Lab Assignment 1 Report

Name: Vedula Kartikeya

Matriculation Number: U1923891G

Subject Code: CE4003

2.1 Contrast Stretching:

a. The following Matlab code was used to read the image and run the whos command:

```
% a. Input Image
image = imread('D:\NTU Class\CE4003\mrt-train.jpg');

%Check for RGB or Gray_scale
whos image
```

The *whos* command returns the following output:

```
  Name          Size                Bytes  Class    Attributes

  image       320x443x3            425280  uint8
```

The output displays an image that can be read as a 320x443x3 uint8 matrix indicating a colour image with byte values.

The Matlab code shown below is used to convert the image to grayscale using the *rbg2gray* command and the *whos* command is run again for verification purpose:

```
%Convert to grayscale
P = rgb2gray(image);
whos P
```

After converting our image to grayscale by utilizing the *rgb2gray* command, the dimensions of the matric changes from 320x443x3 to 320x443. It can be seen in the output displayed below:

```
  Name          Size                Bytes  Class    Attributes

  P           320x443              141760  uint8
```

Hence, it can be shown that our original colour image has been converted to grayscale image.

b. Image is viewed by the command:

```
% b. Show Image
imshow(P);
```

This will display the “mrt-train.jpg” as required:

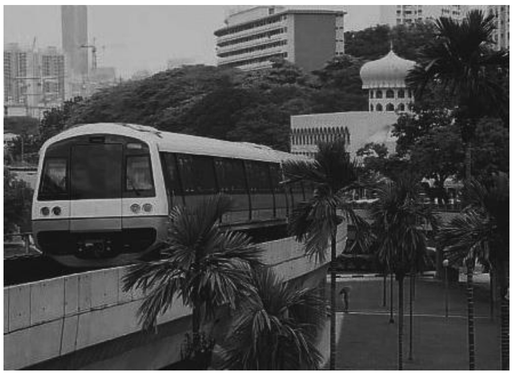
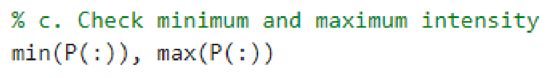


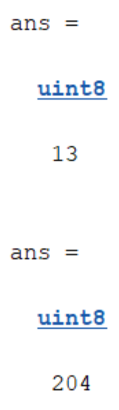
Figure 1: Image of original mrt-train

c. Following Matlab code is used to find the minimum and maximum intensities present in the image:



```
% c. Check minimum and maximum intensity
min(P(:)), max(P(:))
```

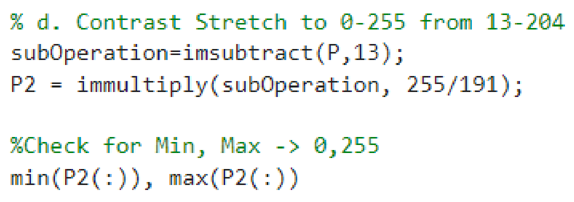
min(P(:)) returns 13, while max(P(:)) returns 204 and can be seen in the image below:



```
ans =

  uint8

   13


ans =

  uint8

   204
```

d. The min and max values observed above are recorded and used to perform contrast stretching.

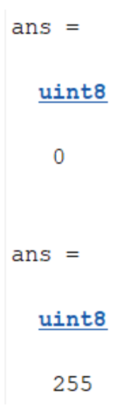
The following code demonstrates contrast stretching:



```
% d. Contrast Stretch to 0-255 from 13-204
subOperation=imsubtract(P,13);
P2 = immultiply(subOperation, 255/191);

%Check for Min, Max -> 0,255
min(P2(:)), max(P2(:))
```

We first subtract the minimum value to reduce the minimum intensity to 0 from 13. Furthermore, a multiplication operation is performed with (255/191) so that the maximum intensity can be increased from 191 to 255 and a correct minimum and maximum intensity of 0 and 255 can be obtained respectively.

To validate our code, min and max operation are performed again and the following output is generated:



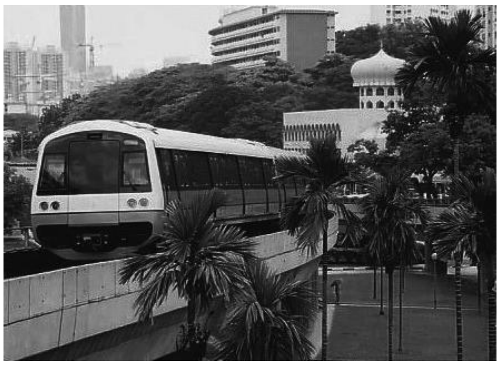
```
ans =

  uint8

   0


ans =

  uint8

   255
```

e. Final image P2 is displayed using the Matlab command:

```
% e. Show Stretched Image
imshow(P2,[]);
```

This displays the following image:

Figure 2: Image of original mrt-train after contrast stretching



As noticed, the contrast of the image has improved.

f. Source Code:

```
%------------------2.1--------------------%
% a. Input Image
image = imread('D:\NTU Class\CE4003\mrt-train.jpg');

%Check for RGB or Gray_scale
whos image

%Convert to grayscale
P = rgb2gray(image);
whos P

% b. Show Image
imshow(P);

% c. Check minimum and maximum intensity
min(P(:)), max(P(:))

% d. Contrast Stretch to 0-255 from 13-204
subOperation=imsubtract(P,13);
P2 = immultiply(subOperation, 255/191);

%Check for Min, Max -> 0,255
min(P2(:)), max(P2(:))

% e. Show Stretched Image
imshow(P2,[]);
```

2.2 Histogram Equalization:

a. Image intensity histogram of P using 10 and 256 bins is displayed by the following code:

```
% a. Show Image Intensity Historygram of Gray Image
imhist(P,10)
imhist(P, 256)
```

The following histograms are obtained:

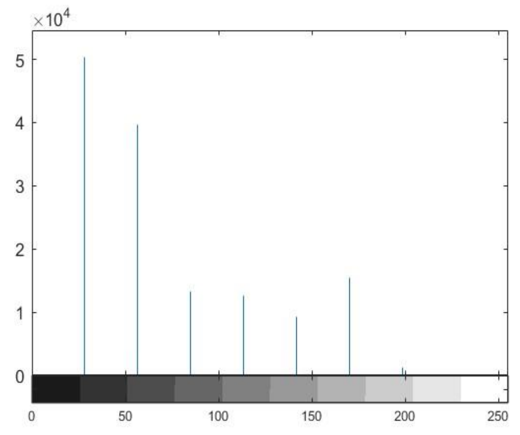
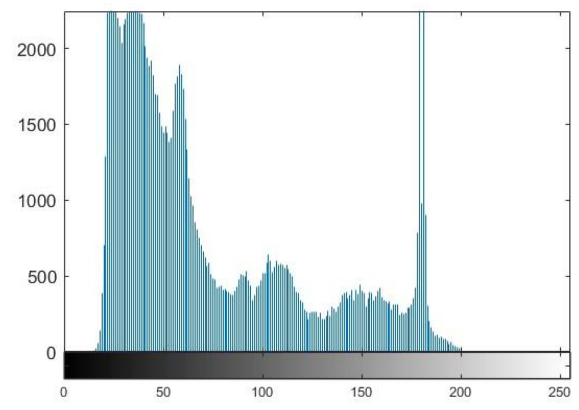
	

Figure 3: Image intensity histogram of P using 10 bins	Figure 4: Image intensity histogram of P using 256 bins

The image intensity histogram of P using 256 shows a better distribution of gray levels than using 10 bins.

This is due to the fact that a histogram with 10 bins has more pixels per bin than a histogram with 256 bins, so it contains far fewer details about the distribution of gray levels. Furthermore, because grayscale images have 256 gray levels, a histogram with 256 bins can show us the details of each grey level.

b. The following code performs histogram equalization:

```
% b. Do Histogram Equalization for Gray Image
P3=histeq(P,255);
imhist(P3, 10)
imhist(P3, 256)
```

Image intensity histogram of P3 using 10 and 256 bins is displayed by using the *imhist* command.

Figures 5 and 6 show that after histogram equalization, the histogram becomes more uniform. Figure 5's histogram, on the other hand, is flatter than Figure 6. This demonstrates that the histogram cannot be completely flat after histogram equalization at a more detailed level of 256 bins, but it appears fairly balanced at a less detailed level of 10 bins.

