it did with the image with Gaussian noise.

We can observe that Filter1 showed better results in reducing Gaussian noise than speckle noise. Similarly, although Filter2 introduced slight blurring of the image with Gaussian noise, it gives a much better result than filtering the image with speckle noise.

Hence, we can conclude that the filters are relatively better at handling Gaussian noise.

Experiment 2.4 : Median Filtering

Source Code:

```
I = imread('ntugn.jpg');

M1 = medfilt2(I,[3 3]); %Median filtering with neighbourhood [3 3]
figure('name','Image1 after using Median Filter[3 3]');
imshow(M1);

M2 = medfilt2(I,[5 5]); %Median filtering with neighbourhood [5 5]
figure('name','Image1 after using Median Filter[5 5]');
imshow(M2);


J = imread('ntusp.jpg');

N1= medfilt2(J,[3 3]); %Median filtering with neighbourhood [3 3]
figure('name','Image2 after using Median Filter[3 3]');
imshow(N1);

N2 = medfilt2(J,[5 5]); %Median filtering with neighbourhood [5 5]
figure('name','Image2 after using Median Filter[5 5]');
imshow(N2);
```

Outputs and Explanations:

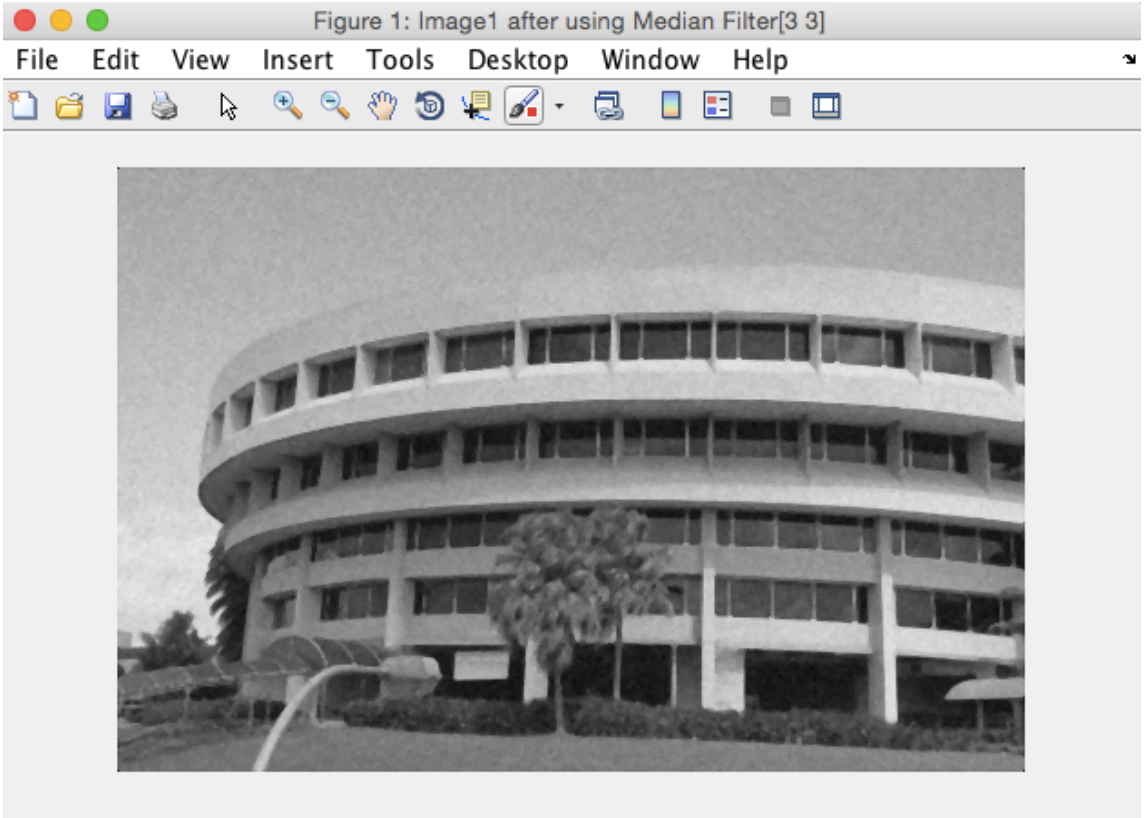

Figure 17 : Image with Gaussian noise filtered using Median Filter [3 3]

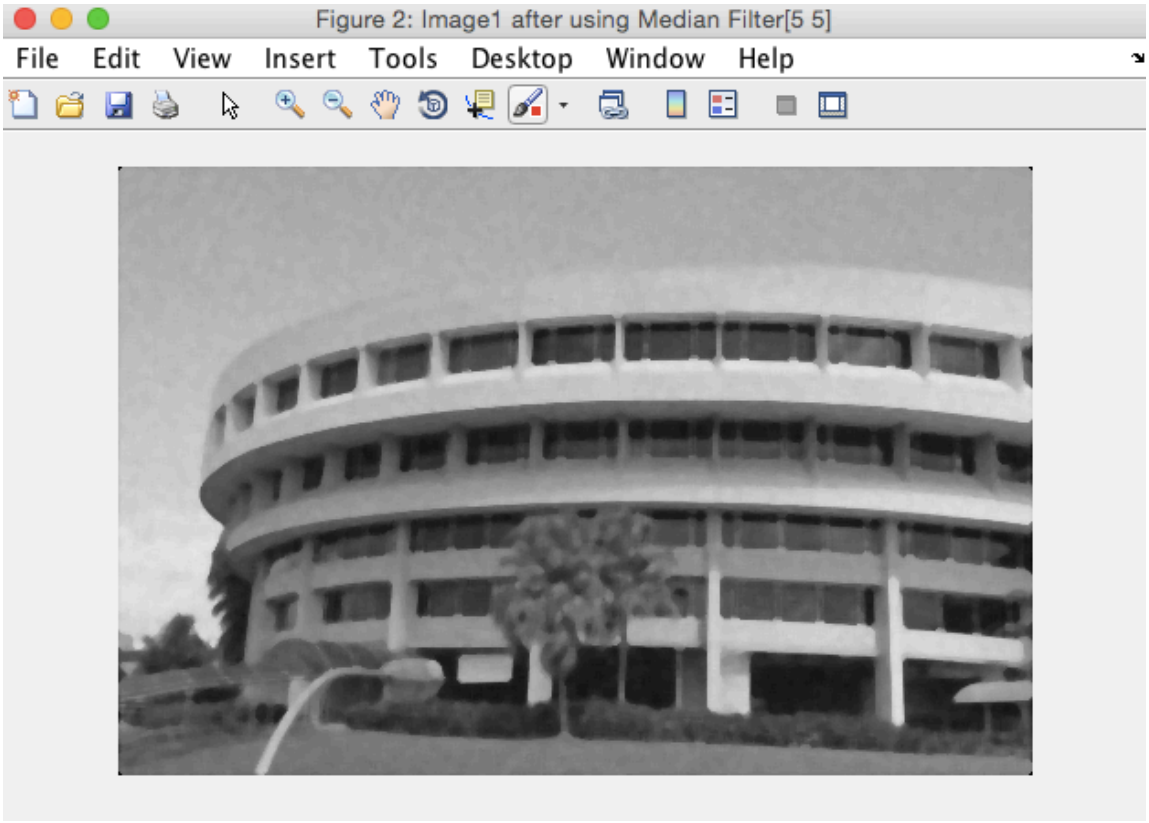

Figure 18 : Image with Gaussian noise filtered using Median Filter [5 5]

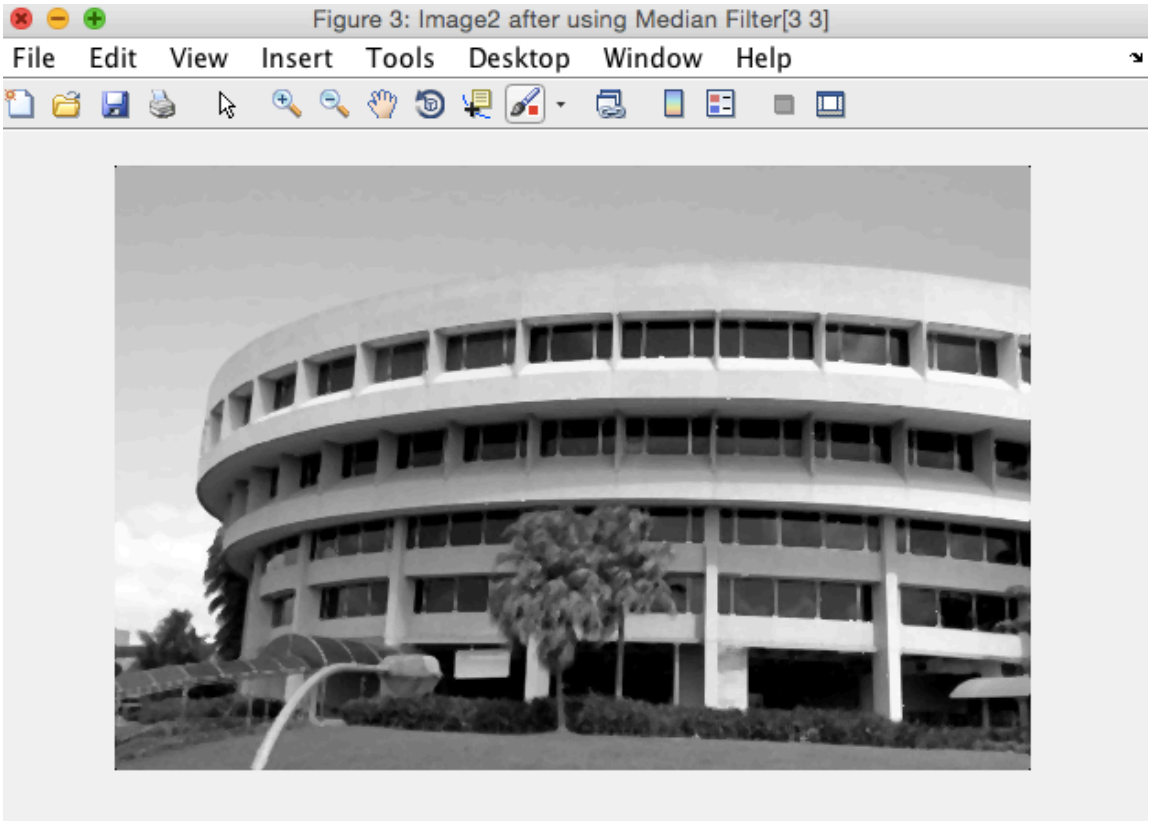

Figure 19 : Image with speckle noise filtered using Median Filter [3 3]