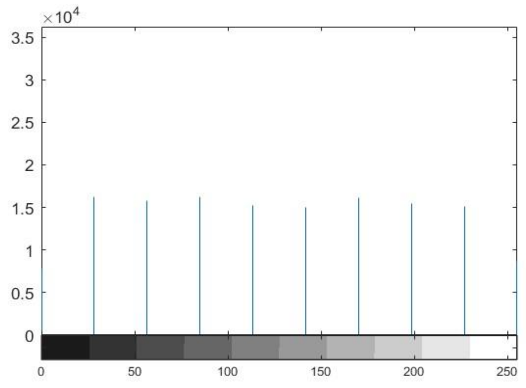


Figure 5: Image intensity histogram of P3 using 10 bins

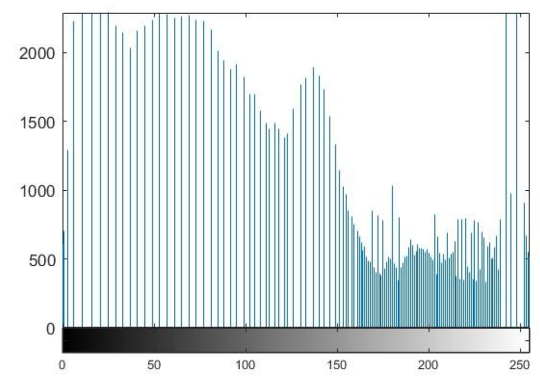


Figure 6: Image intensity histogram of P3 using 256 bins

c. Following code is used to rerun histogram equalization on P3:

```
% c. Re-running histogram equalization on P3
P4=histeq(P3,255);
imhist(P4, 10)
imhist(P4, 256)
```

The following histograms obtained:

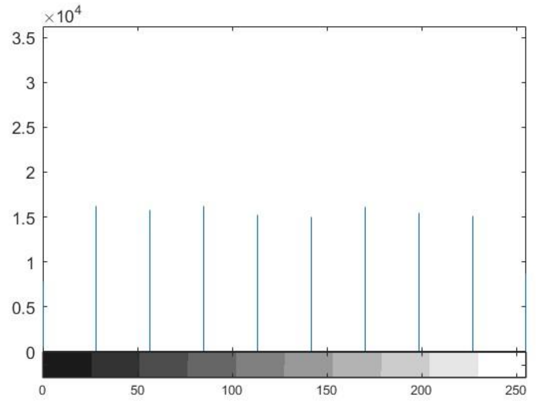


Figure 7: Image intensity histogram of P4 using 10 bins

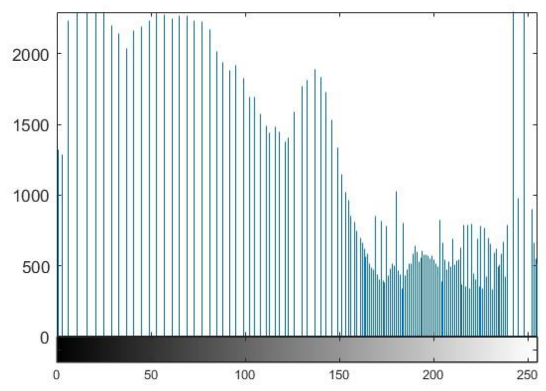


Figure 8: Image intensity histogram of P4 using 256 bins

The histogram, as seen, **does not** become more uniform. In fact, they are nearly identical. This is because the histogram is already mapped out based on the average number of pixels in each bin after the first run of histogram equalization, ensuring that the distribution of gray level intensity is as uniform as possible. Hence, majority of the bins will already be in the correct place, causing the histogram to remain nearly unchanged.

d. Source Code:

```
%------------------2.2-------------------%
% a. Show Image Intensity Historygram of Gray Image
imhist(P,10)
imhist(P, 256)

% b. Do Histogram Equalization for Gray Image
P3=histeq(P,255);
imhist(P3, 10)
imhist(P3, 256)

% c. Re-running histogram equalization on P3
P4=histeq(P3,255);
imhist(P4, 10)
imhist(P4, 256)
```

2.3 Linear Spatial Filtering:

a. Below code is used to generate Gaussian filters with sigma=1.0 and sigma=2.0 after normalizing them

Furthermore, we use *mesh* function to visualize the 3D graph of the filters:

```
% a. Generate filters
% (i) Y and X-dimensions are 5 and σ = 1.0
sig1=1.0;
% (ii) Y and X-dimensions are 5 and σ = 2.0
sig2=2.0;

dimension=5;
range=-floor(dimension/2):floor(dimension/2);
[X, Y]=meshgrid(range,range);

filter1=h(X,Y,sig1);
filter2=h(X,Y,sig2);

%Normalizing filters
filter1=filter1/sum(filter1(:));
filter2=filter2/sum(filter2(:));

%Display filters
mesh(filter1)
mesh(filter2)
```

The *h (x, y) function described in the lab manual has been implemented to generate the filters*:

```
% 2.2 a Normalizing Filter function
function h = h(x,y,sigma)
    h=(1/(2*pi*power(sigma,2)))*(exp(-(power(x,2)+power(y,2))/(2*power(sigma,2))));
end
```

Above code displays the following graphs:

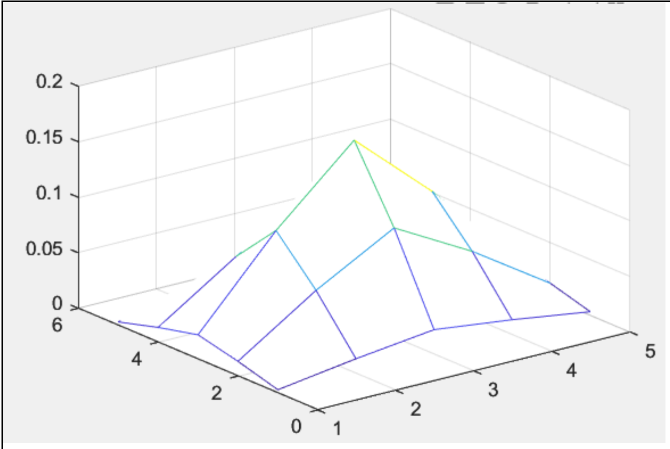
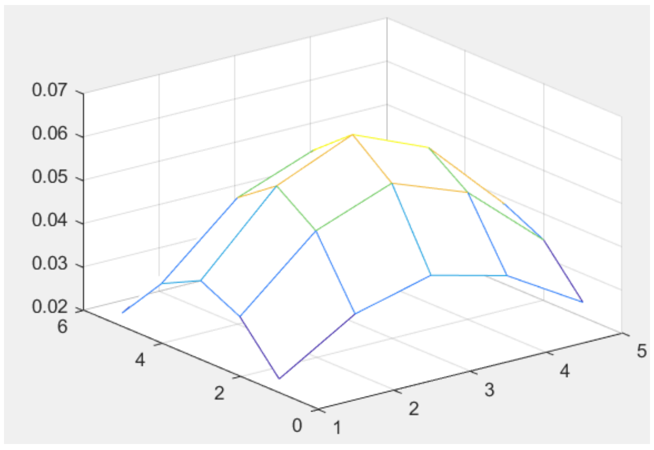
	

Figure 9: 3D graph of filter with sigma=1.0	Figure 10: 3D graph of filter with sigma=2.0

b. Following code displays the “ntu-gn.jpg” image:

```
% b. View ntu-gn.jpg
image_gaussian_noise=imread('D:\NTU Class\CE4003\ntu-gn.jpg');
imshow(image_gaussian_noise);
```

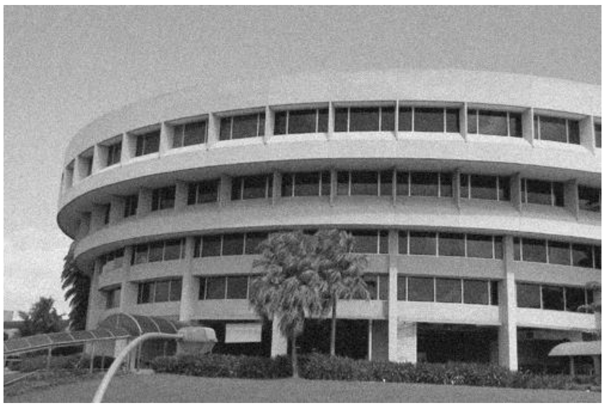


Figure 11: Original image with additive Gaussian noise

We can validate that this image has additive Gaussian noise by using a simple approach of analysing the intensity histogram of the output image.

```
imhist(image_gaussian_noise);
```

If the noise is of Gaussian type, then the histogram is more likely to look similar to Gaussian probability distribution function.

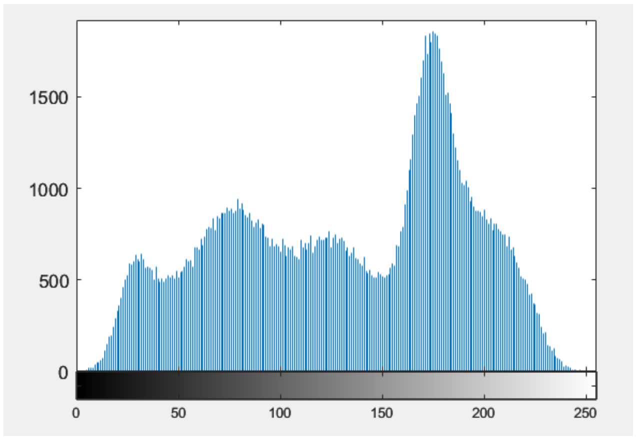


Figure 12: Intensity histogram of original image with additive Gaussian noise

c. The following code is used to filter image with additive gaussian noise and display the resulting image:

```
% c. Linear filtering
P5_1=uint8(conv2(image_gaussian_noise, filter1));
imshow(P5_1);
P5_2=uint8(conv2(image_gaussian_noise, filter2));
imshow(P5_2);
```

We can observe the following from the resultant image after applying filters:

1. Using a filter with a higher sigma value will remove more noise.
2. Higher sigma value makes the image appear more blur.
3. To summarize, a filter with higher coefficient of sigma is more effective in filtering Gaussian noise but it also makes the image appear more blur.

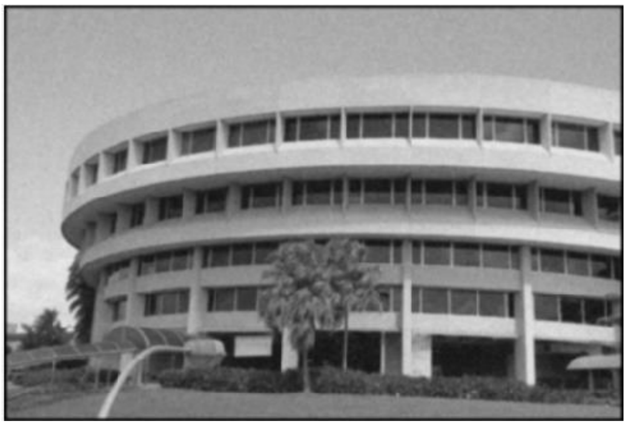
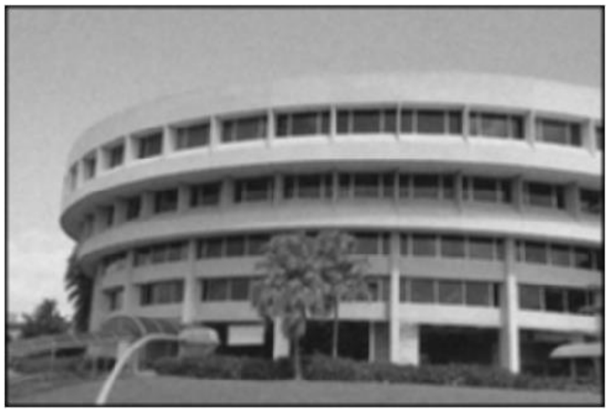
	

Figure 13: Resultant image after applying linear filter with sigma=1.0	Figure 14: Resultant image after applying linear filter with sigma=2.0

d. Following code is used to view the image:

```
% d. View image with pseckle noise
image_speckle_noise=imread('D:\NTU Class\CE4003\ntu-sp.jpg');
imshow(image_speckle_noise);
```

It displays the following image:

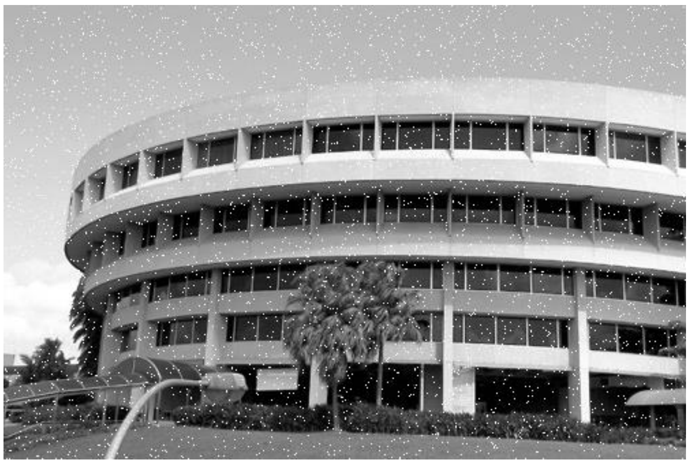


Figure 15: Original image with additive speckle noise

e. The following code is used to filter image with additive speckle noise and display the resulting image:

```
%e. Repeat step con image_speckle_noise
P6_1=uint8(conv2(image_speckle_noise, filter1));
imshow(P6_1);
P6_2=uint8(conv2(image_speckle_noise, filter2));
imshow(P6_2);
```

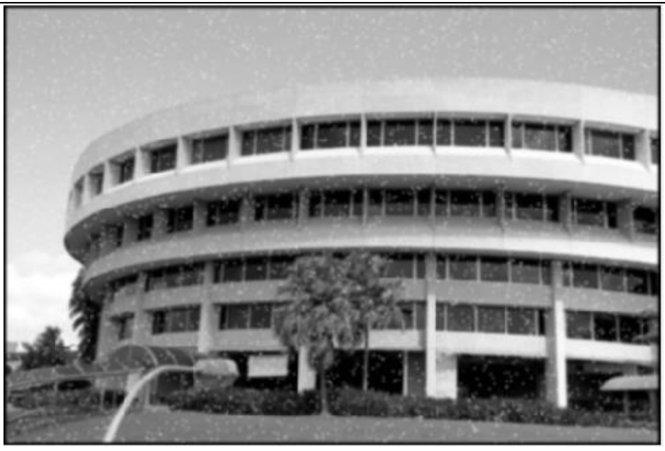
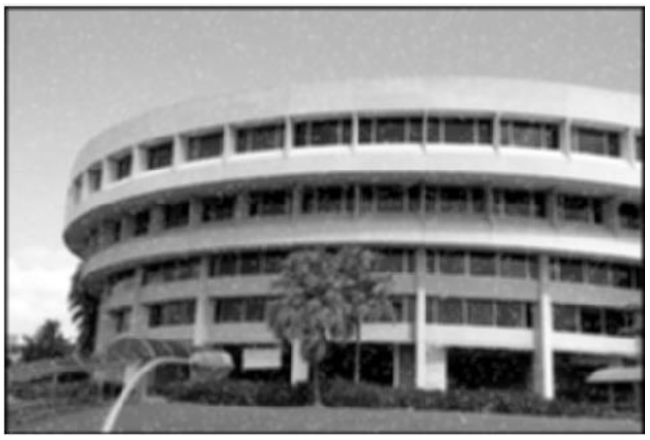
	

Figure 16: Resultant image after applying linear filter with sigma=1.0	Figure 17: Resultant image after applying linear filter with sigma=1.0

The resultant image shows that the filters do not remove speckle noise as well as they remove gaussian noise as we can still see the speckle noise distinctly even with the filter with the higher sigma value.

Additionally, using filter with a higher sigma value will remove more noise. However, it makes the image appear more blur. This observation is like the resultant image after applying these filters on image with additive gaussian noise.

Hence, these filters are better at removing gaussian noise than speckle noise.

f. Source Code:

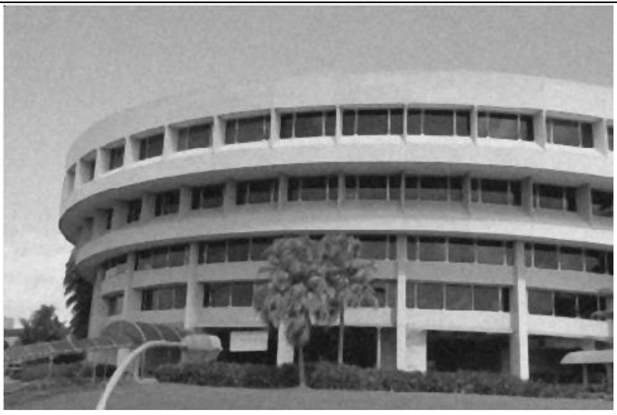
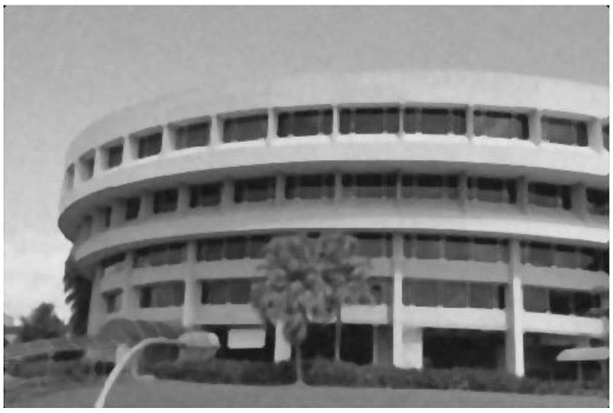
```
%------------------2.3---------------------%
% a. Generate filters
% (i) Y and X-dimensions are 5 and σ = 1.0
sig1=1.0;
% (ii) Y and X-dimensions are 5 and σ = 2.0
sig2=2.0;

dimension=5;
range=-floor(dimension/2):floor(dimension/2);
[X, Y]=meshgrid(range,range);

filter1=h(X,Y,sig1);
filter2=h(X,Y,sig2);

%Normalizing filters
filter1=filter1/sum(filter1(:));
filter2=filter2/sum(filter2(:));

%Display filters
mesh(filter1)
mesh(filter2)


% b. View ntu-gn.jpg
image_gaussian_noise=imread('D:\NTU Class\CE4003\ntu-gn.jpg');
imshow(image_gaussian_noise);
imhist(image_gaussian_noise);

% c. Linear filtering
P5_1=uint8(conv2(image_gaussian_noise, filter1));
imshow(P5_1);
P5_2=uint8(conv2(image_gaussian_noise, filter2));
imshow(P5_2);

% d. View image with pseckle noise
image_speckle_noise=imread('D:\NTU Class\CE4003\ntu-sp.jpg');
imshow(image_speckle_noise);

%e. Repeat step con image_speckle_noise
P6_1=uint8(conv2(image_speckle_noise, filter1));
imshow(P6_1);
P6_2=uint8(conv2(image_speckle_noise, filter2));
imshow(P6_2);
```