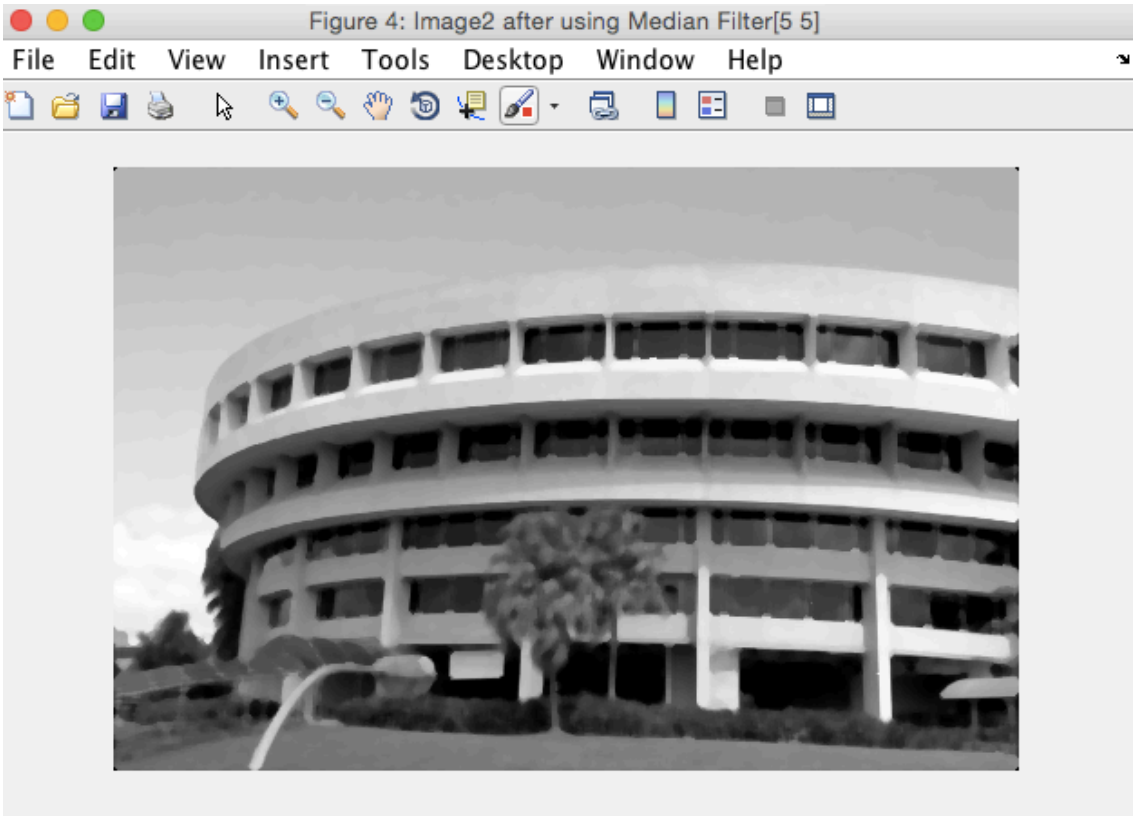

Figure 20 : Image with speckle noise filtered using Median Filter [5 5]

Results:

- As can be seen from the output, using a median filter with a larger neighbourhood size [5 5] results in blurring of the image and makes the image look like a painting.
- Comparing the median-filtered images for Gaussian noise and speckle noise, we can observe that the median filter gives better results for additive speckle noise for both the neighbourhood sizes. The median filter is more robust to impulse noise.
- Whereas, images with additive Gaussian noise filtered with Gaussian filter give slightly better results than median filters.

Tradeoff:

- Using a median filter with a larger neighbourhood removes greater amount of noise but also introduces blurring/painting-like effect in the resultant image.

We can conclude from our results that the Gaussian filter of both values of sigma performs better than the median filter for removing additive Gaussian noise. Whereas, the median filter of both neighborhood sizes performs better than the Gaussian filters for removing additive speckle noise.

Experiment 2.5 : Suppressing Noise Interference Patterns

I. Source Code:

```
%Part a:

I=imread('pckint.jpg');
figure('name', 'Image with interference');
imshow(I);

%Part b:

ID = im2double(I);
FID = fft2(ID);
S=abs(FID).^2;
figure('name', 'Power spectrum with fftshift');
imagesc(fftshift(S.^0.1));
colormap('default');

%Part c:

figure('name', 'Power spectrum without fftshift');
imagesc(S.^0.1);
colormap('default');

%The two peaks of the interference were noted from the power
spectrum of the image.

%Part d:

FID(239:243,7:11) = 0;
FID(15:19,247:251) = 0; %The 5X5 neighborhood elements of the
peaks are set to 0 here
S1=abs(FID).^2;
colormap('default');
figure('name','Power spectrum with fftshift after removing
interference');
imagesc(fftshift(S1.^0.1));

%Part e:


IFID = ifft2(FID);
NewImg = im2uint8(IFID);
figure('name','Resulting Image after removing interference');
imshow(NewImg)

%Further Improvements by setting 15X15 neighborhood elements to 0:


FID(234:248,2:16) = 0;
FID(10:24,242:256) = 0;
%The 15X15 neighborhood elements of the peaks are set to 0 here
```

```
S2=abs(FID).^2;
colormap('default');
figure('name','Power spectrum with fftshift with improvement');
imagesc(fftshift(S2.^0.1));

IFID = ifft2(FID);
NewImg2 = im2uint8(IFID);
figure('name','Improvements');
imshow(NewImg2);
```

Outputs and Explanations:

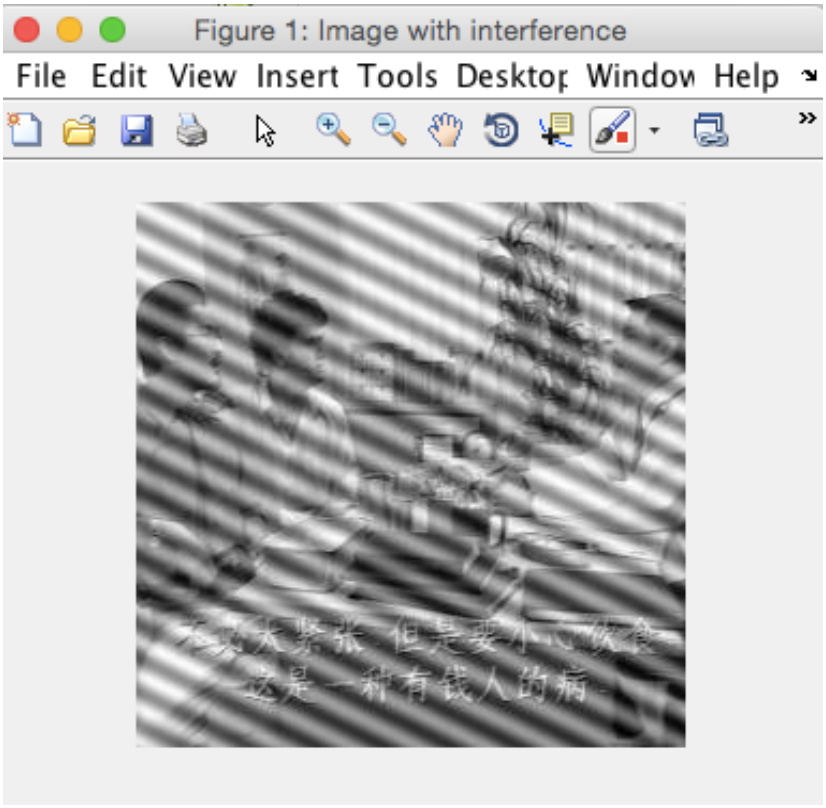

Figure 21 : Original Image with interference

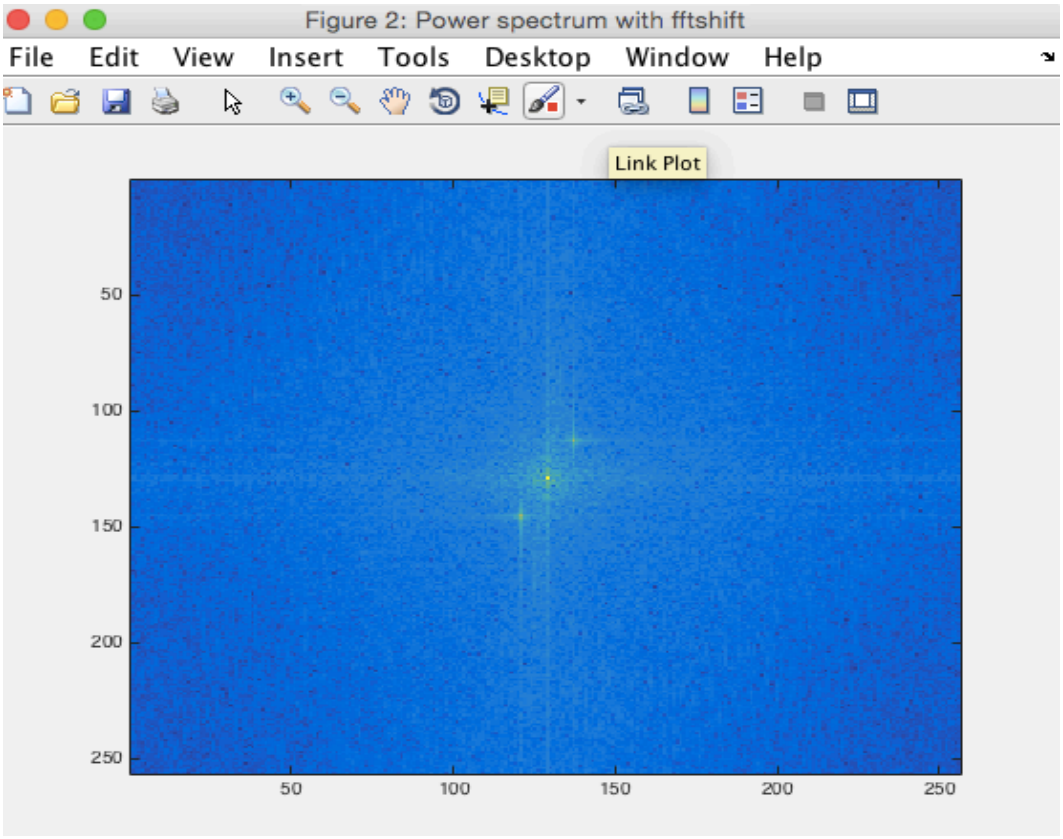

Figure 22 : Power spectrum of image with fftshift

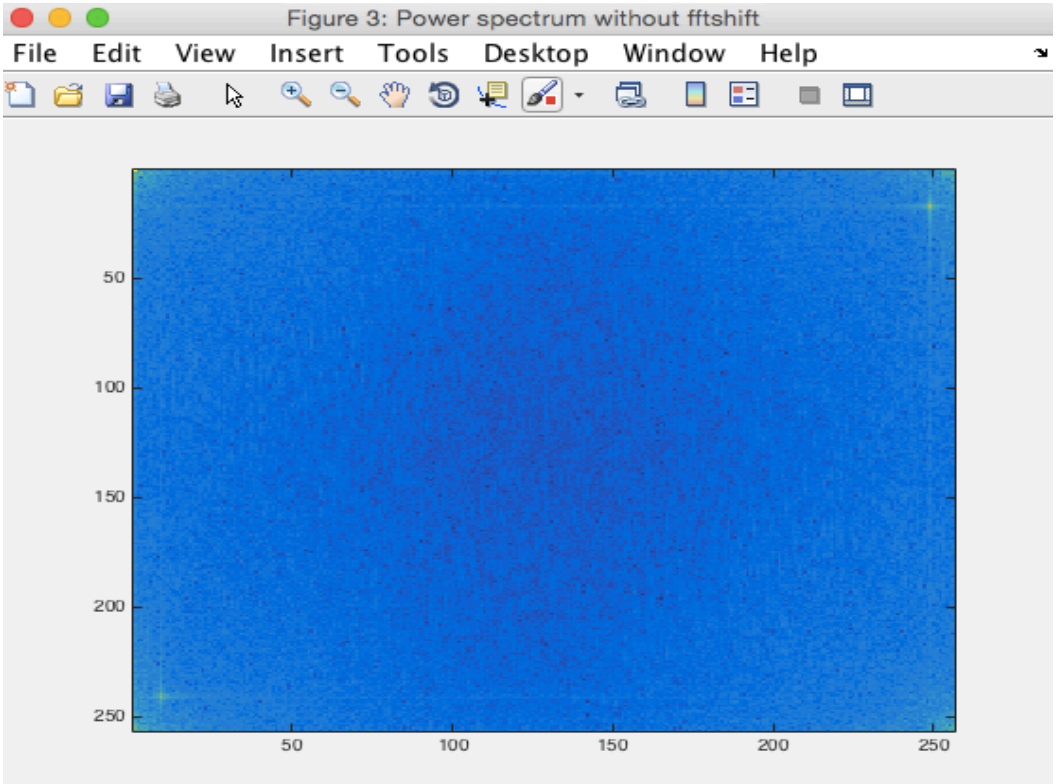

Figure 23 : Power spectrum of image without fftshift

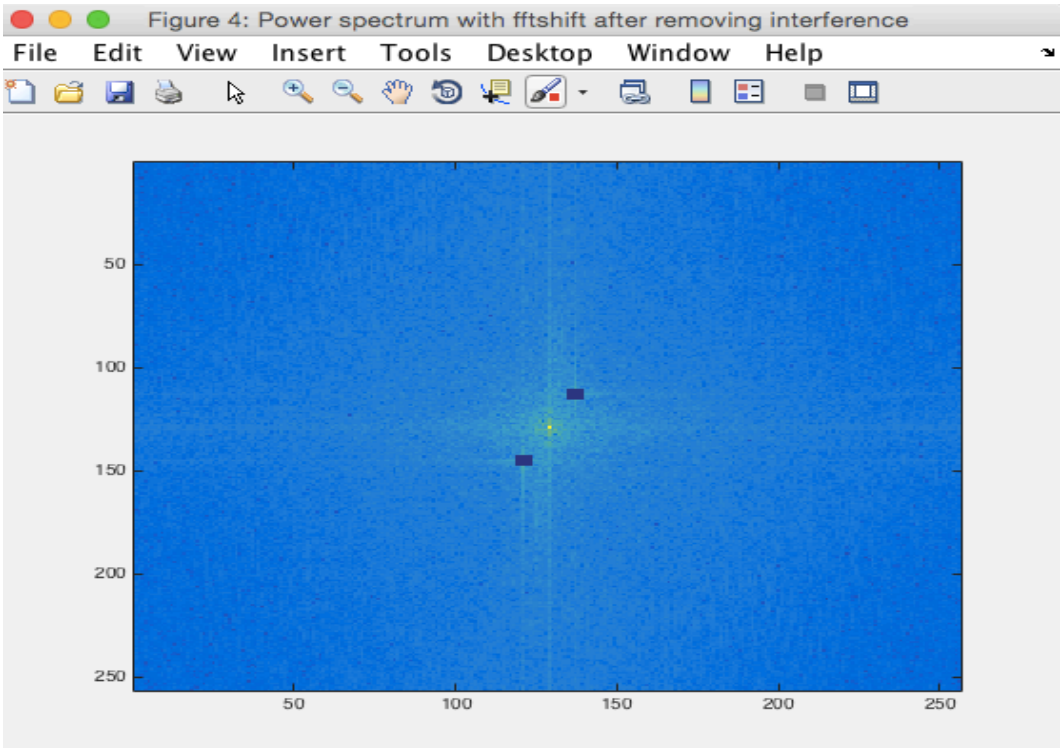

Figure 24 : Power Spectrum after setting elements in 5X5 neighbourhood to zero

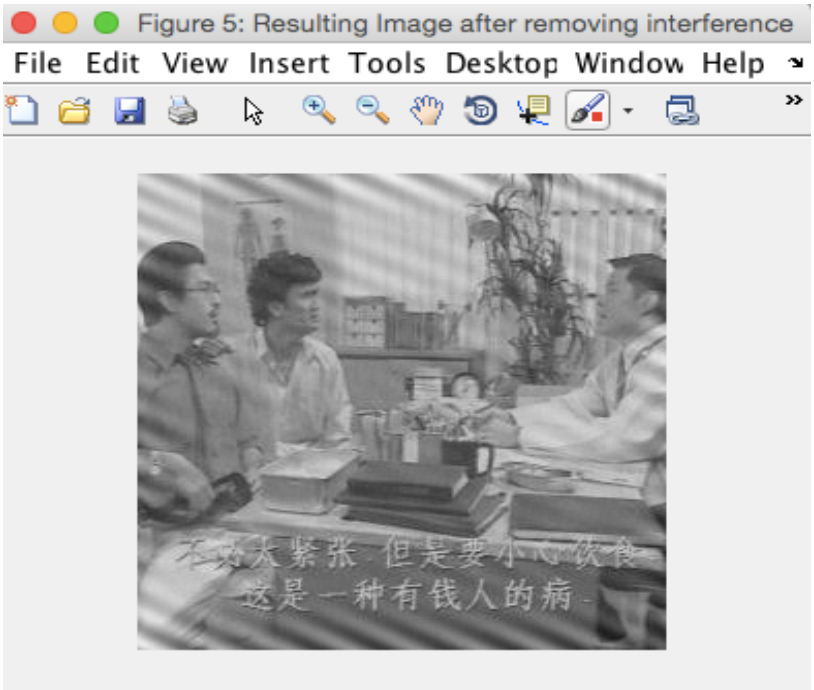

Figure 25 : Resulting Image with reduced interference

- We can observe that the coordinates in a 5X5 neighborhood of the two frequency peaks contributing to interference are set to zero. By doing this, we are suppressing the layers which contribute most to the noise interference pattern. The expected result removes majority of the central interference but there are still some diagonal lines on the borders of the image. Although we cannot achieve a perfect result, we can try to make some improvements to the result.

- **Suggested Improvement** : We can try to improve our result by setting the 15X15 neighbourhood elements around the two peaks to zero.

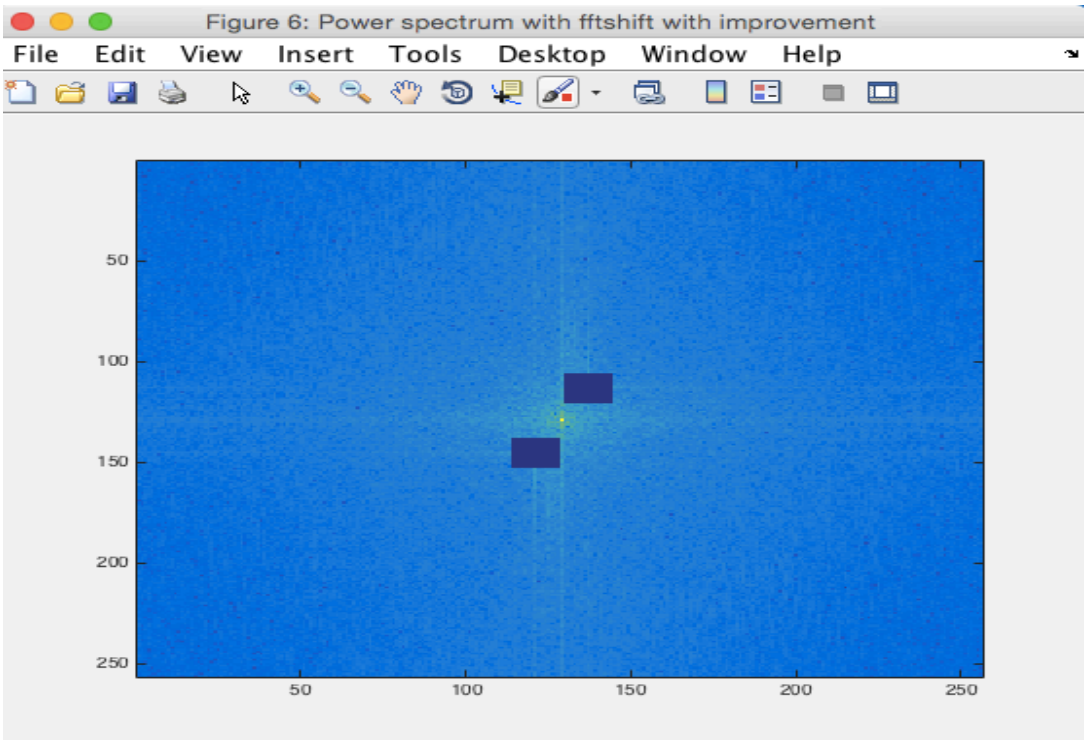

Figure 26 : Improvement by setting elements in neighbourhood of 15X15 to zero

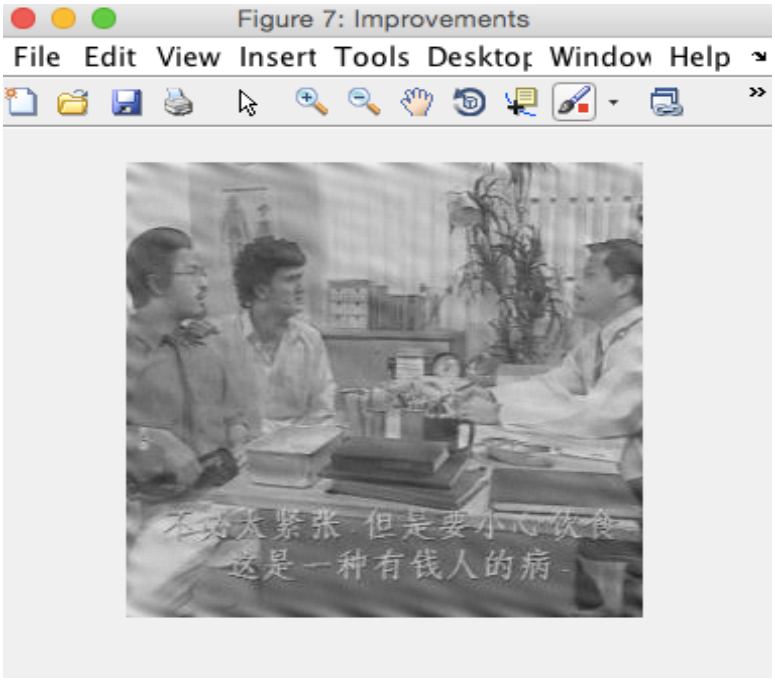

Figure 27 : Resulting image after improvement

- As we can see from the above image, increasing the neighbourhood size to 15X15 helps us remove interference patterns to a greater extent and achieve better results.