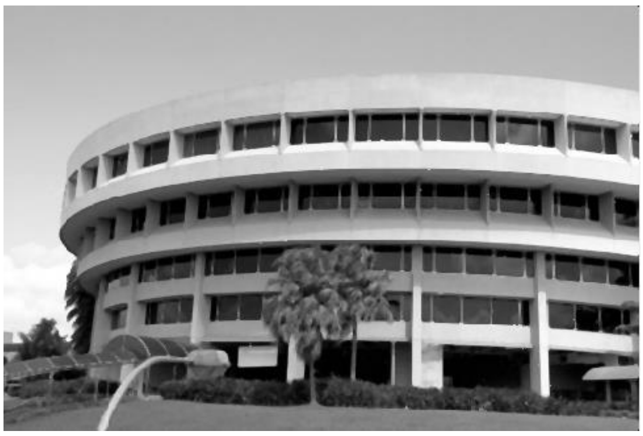
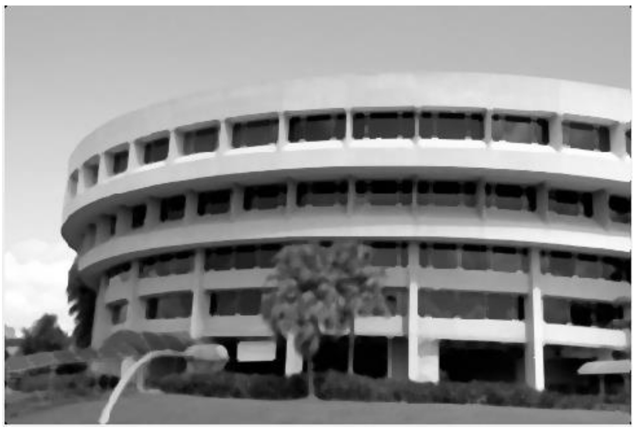
2.4 Median Filtering:

To perform median filtering with size 3x3 filter and 5x5 filter, and display the resulting image, the following code is used:

```
%-----------------2.4--------------------%
% Median filtering of image with gaussian noise
filter_3_3= [3 3];
P7_1=uint8(medfilt2(image_gaussian_noise, filter_3_3));
imshow(P7_1);
filter_5_5= [5 5];
P7_2=uint8(medfilt2(image_gaussian_noise, filter_5_5));
imshow(P7_2);

% Median filtering of image with speckle noise
P8_1=uint8(medfilt2(image_speckle_noise, filter_3_3));
imshow(P8_1);
P8_2=uint8(medfilt2(image_speckle_noise, filter_5_5));
imshow(P8_2);
```

The following images are generated after performing median filtering:

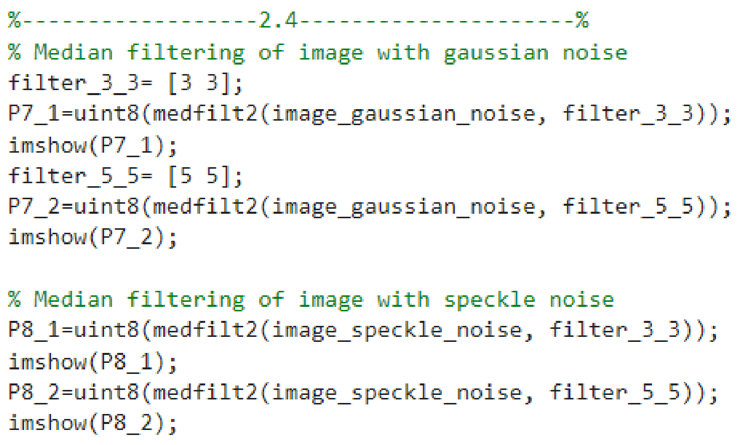
	Figure 18: Image with additive gaussian noise that has undergone median filtering with median filter of size 3x3
	Figure 19: Image with additive gaussian noise that has undergone median filtering with median filter of size 5x5

	Figure 20: Image with additive speckle noise that has undergone median filtering with median filter of size 3x3
	Figure 21: Image with additive speckle noise that has undergone median filtering with median filter of size 5x5

From the results above, it can be observed that median filtering is better at filtering speckle noise than Gaussian noise.

In addition, a median filter of size 3x3 can remove speckle noise better than the linear filters (of size 3x3 and 5x5) used in Gaussian filtering.

f. Source Code:



```
%------------------2.4---------------------%
% Median filtering of image with gaussian noise
filter_3_3= [3 3];
P7_1=uint8(medfilt2(image_gaussian_noise, filter_3_3));
imshow(P7_1);
filter_5_5= [5 5];
P7_2=uint8(medfilt2(image_gaussian_noise, filter_5_5));
imshow(P7_2);

% Median filtering of image with speckle noise
P8_1=uint8(medfilt2(image_speckle_noise, filter_3_3));
imshow(P8_1);
P8_2=uint8(medfilt2(image_speckle_noise, filter_5_5));
imshow(P8_2);
```

2.5 Suppressing Noise Interference Patterns:

a. The following code is used to display “pck-int.jpg” image:

```
%-----------------2.5--------------------%
% a. Display pck-int.jpg
Pck_Int=imread('D:\NTU Class\CE4003\pck-int.jpg');
imshow(Pck_Int);
```

The image displayed is as follows:

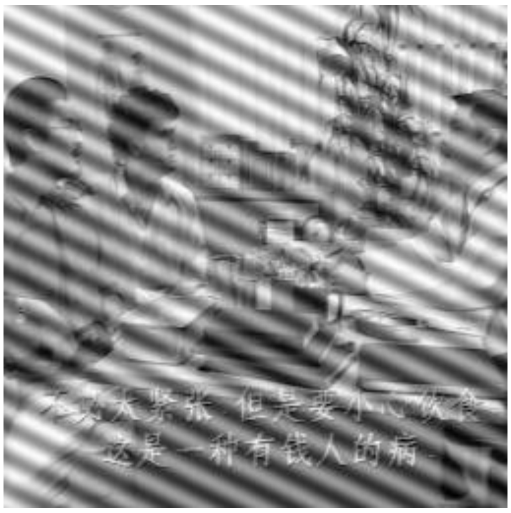
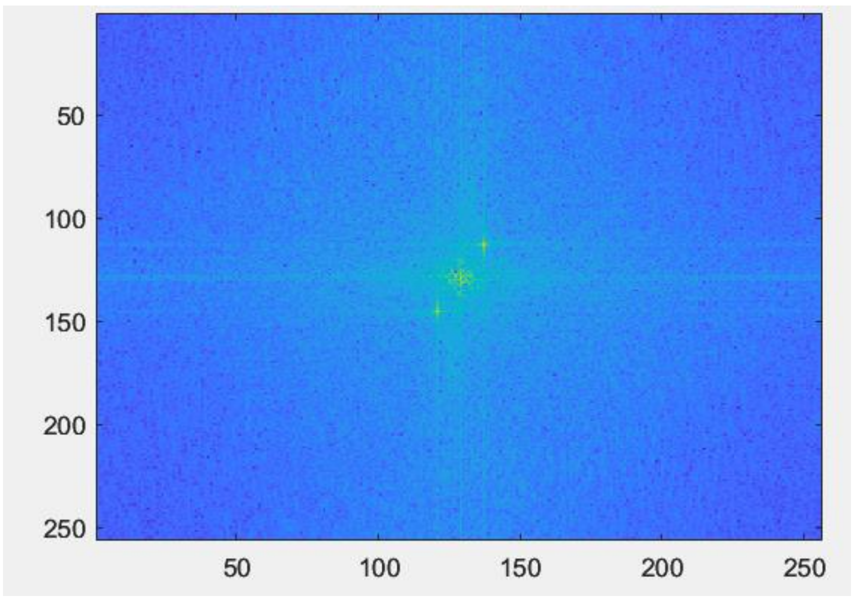


Figure 22: pck-int.jpg with dominant interference pattern

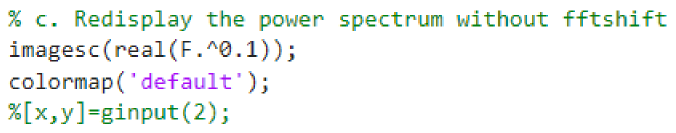
b. The following code is used to obtain the Fourier transform of the image using *fft2*, the power spectrum and display the resultant power spectrum:

```
% b. Obtain the Fourier transform F of the image using fft2, and subsequently compute the power spectrum S
F=fft2(Pck_Int);
% S=abs(F).^2 / length(Pck_Int);
imagesc(fftshift(real(F.^0.1)));
colormap('default');
```

Figure 23: Power Spectrum after fftshift of original image



c. Code below is used to redisplay the power spectrum without fftshift:



```
% c. Redisplay the power spectrum without fftshift
imagesc(real(F.^0.1));
colormap('default');
%[x,y]=ginput(2);
```

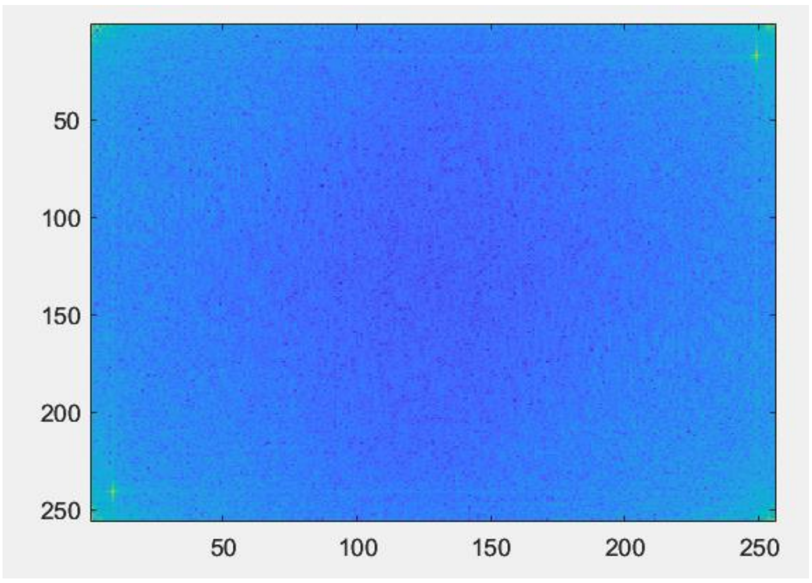
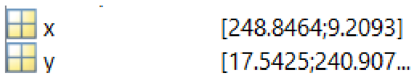