II. Caged Primate

Source Code:

```
J=imread('primatecaged.jpg');
I=rgb2gray(J)
figure('name', 'Image with interference');
imshow(I);

ID = im2double(I);
FID = fft2(ID);
S=abs(FID).^2;
figure('name', 'Power spectrum with interference');
colormap('default');
imagesc(S.^0.1);

%Peaks at 5,247 and 253,11 noted from power spectrum
%Setting 7X7 neighbourhood elements around interference peaks to
zero

FID(250:256,8:14)=0;
FID(2:8,244:250)=0;
S1=abs(FID).^2;
figure('name', 'Power spectrum without interference');
colormap('default');
imagesc(fftshift(S1.^0.1));
IFID=ifft2(FID);
NewImg = im2uint8(IFID);
figure('name', 'Free primate');
imshow(NewImg);
```

Outputs:

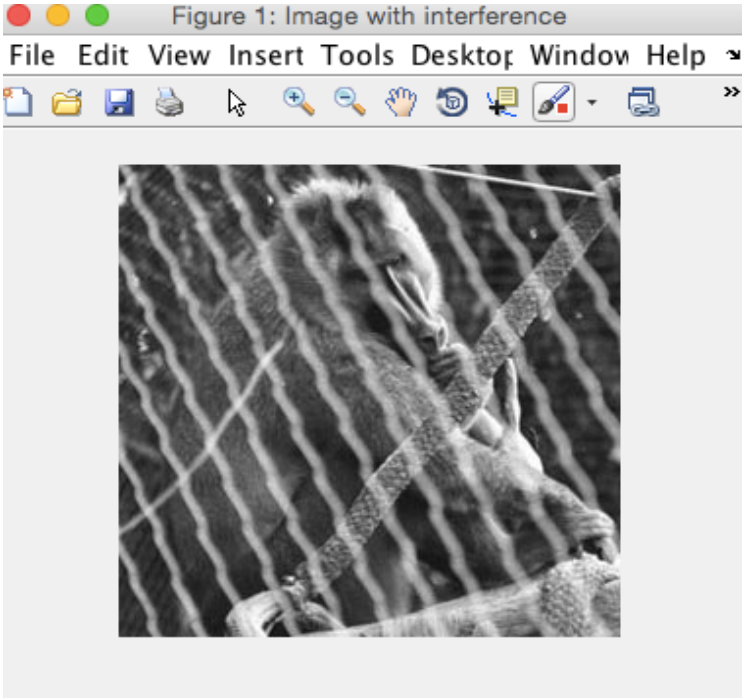

Figure 28 : Original Image with interference

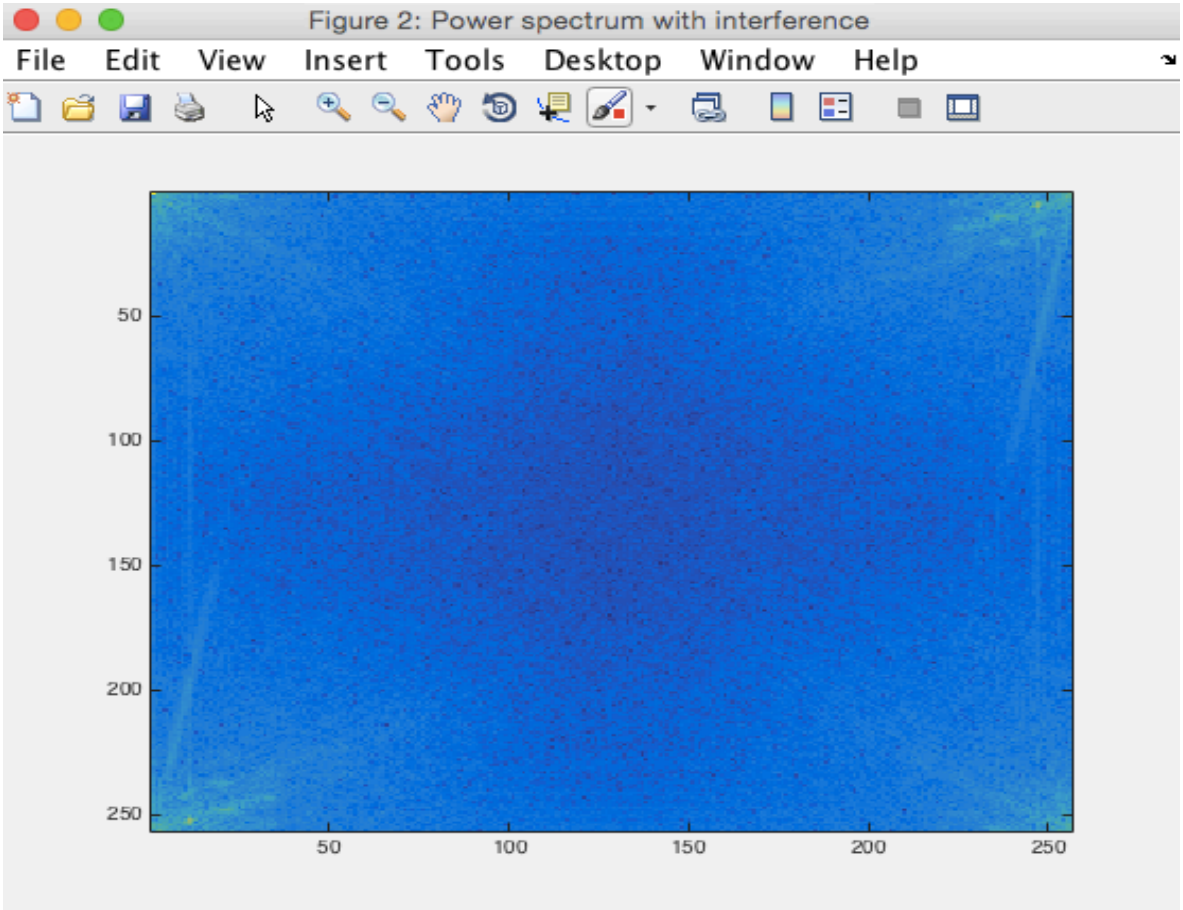

Figure 29 : Power spectrum of image with interference

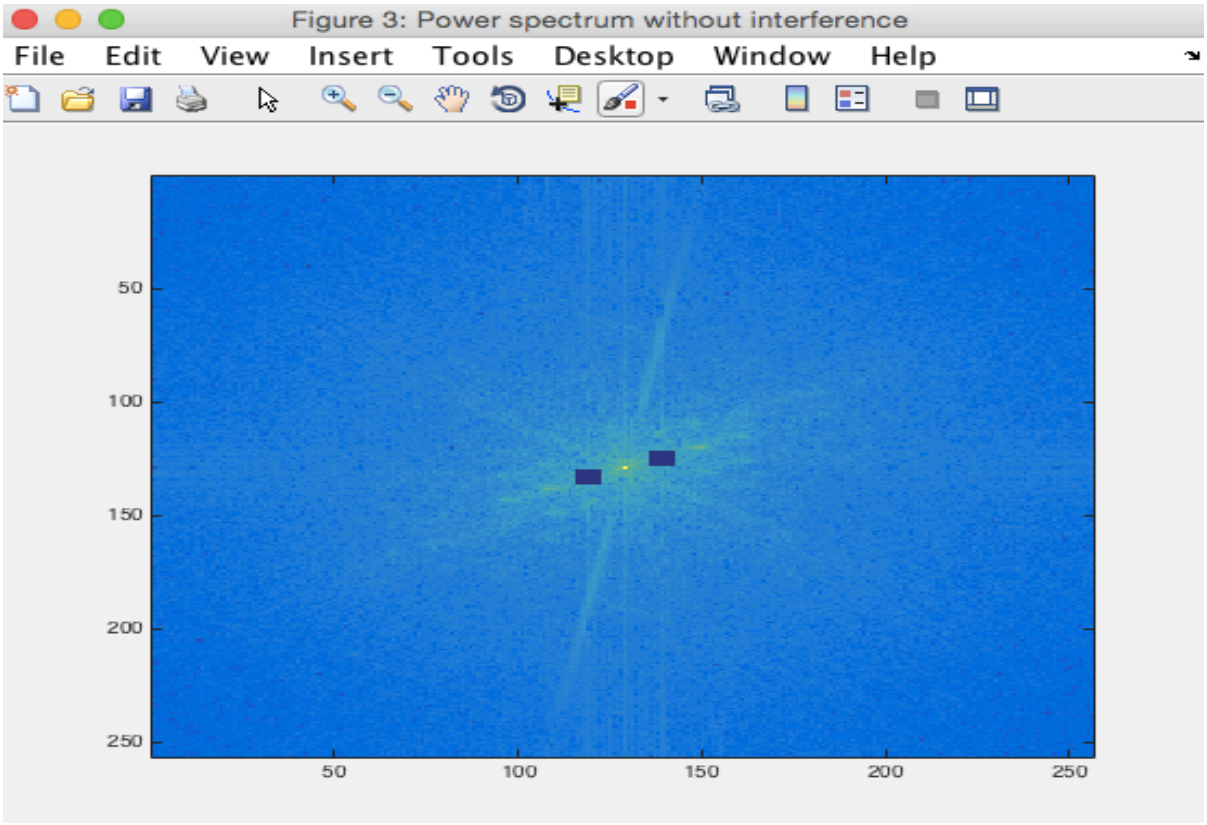

Figure 30 : Power spectrum with fftshift after removing interference

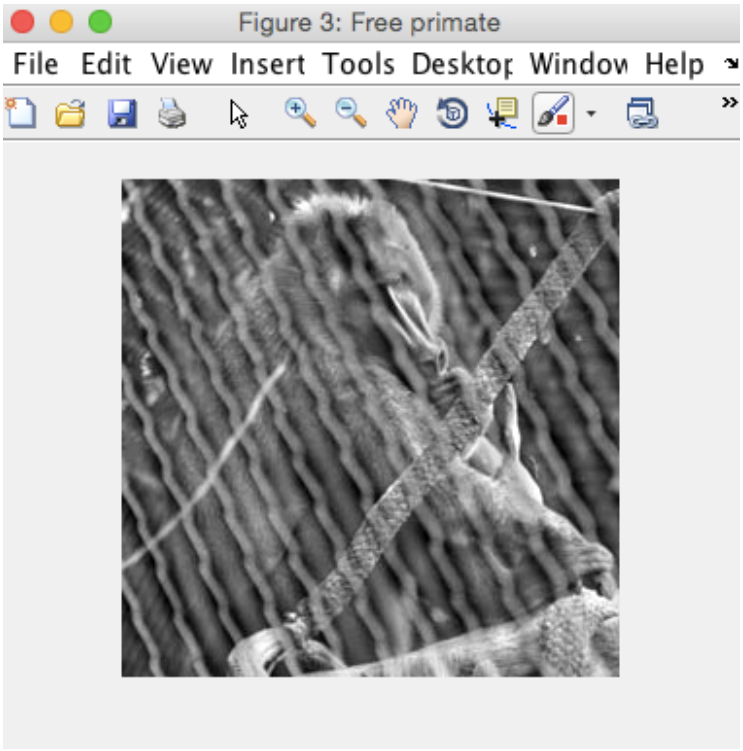

Figure 31 : Resultant image after removing intereference peaks and elements in neighbourhood size 7X7

Experiment 2.6 : Undoing Perspective Distortion of Planar Surface

Source Code:

```
%Part a:
I=imread('book.jpg');
P=rgb2gray(I);
figure('name', 'Original Image');
imshow(P);

%Part b:
[X Y] = ginput(4); % 4 corners of the book obtained in order
top-left, top-right, bottom-right, bottom-left
X
Y


%Desired coordinates, using dimensions of the book 210mmX297mm
x = [0,210,210,0];
y = [0,0,297,297];

%Building matrices v and A
v = [x(1);y(1);x(2);y(2);x(3);y(3);x(4);y(4)];

A = [X(1),Y(1),1,0,0,0,-x(1)*X(1),-y(1)*Y(1);
0,0,0,X(1),Y(1),1,-y(1)*X(1),-y(1)*Y(1);
X(2),Y(2),1,0,0,0,-x(2)*X(2),-x(2)*Y(2);
0,0,0, X(2),Y(2),1,-y(2)*X(2),-y(2)*Y(2);
X(3),Y(3),1,0,0,0,-x(3)*X(3),-x(3)*Y(3);
0,0,0, X(3),Y(3),1,-y(3)*X(3),-y(3)*Y(3);
X(4),Y(4),1,0,0,0,-x(4)*X(4),-x(4)*Y(4);
0,0,0, X(4),Y(4),1,-y(4)*X(4),-y(4)*Y(4)];

%Part c:
u=A\v;
U = reshape([u;1], 3, 3)'
w = U*[X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:))

%Part d:
T = maketform('projective', U');
P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);


%Part e:
figure('name', 'Transformed Image');
imshow(P2);
```

Outputs and Explanations:

```
X =

  143.0000
  309.0000
  257.0000
    3.0000

Y =

   27.0000
   47.0000
  223.0000
  160.0000

U =

    1.4944    1.5730 -256.1645
   -0.4475    3.7142  -36.2919
    0.0002    0.0055    1.0000

w =

         0  210.0000  210.0000   -0.0000
         0   -0.0000  297.0000  297.0000
    1.0000    1.0000    1.0000    1.0000
```

Here, the w matrix does give us back the coordinates of the 4 corners of the desired image, which means the transformation is correct.