Figure 24: Power Spectrum of original image without fftshift

From Figure 24, we use *ginput* function to obtain the following peak points:

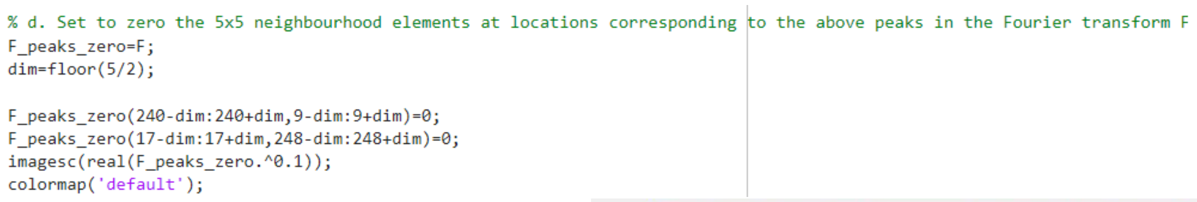


```
x          [248.8464;9.2093]
y          [17.5425;240.907...
```

Hence,

(y , x) = (17.54, 248.84) and (240.90,9.20)

d. The following code sets to zero the 5x5 neighbourhood elements at locations corresponding to peaks in the Fourier transform and compute the power spectrum:



```
% d. Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks in the Fourier transform F
F_peaks_zero=F;
dim=floor(5/2);

F_peaks_zero(240-dim:240+dim,9-dim:9+dim)=0;
F_peaks_zero(17-dim:17+dim,248-dim:248+dim)=0;
imagesc(real(F_peaks_zero.^0.1));
colormap('default');
```

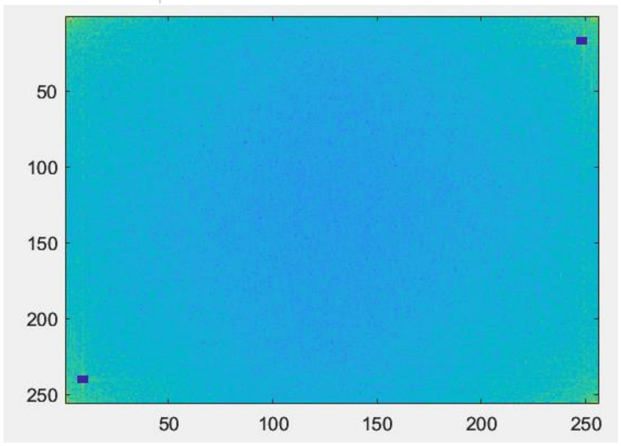


Figure 25: Power Spectrum with 5x5 the neighbourhood elements at locations corresponding to peak set to zero

Additional point to notice would be that the interference pattern frequency was in the top right and bottom left edge of the graph due to Euler's identity. By performing *fftshift*, the low frequency component of the image(DC component of the image) will be at the centre of the image and frequency will go higher as it approaches the edge of the image.

e. The code below computes the Inverse Fourier transform using *ifft2* and displays the resultant image:

```
% e. Compute the inverse Fourier transform using ifft2 and display the resultant image
F_peaks_zero_inversefft=uint8(ifft2(F_peaks_zero));
imshow(F_peaks_zero_inversefft);
```

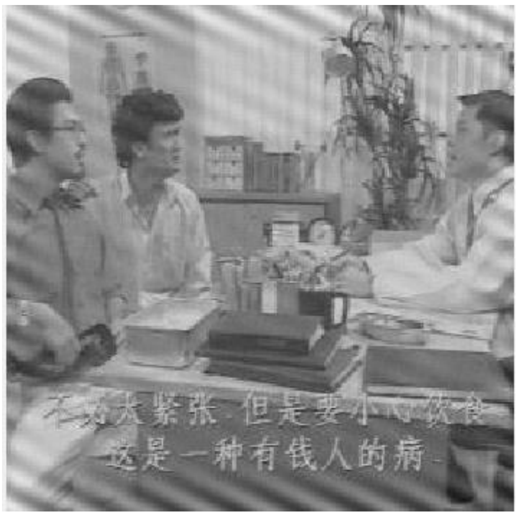


Figure 26: Image after Inverse Fourier transform

Figure 26 shows an improvement when compared to the original image. The image looks better, and the intensity of the original interference pattern has been significantly reduced, particularly around central area of the image.

Since, there are still interference frequency shown in the spatial domain, we can further improve the image by zeroing the frequency, by using the following code:

```
% Improvement
F_peaks_zero2=F;
F_peaks_zero2(240-dim:240+dim,9-dim:9+dim)=0;
F_peaks_zero2(17-dim:17+dim,248-dim:248+dim)=0;

%Extending the zero to edge
F_peaks_zero2(:,9)=0;
F_peaks_zero2(241,:)=0;
F_peaks_zero2(:,249)=0;
F_peaks_zero2(17,:)=0;

imagesc(real(F_peaks_zero2.^0.1))
F_peaks_zero2_inversefft = uint8(ifft2(F_peaks_zero2));
imshow(real(F_peaks_zero2_inversefft))
```

The resultant image and its power spectrum are shown below:

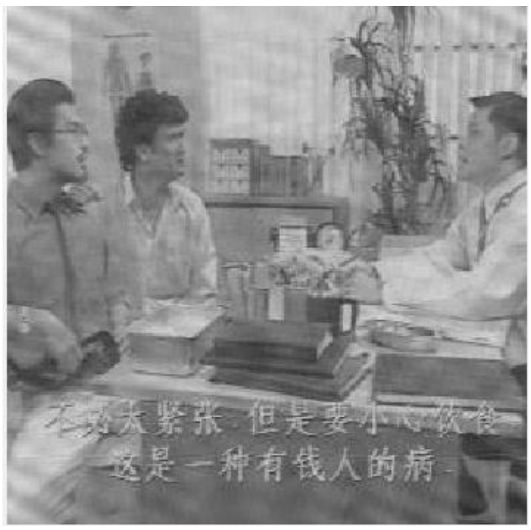


Figure 27: Resulting image after Improvement

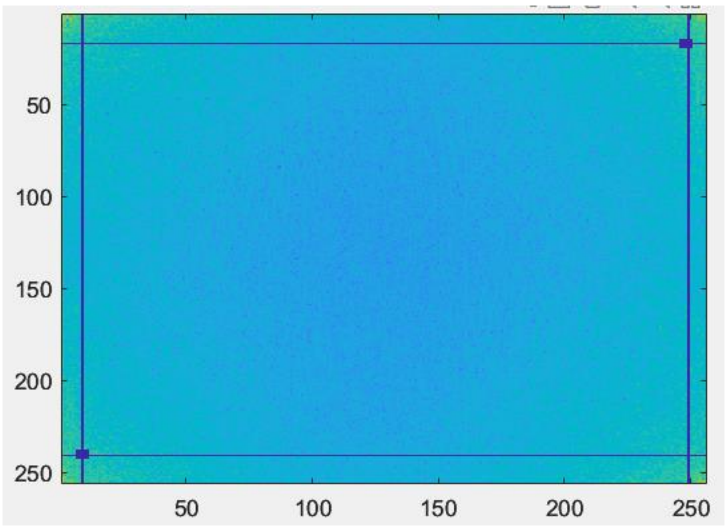


Figure 28: Power spectrum of the resulting image

The image looks much better, and intensity of the original interference pattern have been reduced further. However, the contrast of the image has been reduced and the image has become slightly blurred. To fix this, we can perform contrast stretching with the following code:

```
F_peaks_zero2_minVal=double(min(real(F_peaks_zero2_inversefft(:))));
F_peaks_zero2_maxVal=double(max(real(F_peaks_zero2_inversefft(:))));
subt=imsubtract(real(F_peaks_zero2), F_peaks_zero2_minVal);
Final_F_peaks_zero2=immultiply(subt, 255/(F_peaks_zero2_maxVal-F_peaks_zero2_minVal));
```

The final image obtained is:

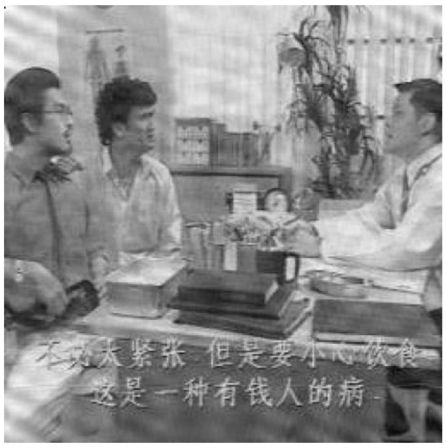


Figure 29: Final image performing contrast stretching and zeroing frequency

f. First, we import the image and check if its rgb or grayscale:

```
% f. Primate caged
primate=imread('D:\NTU Class\CE4003\primate-caged.jpg');
whos primate
primate=rgb2gray(primate);
imshow(primate);
```

It's a rgb image, hence we convert it to grayscale.

The power spectrum after *fftshift* of the image can be seen by the following code:

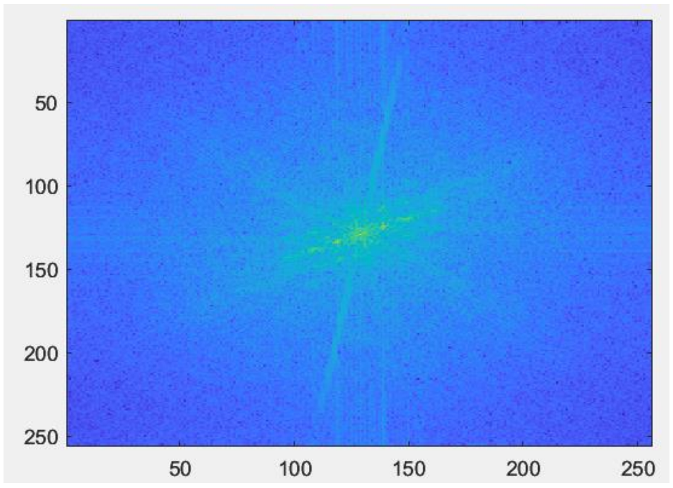


Figure 31: Power spectrum of 'primate-caged' after fftshift

The power spectrum without fftshift is as follows:

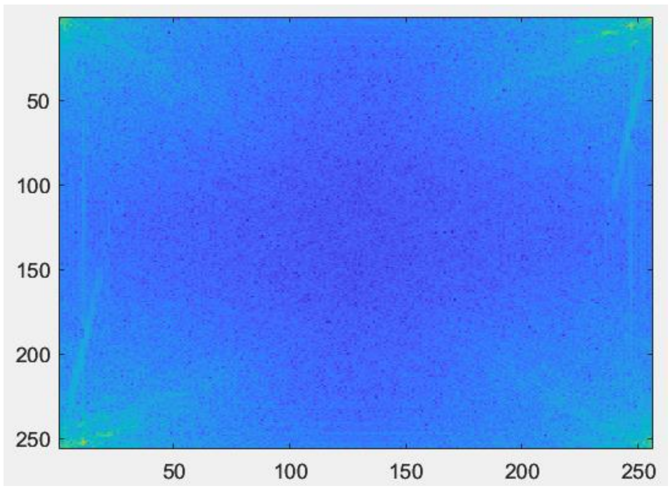


Figure 32: Power spectrum of 'primate-caged' without fftshift

From Figure 32, the coordinates of the peaks can be found by using the *ginput* function:

```
primate_fourier_peak_zero=primate_fourier;
imagesc(real(primate_fourier_peak_zero.^0.1));
% [x,y]=ginput(4);
```

The coordinates are:

(y , x) = (21,247), (11, 252), (237, 9), and (246, 6)

Furthermore, we set to zero the 5x5 neighbourhood elements at locations corresponding to peaks in the Fourier Transform, compute the power spectrum and display the resultant image, using the following code:

```
% (y,x)=(21,247), (11, 252), (237, 9), (246, 6)
y1=21; x1=247;
y2=11; x2=252;
y3=237; x3=9;
y4=246; x4=6;

primate_fourier_peak_zero(x1-3:x1+3,y1-3:y1+3) = 0;
primate_fourier_peak_zero(x2-3:x2+3,y2-3:y2+3) = 0;
primate_fourier_peak_zero(x3-3:x3+3,y3-3:y3+3) = 0;
primate_fourier_peak_zero(x4-3:x4+3,y4-3:y4+3) = 0;
imagesc(real(primate_fourier_peak_zero.^0.1));

primate_inverse_fourier_peak_zero=uint8(ifft2(primate_fourier_peak_zero));
imshow(real(primate_inverse_fourier_peak_zero));
```

The corresponding power spectrum and final image are as follows:

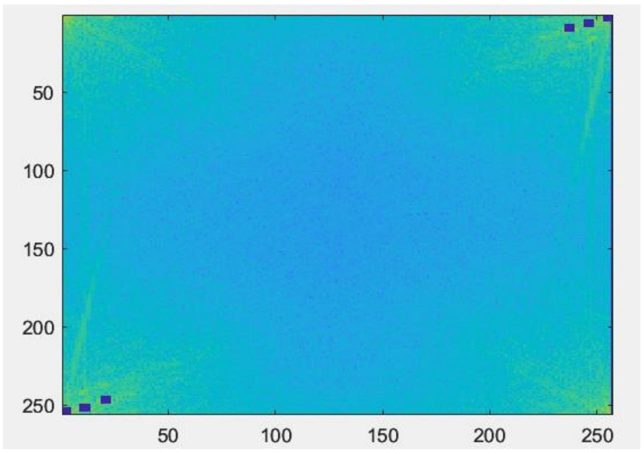


Figure 33: Power spectrum with 5x5 neighbourhood elements at locations corresponding to peaks set to zero

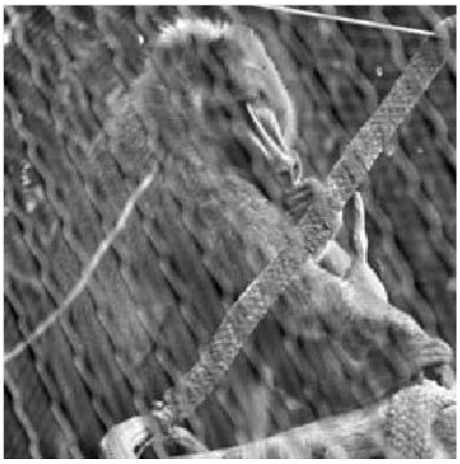
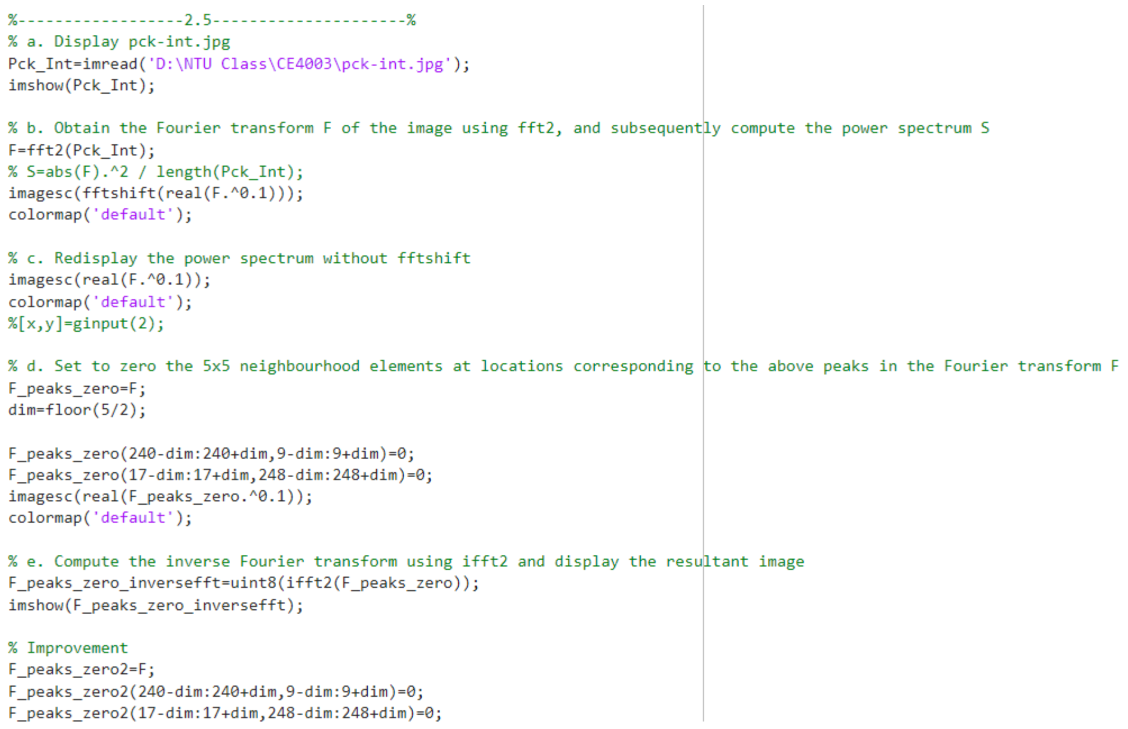


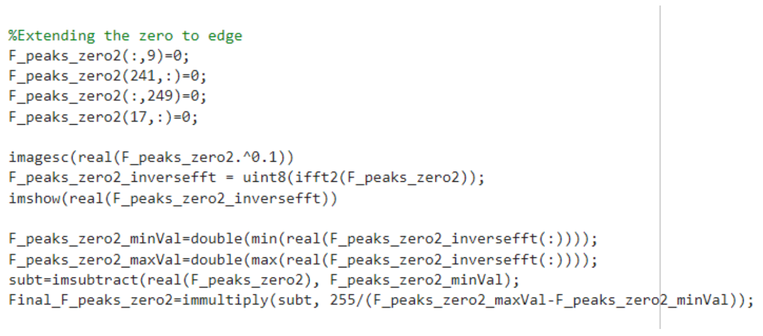
Figure 34: Final image

As observed, the cage is not removed properly, though the case seems blurred out. A possible explanation for this would be that the cage is not actually a noise but a part of the picture itself.

g. Source Code for processing pck-int.jpg:

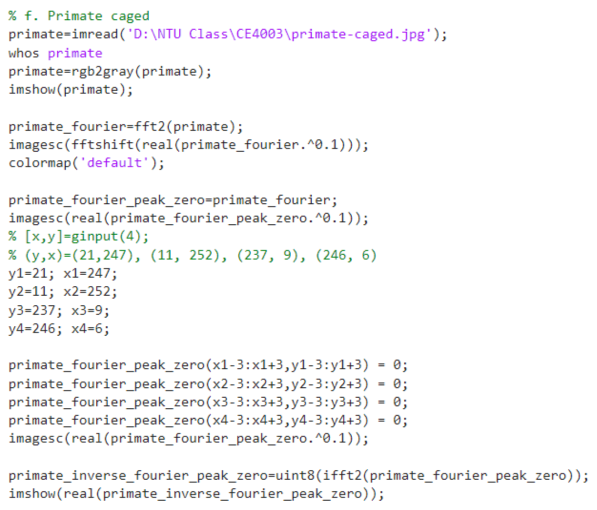


```
%------------------2.5---------------------%
% a. Display pck-int.jpg
Pck_Int=imread('D:\NTU Class\CE4003\pck-int.jpg');
imshow(Pck_Int);

% b. Obtain the Fourier transform F of the image using fft2, and subsequently compute the power spectrum S
F=fft2(Pck_Int);
% S=abs(F).^2 / length(Pck_Int);
imagesc(fftshift(real(F.^0.1)));
colormap('default');

% c. Redisplay the power spectrum without fftshift
imagesc(real(F.^0.1));
colormap('default');
%[x,y]=ginput(2);

% d. Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks in the Fourier transform F
F_peaks_zero=F;
dim=floor(5/2);

F_peaks_zero(240-dim:240+dim,9-dim:9+dim)=0;
F_peaks_zero(17-dim:17+dim,248-dim:248+dim)=0;
imagesc(real(F_peaks_zero.^0.1));
colormap('default');

% e. Compute the inverse Fourier transform using ifft2 and display the resultant image
F_peaks_zero_inversefft=uint8(ifft2(F_peaks_zero));
imshow(F_peaks_zero_inversefft);

% Improvement
F_peaks_zero2=F;
F_peaks_zero2(240-dim:240+dim,9-dim:9+dim)=0;
F_peaks_zero2(17-dim:17+dim,248-dim:248+dim)=0;
```



```
%Extending the zero to edge
F_peaks_zero2(:,9)=0;
F_peaks_zero2(241,:)=0;
F_peaks_zero2(:,249)=0;
F_peaks_zero2(17,:)=0;

imagesc(real(F_peaks_zero2.^0.1))
F_peaks_zero2_inversefft = uint8(ifft2(F_peaks_zero2));
imshow(real(F_peaks_zero2_inversefft))

F_peaks_zero2_minVal=double(min(real(F_peaks_zero2_inversefft(:))));
F_peaks_zero2_maxVal=double(max(real(F_peaks_zero2_inversefft(:))));
subt=imsubtract(real(F_peaks_zero2), F_peaks_zero2_minVal);
Final_F_peaks_zero2=immultiply(subt, 255/(F_peaks_zero2_maxVal-F_peaks_zero2_minVal));
```

Source Code for processing primate-caged.jpg:

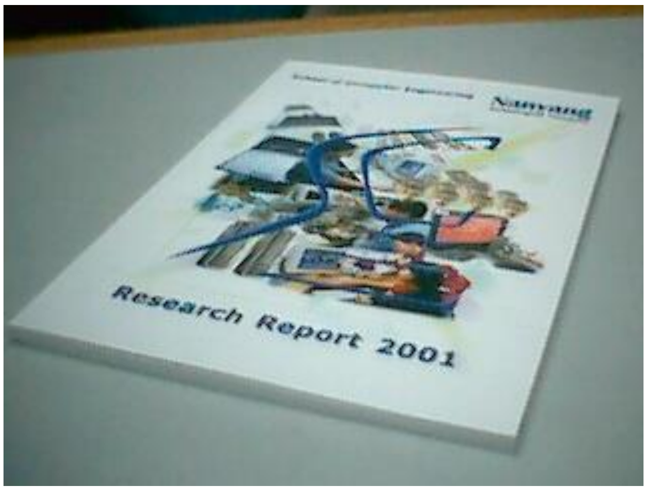


```
% f. Primate caged
primate=imread('D:\NTU Class\CE4003\primate-caged.jpg');
whos primate
primate=rgb2gray(primate);
imshow(primate);

primate_fourier=fft2(primate);
imagesc(fftshift(real(primate_fourier.^0.1)));
colormap('default');

primate_fourier_peak_zero=primate_fourier;
imagesc(real(primate_fourier_peak_zero.^0.1));
% [x,y]=ginput(4);
% (y,x)=(21,247), (11, 252), (237, 9), (246, 6)
y1=21; x1=247;
y2=11; x2=252;
y3=237; x3=9;
y4=246; x4=6;

primate_fourier_peak_zero(x1-3:x1+3,y1-3:y1+3) = 0;
primate_fourier_peak_zero(x2-3:x2+3,y2-3:y2+3) = 0;
primate_fourier_peak_zero(x3-3:x3+3,y3-3:y3+3) = 0;
primate_fourier_peak_zero(x4-3:x4+3,y4-3:y4+3) = 0;
imagesc(real(primate_fourier_peak_zero.^0.1));

primate_inverse_fourier_peak_zero=uint8(ifft2(primate_fourier_peak_zero));
imshow(real(primate_inverse_fourier_peak_zero));
```

2.5 Undoing Perspective Distortion of Planar Surface:

a. “book.jpg” is displayed as follows:

```
%-----------------2.6--------------------%
% a. Display book.jpg
P=imread('D:\NTU Class\CE4003\book.jpg');
whos P
imshow(P);
```



b. Using the ginput function:

```
% b. Find out the location of 4 corners of the book and store in x and y

%[X, Y] = ginput(4);
ximvec = [0 210 210 0];
yimvec = [0 0 297 297];
```

The following coordinates are obtained,

X=[144.1827; 309.3952; 256.3640; 4.1261]

Y=[28.0354; 47.7521; 215.0042; 159.9334]

c. The following code is used by employing the matrix structure given in the lab manual to estimate the projective transformation parameters:

```
% c. Projective transformations on image
v = [ximvec(1); yimvec(1); ximvec(2); yimvec(2); ximvec(3); yimvec(3); ximvec(4); yimvec(4)];
A = [
[X(1), Y(1), 1, 0, 0, 0, -ximvec(1) * X(1), -ximvec(1) * Y(1)];
[0, 0, 0, X(1), Y(1), 1, -yimvec(1) * X(1), -yimvec(1) * Y(1)];
[X(2), Y(2), 1, 0, 0, 0, -ximvec(2) * X(2), -ximvec(2) * Y(2)];
[0, 0, 0, X(2), Y(2), 1, -yimvec(2) * X(2), -yimvec(2) * Y(2)];
[X(3), Y(3), 1, 0, 0, 0, -ximvec(3) * X(3), -ximvec(3) * Y(3)];
[0, 0, 0, X(3), Y(3), 1, -yimvec(3) * X(3), -yimvec(3) * Y(3)];
[X(4), Y(4), 1, 0, 0, 0, -ximvec(4) * X(4), -ximvec(4) * Y(4)];
[0, 0, 0, X(4), Y(4), 1, -yimvec(4) * X(4), -yimvec(4) * Y(4)];
];
```

We then calculate the matrix “u” using the given matric formula and display it:

```
u = A \ v;
disp(u);
```

The “u” matrix obtained is:

```
    1.4368
    1.5257
 -249.9355
   -0.4341
    3.6374
  -39.3877
    0.0001
    0.0051
```

Following code is used to reshape the projective transformation parameters to the normal matrix form by using the given matrix formula:

```
U = reshape([u;1], 3, 3)';
disp(U);
```

The “U” matrix obtained is:

```
    1.4368    1.5257 -249.9355
   -0.4341    3.6374  -39.3877
    0.0001    0.0051    1.0000
```

Finally, inorder to verify that the Matrix U is correct, we employ the given formula in the code:

```
w = U * [X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
disp(w);
```

The output obtained after this operation is as follows:

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

Since, the “U” matrix is correct, the transformation returns the 4 corners of the desired image.

d. Employing the given instruction, we warp the image in the following way:

```
% d. Warping the image
T = maketform('projective', U');
P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
```

e. The image is shown by using the following code:

```
% e. Display image
imshow(P2);
```

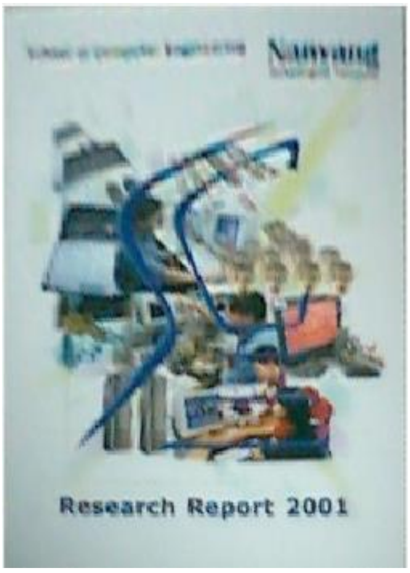


Figure 35: Image of book after transformations

Above transformations were successful in getting the desired image. The quality of the image is better for the parts that are closer to the camera. Hence, the top part of the book is blurry, particularly the top left part that is furthest from the camera. This can be observed as the text on the top left corner of the book is difficult to read and it was difficult to read in the original image as well.

f. Following code was used to highlight the "big rectangular pink area, which seems to be a computer screen, at the middle place between 'Nanyang' and '2001' ".

```
% f. Identify the pink area
red=P2(:,:,1);
green=P2(:,:,2);
blue=P2(:,:,3);

output=red>200 & red<225 & green<150 & green>0 & blue<160 & blue>0;
imshow(output);

output2=imfill(output,'holes');
output3=bwmorph(output2,'dilate',3);
output3=imfill(output3,'holes');
imshow(output3);

imBoth=imoverlay(P2,out3,'black');
imshow(imBoth);
```

(Notations will be used according to the code displayed above)

I employed a RGB filter to filter out the pink area mentioned in the question.

imoverlay is used to highlight that specific part of the image which has pink colour on the original image (Figure 35).