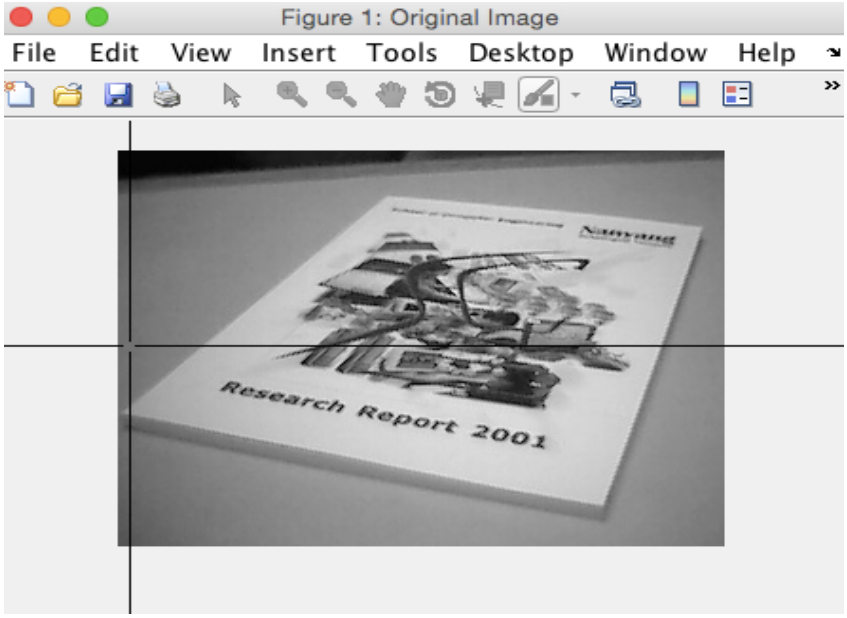

Figure 32 : Original Image with slanted book

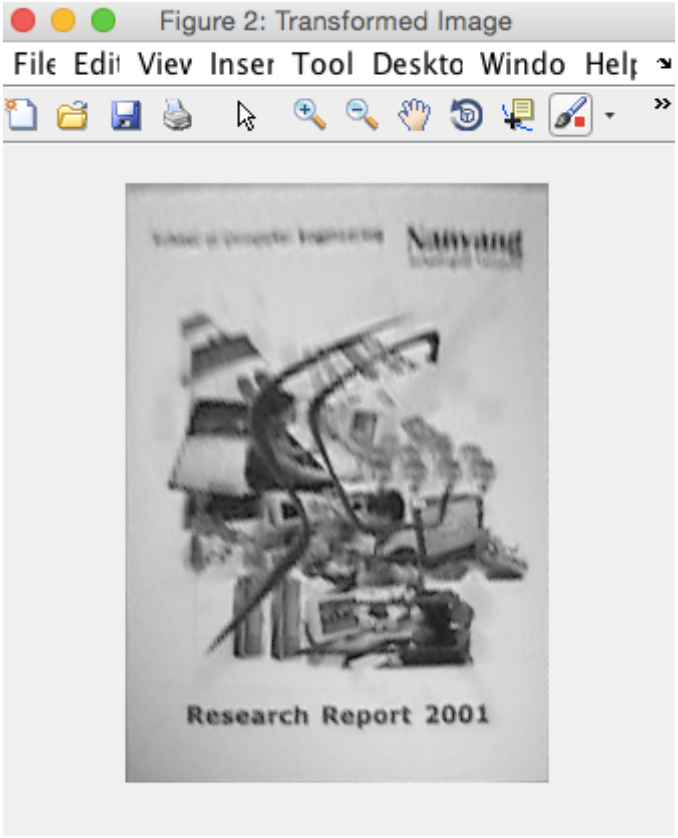

Figure 33 : Image obtained after projective transformation

Comments:

The result of the projective transformation shows a frontal view of the book and is what we expected as the result. However, there is some blurring in the top part of the image which is because the top part of the book is not clearly visible in the original slanted image. The transformation is not able to recreate the top part of the image clearly.
The quality of the transformation is still decent, as the frontal view is accurate and the majority of the image is still clear.

LAB 2

Experiment 3.1 : Edge Detection

Source Code:

```
%Part a:

P=imread('macritchie.jpg');
Pc=rgb2gray(P);
figure('name', 'Original Image');
imshow(Pc)

%Part b:

GX=[-1,-2,-1;0,0,0;1,2,1];  %Horizontal Sobel Filter
GY=[-1,0,1;-2,0,2;-1,0,1];  %Vertical Sobel Filter
FX=conv2(Pc, GX);
figure('name', 'Horizontal Sobel Filter Output');
imshow(FX);
FY=conv2(Pc,GY);
figure('name', 'Vertical Sobel Filter Output');
imshow(FY)

%Part c:

F=uint8(sqrt(FX.^2+FY.^2)); %Combined Sobel Filter
figure('name', 'Combined Sobel Filter Output');
imshow(F)

%Part d:

Et=F>100; %Threshholding
figure('name', 'Threshhold=100');
imshow(Et)
Et2=F>200;
figure('name', 'Threshhold=200');
imshow(Et2)

%Part e:

E1 = edge(Pc,'canny',[0.04 0.1],1.0);
figure('name', 'Canny Edge Detector Output(Sigma=1.0, tl=0.04,
th=0.1');
imshow(E1)

E2 = edge(Pc,'canny',[0.04 0.1],3.0);
figure('name', 'Canny Edge Detector Output(Sigma=3.0, tl=0.04,
th=0.1');
imshow(E2)
```

```
E3 = edge(Pc,'canny',[0.04 0.1],5.0);
figure('name', 'Canny Edge Detector Output(Sigma=5.0, tl=0.04,
th=0.1');
imshow(E3)

E4 = edge(Pc,'canny',[0.09 0.1],1.0);
figure('name', 'Canny Edge Detector Output(Sigma=1.0, tl=0.09,
th=0.1');
imshow(E4)

E5 = edge(Pc,'canny',[0.02 0.1],1.0);
figure('name', 'Canny Edge Detector Output(Sigma=1.0, tl=0.02,
th=0.1');
imshow(E5)
```

Outputs and Explanations:

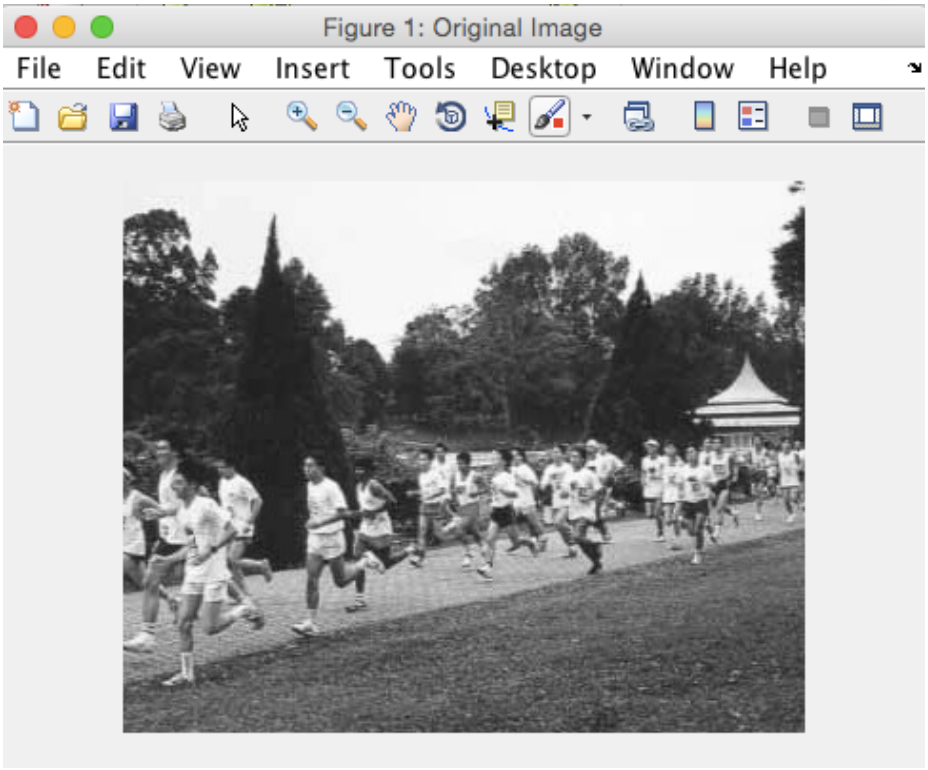

Figure 34 : Original grayscale image

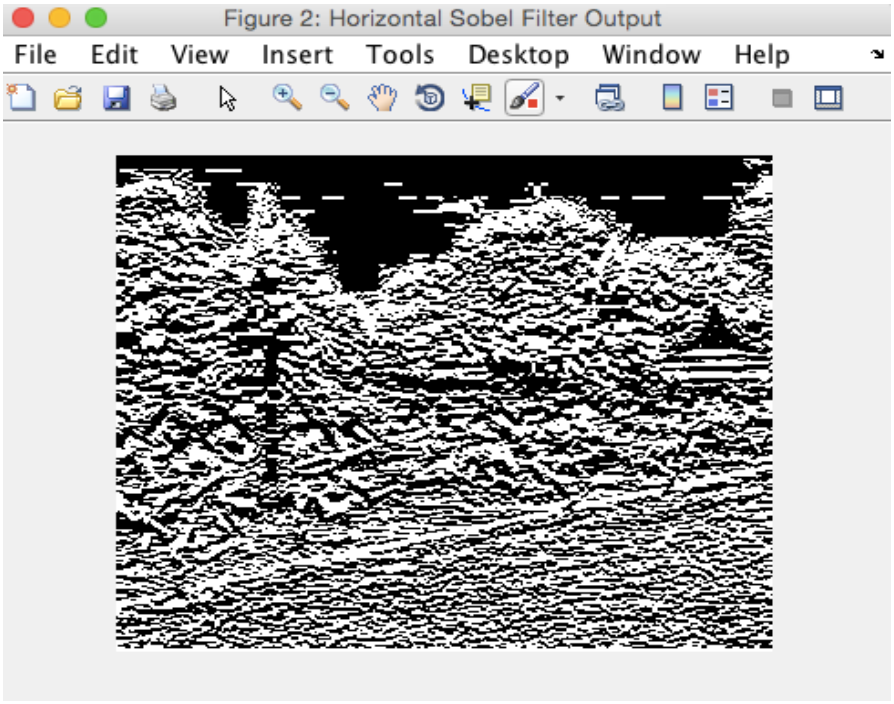

Figure 35 : Horizontal Sobel Filter Output

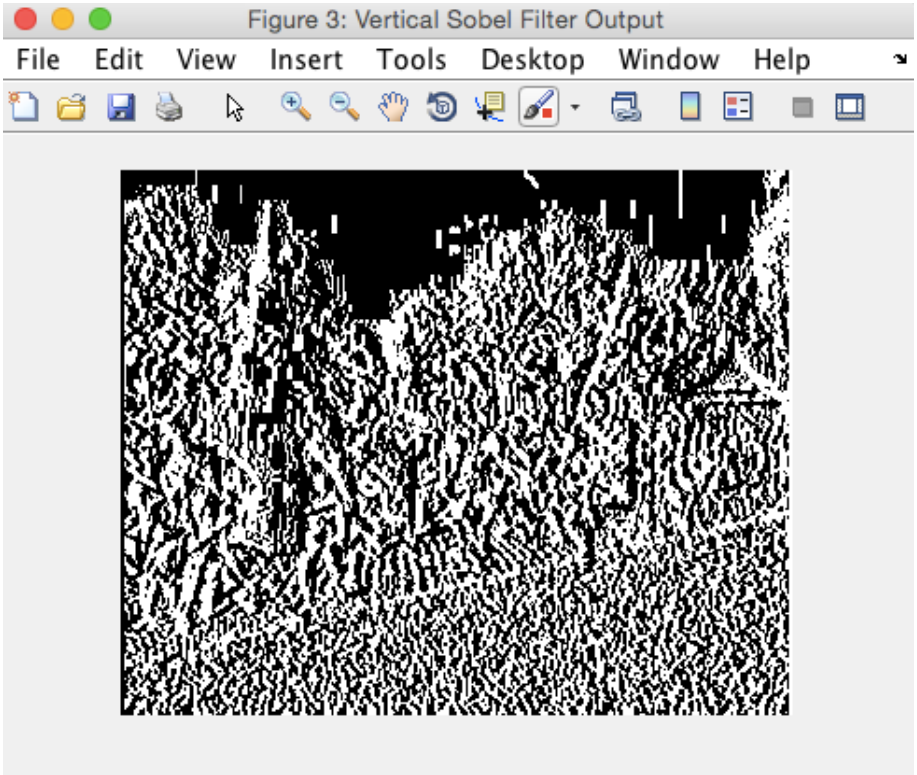

Figure 36 : Vertical Sobel Filter Output

From the above two filtered outputs, we can observe that edges that are not strictly vertical or horizontal are partially detected and approximated by the filters.

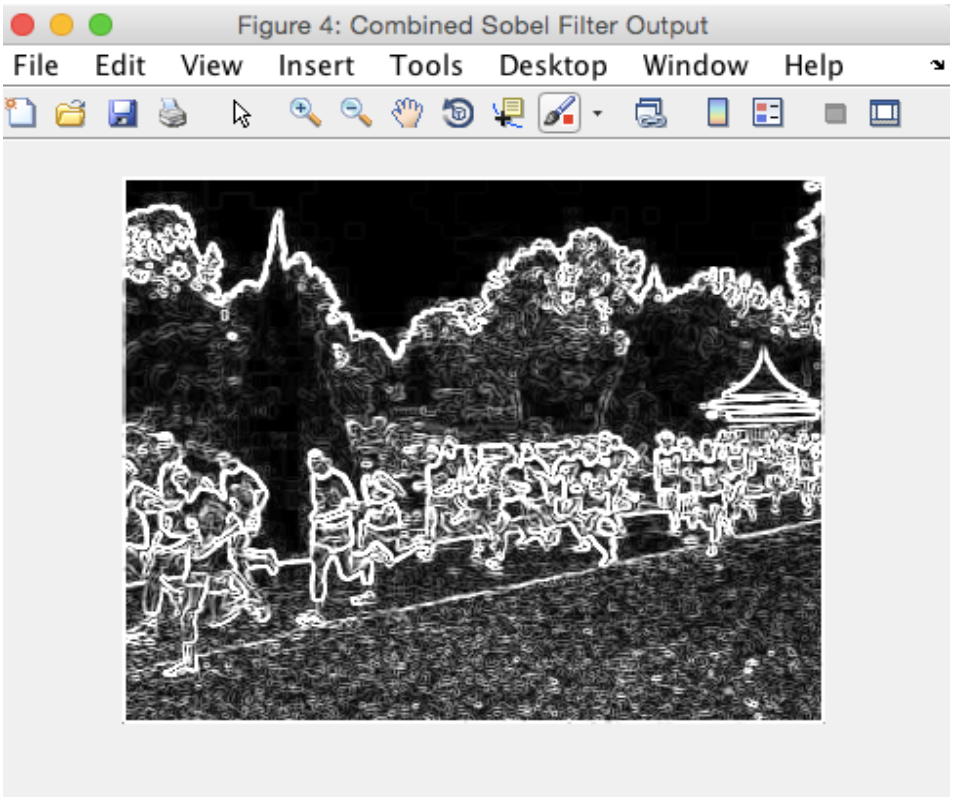

Figure 37 : Filter Output of squared and summed horizontal and vertical sobel filter

- The horizontal and vertical Sobel filters estimate the horizontal or vertical gradient magnitudes of the image. We use a squaring operation to combine the outputs of these two filters, which detect edges in horizontal and vertical directions.

- The combined filter gives us the exact edges of the image in all directions by approximating the absolute gradient magnitude. The image we obtain is the gradient image.

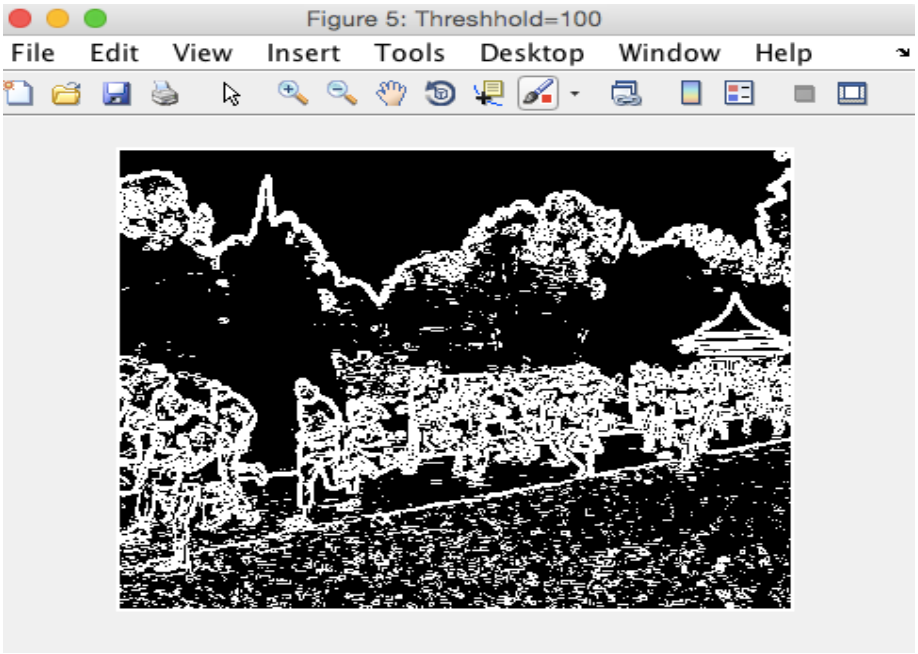

Figure 38 : Combined filter output with threshhold=100

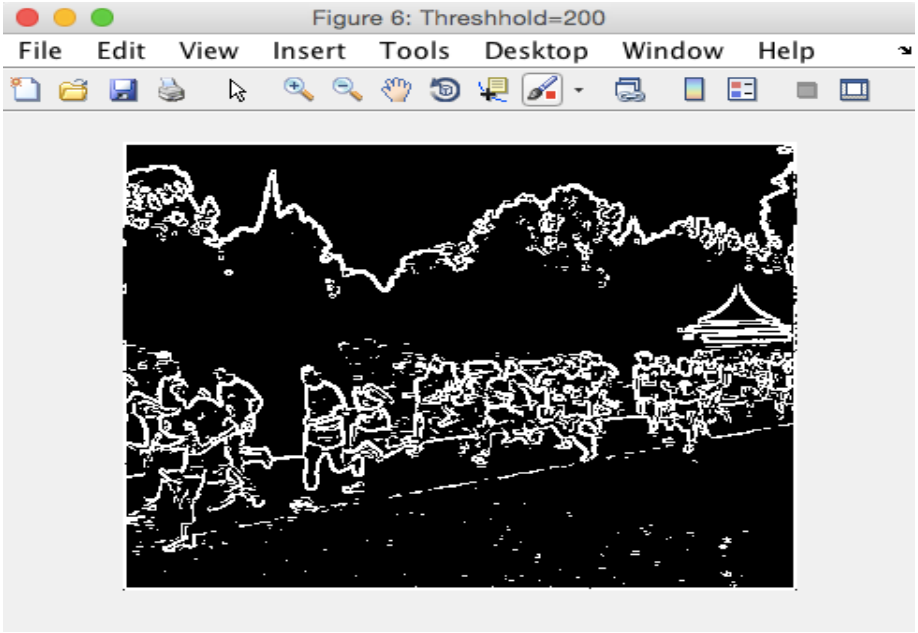

Figure 39 : Combined Filter Output with threshhold=200

- **Effect of changing threshhold** : The lower the threshold, the more edges will be detected, but the result is more susceptible to noise and detecting edges of irrelevant features in the image. Since the edge threshholding has been applied to the gradient image, the edges in the resultant image are thick.
- **Advantages and Disadvantages of threshhold values** : Using a lower threshhold(100) allows us to detect more edges in the image but introduces some noise in the output, whereas using a higher threshhold(200), we may detect relatively lesser edges and detect some fragmented edges, but there is lesser noise in the output.

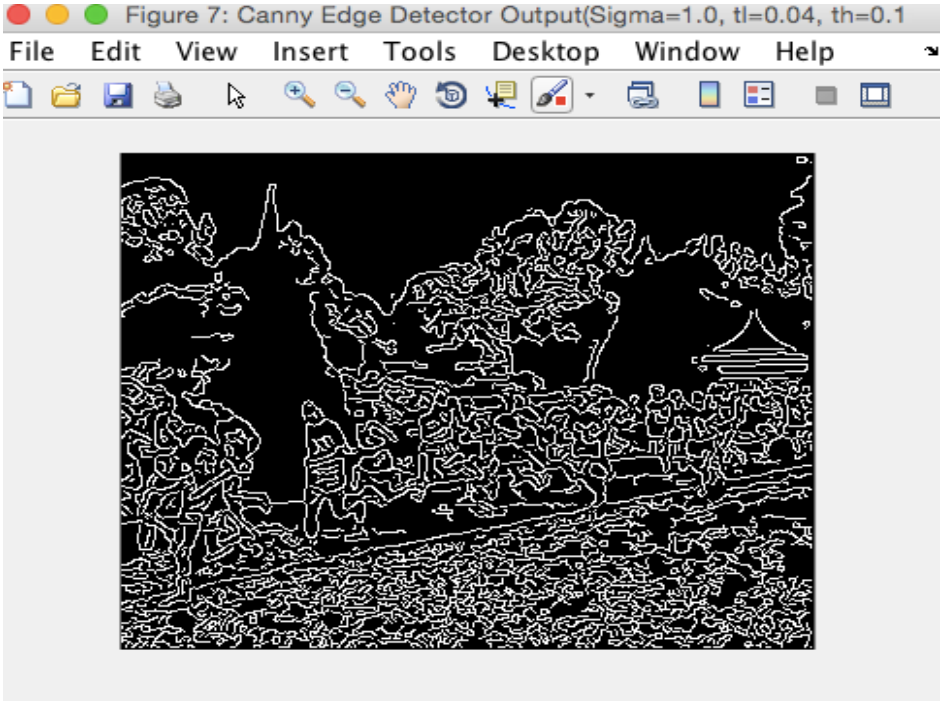

Figure 40 : Canny Edge Detector Output (Sigma=1.0, tl=0.04,th =0.1)