Following explains the procedure to obtain the highlighted part:

Here, P2 is a RGB image, hence, to be able to filter out a specific colour, we must extract the individual layers of the image plane. By performing P2(:,:,x), we get the mean of all rows and columns in the corresponding image plane. x=1 for red, x=2 for green and x=3 for blue.

Now to highlight the colour “pink” we limit the pixels of every layer to be within a range via brute force to find the most suitable value.

At this stage, we get the following output:

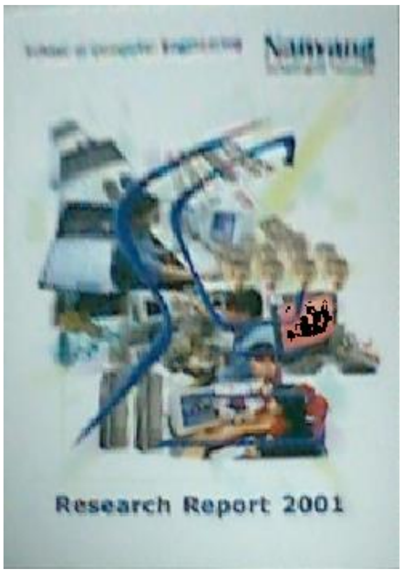


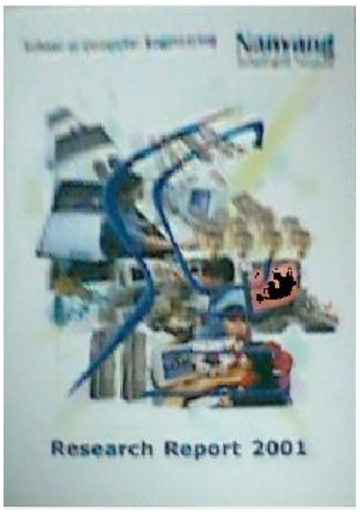
Figure 36: Highlighted pink area without enhancement

Figure 36 shows that the pink area is highlighted with the color black, and thus far we have successfully segmented the pink color in the image, but there are some "holes" in the image, i.e., unfilled/uncovered portions of the pink area. We can perform some enhancement operations to get a much clearer and neater image.

imfill: Fills up the holes

The following output is obtained:

Figure 37: Highlighted pink area after filling holes



As we can see, filling the holes is insufficient to segment the entire pink area.

As a result, we dilate the image with the bwmorph Matlab function. This function is used to apply the morphological operation of image dilation. The function adds pixels to the segmented boundaries. The number '3' represents the number of times the image should be dilated. Furthermore, we use the imfill once more to get a clearer image. The image obtained because of this operation is as follows:

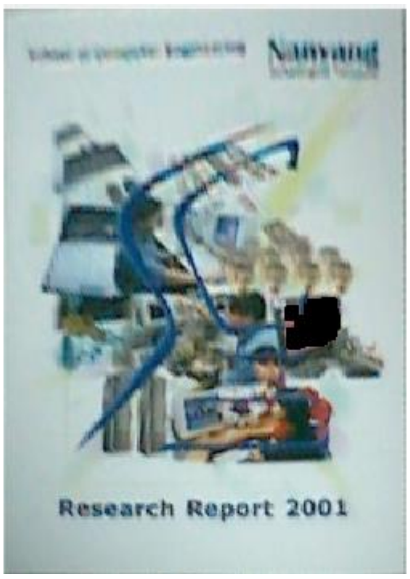
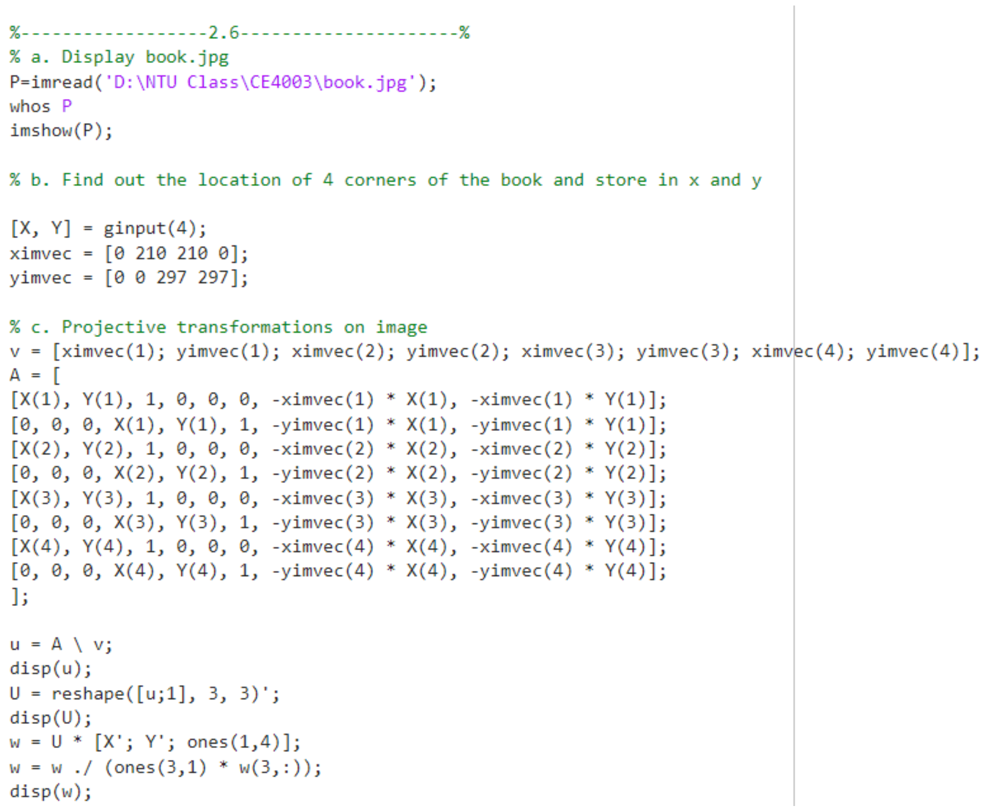


Figure 38: Highlighted pink area after dilating and filling holes

Finally, we get required segmented pink area. The operations performed above were able to highlight the “big rectangular pink area, which seems to be a computer screen, at the middle place between ‘Nanyang’ and ‘2001’ ” as required.

g. Source code:



```
%------------------2.6---------------------%
% a. Display book.jpg
P=imread('D:\NTU Class\CE4003\book.jpg');
whos P
imshow(P);

% b. Find out the location of 4 corners of the book and store in x and y

[X, Y] = ginput(4);
ximvec = [0 210 210 0];
yimvec = [0 0 297 297];

% c. Projective transformations on image
v = [ximvec(1); yimvec(1); ximvec(2); yimvec(2); ximvec(3); yimvec(3); ximvec(4); yimvec(4)];
A = [
[X(1), Y(1), 1, 0, 0, 0, -ximvec(1) * X(1), -ximvec(1) * Y(1)];
[0, 0, 0, X(1), Y(1), 1, -yimvec(1) * X(1), -yimvec(1) * Y(1)];
[X(2), Y(2), 1, 0, 0, 0, -ximvec(2) * X(2), -ximvec(2) * Y(2)];
[0, 0, 0, X(2), Y(2), 1, -yimvec(2) * X(2), -yimvec(2) * Y(2)];
[X(3), Y(3), 1, 0, 0, 0, -ximvec(3) * X(3), -ximvec(3) * Y(3)];
[0, 0, 0, X(3), Y(3), 1, -yimvec(3) * X(3), -yimvec(3) * Y(3)];
[X(4), Y(4), 1, 0, 0, 0, -ximvec(4) * X(4), -ximvec(4) * Y(4)];
[0, 0, 0, X(4), Y(4), 1, -yimvec(4) * X(4), -yimvec(4) * Y(4)];
];

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

```
% d. Warping the image
T = maketform('projective', U');
P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);

% e. Display image
imshow(P2);

% f. Identify the pink area
%Validating that it ias a RGB image
whos(p2)
%Extracting the image layers
red=P2(:,:,1);
green=P2(:,:,2);
blue=P2(:,:,3);

% thresholding the output pixel range
output=red>200 & red<225 & green<150 & green>0 & blue<160 & blue>0;
imshow(output);

% enhancing the image
output2=imfill(output,'holes');
output3=bwmorph(output2,'dilate',3);
output3=imfill(output3,'holes');
imshow(output3);

imBoth=imoverlay(P2,out3,'black');
imshow(imBoth);
```

The end