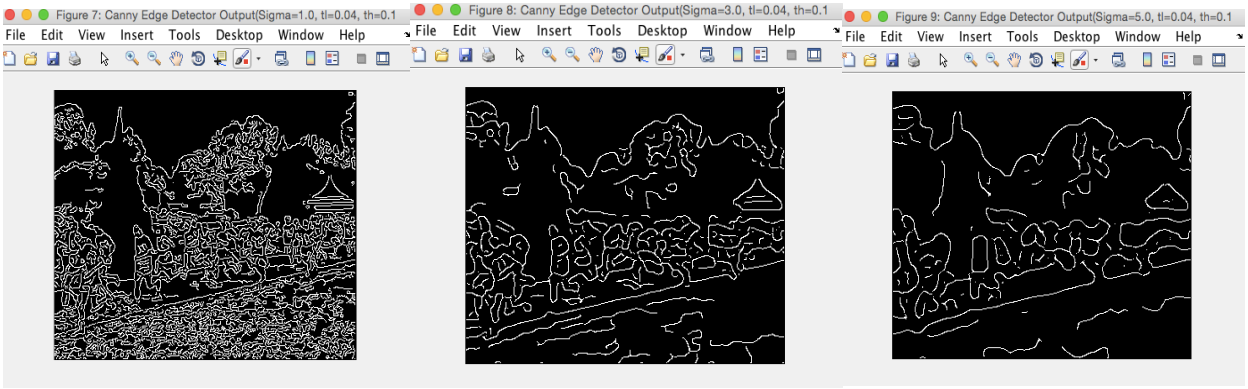

Figure 41 : Canny Edge Detector Output (tl=0.04, th=0.1) – Effect of increasing Sigma -1.0,3.0,5.0

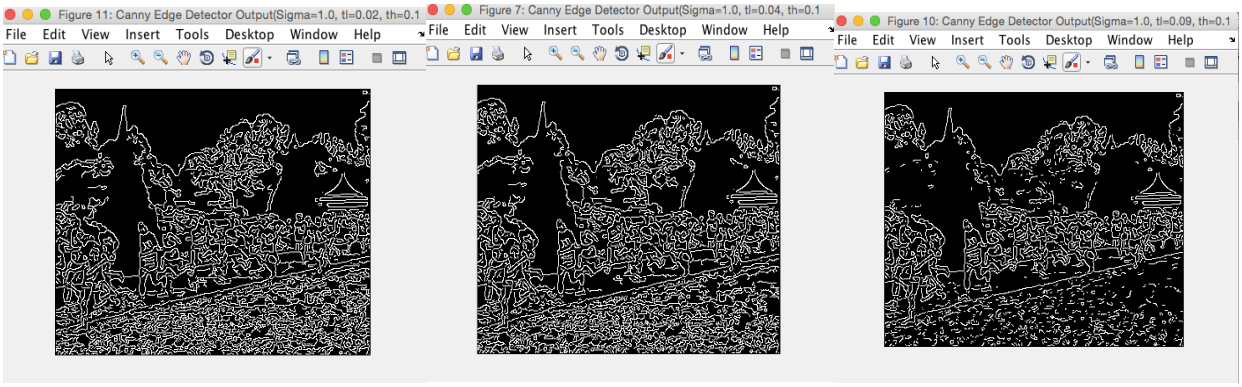

Figure 42 : Canny Edge Detector Output (Sigma=1.0, th=0.1) -> Effect of increasing lower threshhold – 0.02,0.04, 0.09

Explanation:

- **Increasing value of sigma:** We can observe the output of the Canny Edge Detector as we increase the sigma from 1.0 to 3.0 to 5.0 in the Figure 41. Sigma represents the standard deviation of the Gaussian filter used in edge detection and decides the size of the filter used. As we increase the value of sigma, we detect lesser edges in the image and the output becomes less detailed. We can also notice that sigma=3.0 is better for location accuracy of edgels and sigma=5.0 is better at handling noisy edges.

- **Changing lower threshhold:** We use two threshholds tl and th to perform hysteresis threshholding. Pixels with magnitudes smaller than th but larger than tl are set to 1 if they have neighbouring pixels perpendicular to the edge gradient that have been set to 1. It is useful in removing tiny noisy edges.
  From the output in Figure 42, we can observe that as we lower the value of tl (0.02), we detect many fragmented edges in the upper and bottom part of the image. On the other hand, as tl(0.09) becomes closer to th(0.1), we detect more connected edgels. This is expected as a lower threshhold allows more edges to be set to 1 depending on their neighbouring pixels through hysteresis threshholding.

Experiment 3.2 : Line Finding using Hough Transform

Source Code:

```
%Part a:

P=imread('macritchie.jpg');
Pc=rgb2gray(P);
E = edge(Pc,'canny',[0.01 0.1],1.0);

%Part b:

[H, xp] = radon(E);
figure('name','Hough Transform as image');
imagesc(H);
colormap('hot');
xlabel('\theta (degrees)'); ylabel('x\');
title('Hough Transform');
colorbar
% figure('name', 'Hough Transform displayed as image');
% imshow(H);

%Part c:

[Houghmax,index] = max(H(:));
[m,n] = ind2sub(size(H),index);
radius=xp(m);
theta=n;
[theta,radius]

%Part d:

[A, B] = pol2cart(theta*pi/180, radius);
B=-B;
[A B]
C=(A*radius*cos(theta*pi/180))+(-B*radius*sin(theta*pi/180));

%Part e:

imsize = size(Pc);
xl=0; xr=imsize(2)-1;
yl=((C-A*xl)/B);
yr=((C-A*xr)/B);

%Part f:

figure('name', 'Superimposing estimated line on original
image');
imshow('macritchie.jpg');
line([xl xr], [yl yr],'LineWidth',3);
```

Outputs and Explanations:

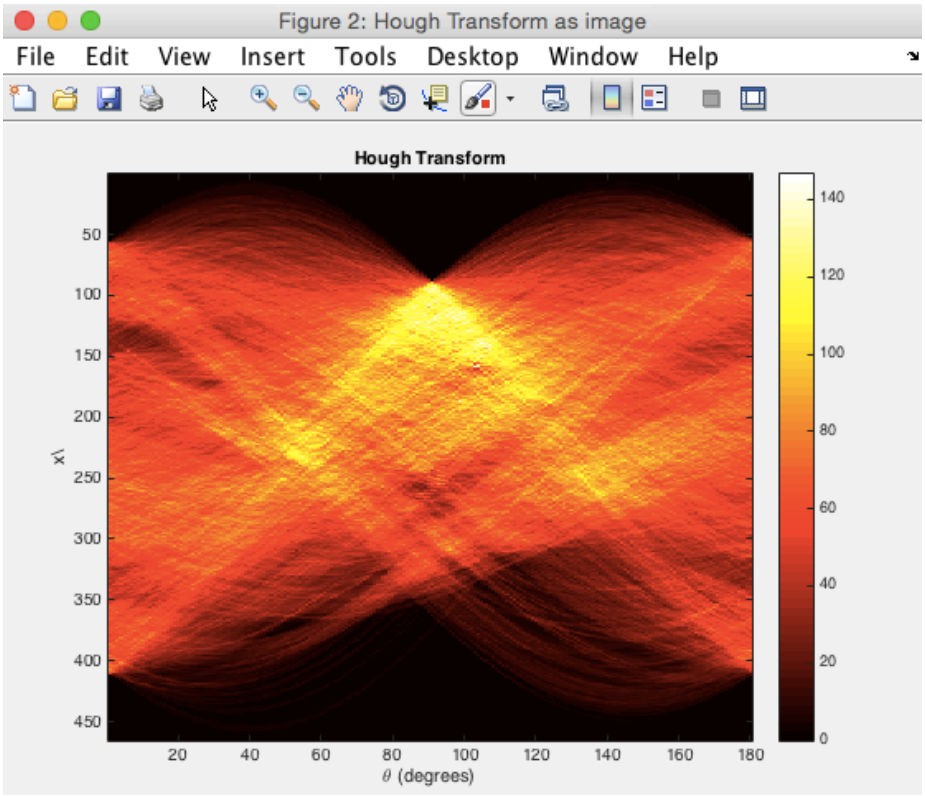

Figure 43 : Hough Transform displayed as an image

Why is Radon Transform equivalent to Hough Transform in this case?

- Radon transform is very similar to Hough transform, in a way that Hough transform is like a discretized version of the Radon transform. The Hough transform applies a discrete algorithm that detects lines by polling and binning whereas the Radon transform computes line integrals of continuous functions.
- The location of the peaks in the Radon transform matrix correspond to the location of straight lines in the original image. So it can be used to compute the classical Hough transform.
- During this experiment, we are attempting to extract the long edge of the image that is a straight line, therefore, Radon transform would work well. However, for a generalized Hough transform, where we detect parameterized curves, the Hough and Radon transform would be different.

We obtained [theta, radius] = [105,91] for the location of the maximum pixel intensity in the Hough image. This was also verified by checking the intensity at the respective coordinates in the Hough image. The value of C was found to be 8836.

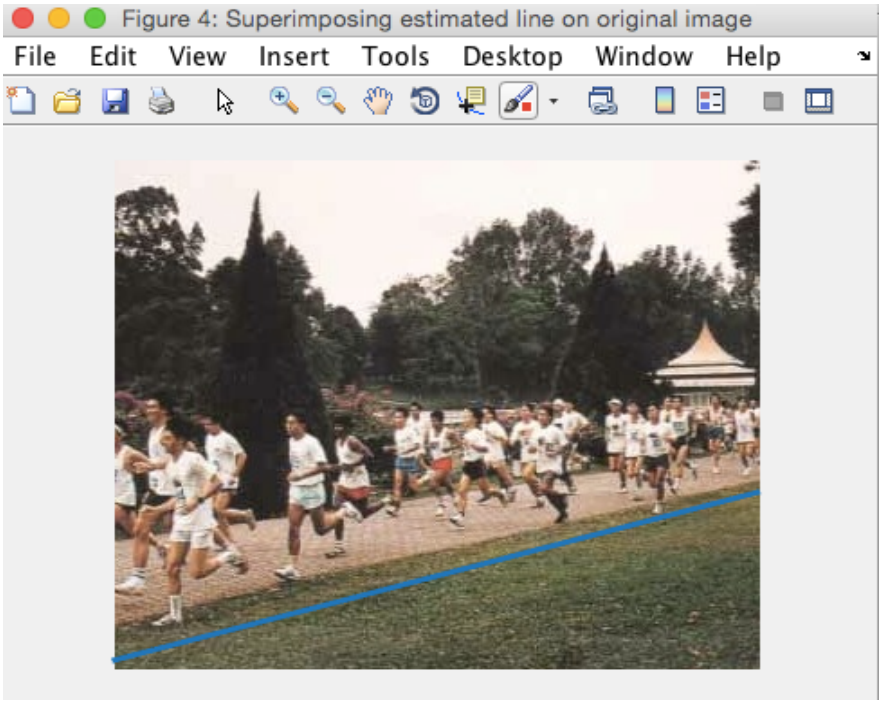

Figure 44 : Superimposition of estimated line on original image, sigma=1.0

As we can see, our estimated line does not exactly match up with the edge of the running path. This might have occurred due to noisy edges around the running path that cause our estimated line to deviate. A possible fix could be to modify the canny edge detector parameters used for edge processing. We had observed in the previous experiment that as we increase the value of sigma, lesser edges are detected, and higher sigma is better for handling noisy edges.
As we increase the value of sigma to 5.0 in the below image,we observe that the estimated line exactly superimposes the running path.

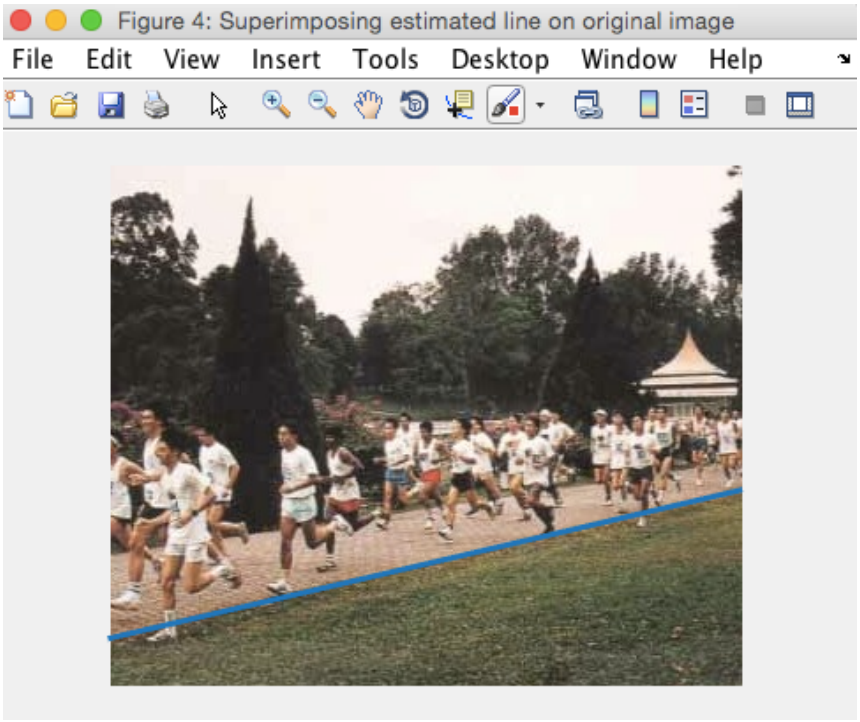

Figure 45: Superimposition of estimated line on original image, sigma=5.0

Experiment 3.3 : 3D Stereo

Source Code:

```
function[] = stereo(imageLeft, imageRight, templatel, templatew)

k1=imread(imageLeft);
k2=imread(imageRight);
k1c=rgb2gray(k1);
k2c=rgb2gray(k2);
[r,c] = size(k1c);
dispMap=ones(r,c); %Initializing disparity map
minValue=100; %Initializing a min disparity value
matchedX=0; %Initializing SSD matched pixel x-coordinate

for i=1+templatew:r-templatew
    for j=1+templatel:c-templatel
        leftpatch=k1c(i-templatew:i+templatew,j-
templatel:j+templatel);  %Extracting patch from left image
          %Scanning right image in the same row
          for l=max(templatew+1,i-15):min(256-templatew,i+15)
            rightpatch=k2c(l-templatew:l+templatew,j-
templatel:j+templatel); %Extracting patch from right image
            S=corr2(rightpatch.^2,ones(size(rightpatch)))-
2*corr2(rightpatch,leftpatch);  %Finding value of S using
correlation
            if (S<minValue)     %Computing minimum value of S
                minValue=S;
                matchedX=l; %Obtains the SSD matched pixels X-
coordinate
            end
        end
        dispMap(i,j) = i-matchedX;   %Calculation of disparity
map
    end
end

imshow(uint8(-dispmap),[-15,15]);  %Displays the disparity map
```

Outputs and Explanations:

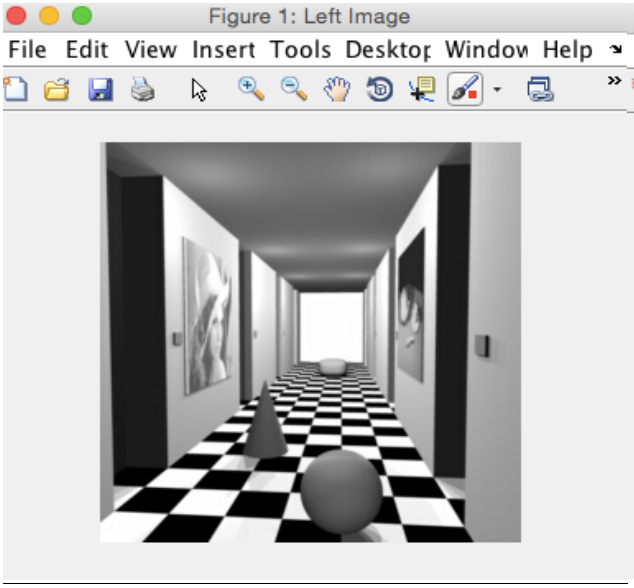

Figure 46 : Left Image : corridorl.jpg

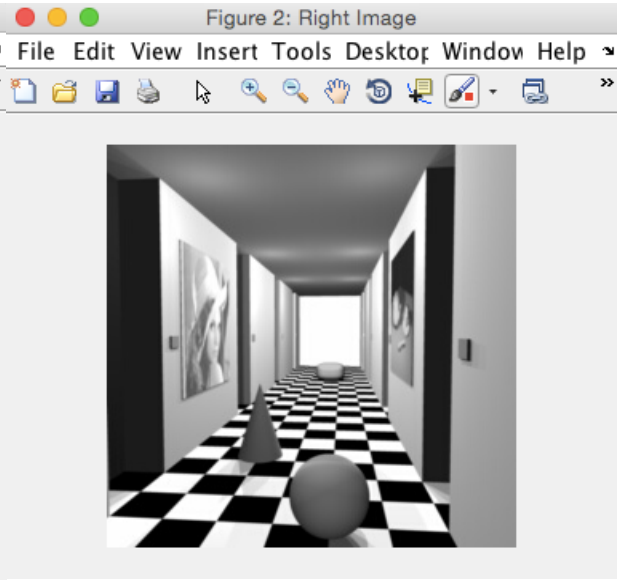

Figure 47 : Right Image : corridorr.jpg

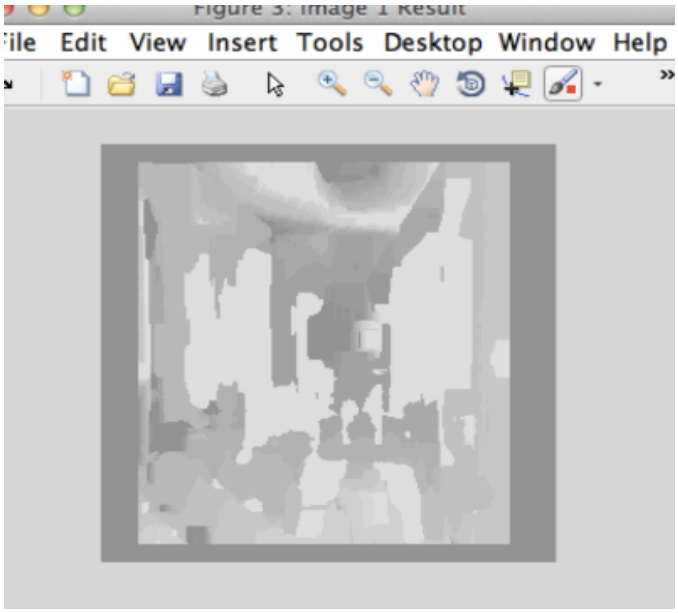

Figure 48 : Disparity Map

Since the original images contain defined shapes like rectangles, triangles, the disparity map is of good quality. We can observe that the components of the disparity map are also distinguishable corresponding to the local image, the lighter rectangles at the bottom of the disparity map are closer and the darker rectangle at the center of the image is farther.

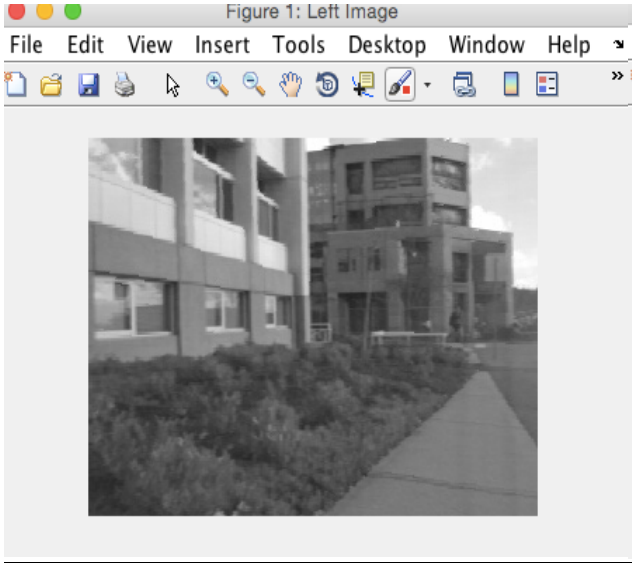

Figure 49 : Left Image : triclopsi2l.jpg

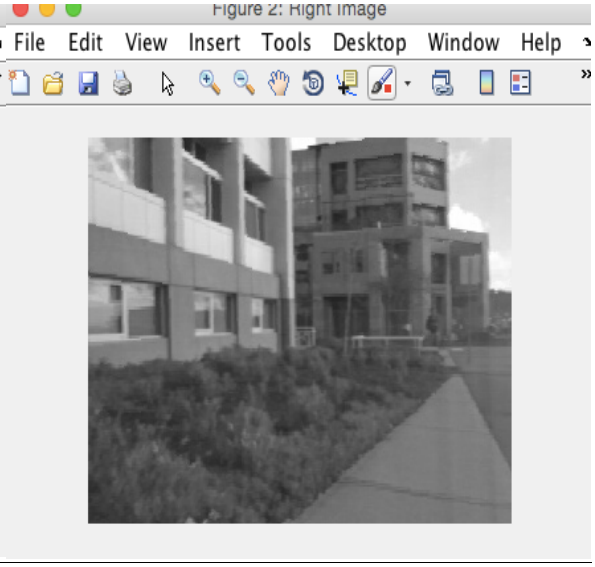

Figure 50 : Right Image : triclopsi2r.jpg

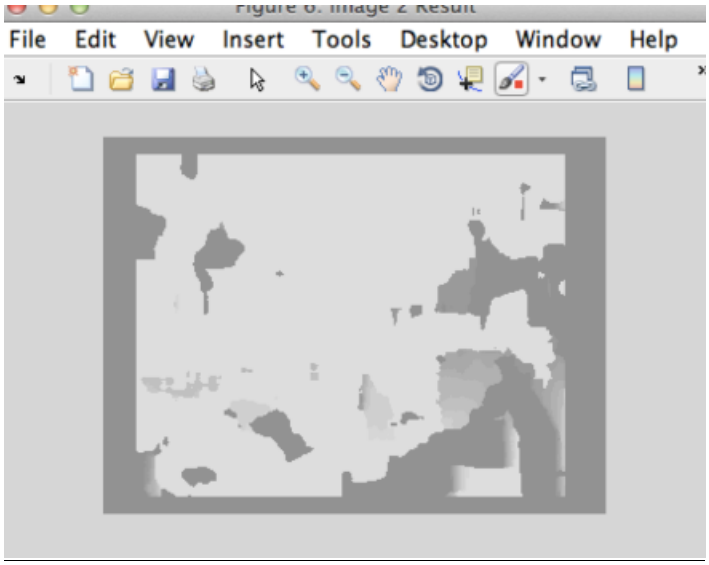

Figure 51 : Disparity Map

In these images' disparity map, components of the image are not well distinguishable, 80% of the disparity map is almost transparent. Therefore, the depths of different parts of the image are not easily identified. This is because the image structure of the original images is not well defined with forms and shapes. We can conclude that the image structure of the stereo images greatly affects the quality of the disparity map produced.