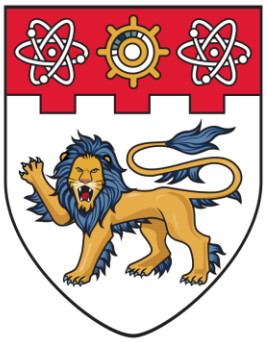

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4003

Lab 1 Report

Name : Chua Zi Jian

Matriculation No.: U2022354J

Date : 2/10/2022

Content

Page

1. Contrast Stretching

Contrast stretching (often called normalization) is a simple image enhancement technique that attempts **to improve the contrast in an image** by 'stretching' the range of intensity values it contains to span a desired range of values, the full range of pixel values that the image type concerned allows

Steps:

a. Input the image into a MATLAB matrix variable by executing:

```
% CZ4003 Lab 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2.1 Contrast Stretching
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% a. Input the image into a MATLAB matrix variable by executing:
%-------------------------------------------------------------------
Pc = imread('mrt-train.jpg');
whos Pc ;
P=rgb2gray(Pc);
whos P;
```

Figure 1.1.1 Code to read image

Result:

```
>> zchua034_CZ4003Lab1
  Name         Size                Bytes  Class    Attributes

  Pc         320x443x3            425280  uint8

  Name         Size                Bytes  Class    Attributes

  P          320x443              141760  uint8
```

Figure 1.1.2 Result of code

From the result, we can see that at first the image matrix is 320 x 443 x 3 which means that there are 3 matrix of R, G, B with dimension of 320 x 443 to indicate respective colours. After we converted it to grayscale there only left with one matrix of 320 x 443 and the bytes of image has also decreased since we only need one matrix to represent a grayscale image.

b. View this image using imshow. Notice that the image has poor contrast. Your initial task will be to investigate different methods for improving the image appearance using point processing. In point processing, the image transformation is carried for each pixel independently of neighbouring pixels.

```
%-----------------------------------------------------------------
% b. View this image using imshow
%-----------------------------------------------------------------
imshow(Pc);
imshow(P);
```

Figure 1.2.1 Code to run

Result:

Figure 1.2.2 P(Original Image)

Figure 1.2.3 Pc(Gray Image)

c. Check the minimum and maximum intensities present in the image:

```
>> min(P(:)), max(P(:))
```

Contrast stretching involves linearly scaling the gray levels such the smallest intensity present in the image maps to 0, and the largest intensity maps to 255.

```
%-----------------------------------------------------------------
% c.Check the minimum and maximum intensities present in the image:
%-----------------------------------------------------------------
% min(P(:)), max(P(:));
r_min = double(min(P(:)));
r_max = double(max(P(:)));
disp("Max Value=")
disp(r_max);
disp("Min Value=")
disp(r_min);
```

Figure 1.3.1 Code to run

Result:

```
Max Value=
   204

Min Value=
    13
```

Figure 1.3.2 Result

We get to know that for the original image(with RGB) the minimum intensity is 13 and maximum intensity is 204.

d. Next, write two lines of MATLAB code to do contrast stretching. Hint: This involves a subtraction operation followed by multiplication operation (note that for images which contain uint8 elements, you will need to use imadd, imsubtract, etc. for the arithmetic instead of the usual operators; alternatively you can convert to a double-valued matrix first using double, but you will need to convert back via uint8 prior to using imshow — see below). Check to see if your final image P2 has the correct minimum and maximum intensities of 0 and 255 respectively by using the min and max commands again.

```
%-------------------------------------------------------------------
% d. Perform contrast stretching
%-------------------------------------------------------------------
% Formula ~ s=(255*(r-rmin)/(rmax-rmin))


P2a(:,:) = imsubtract(P(:,:), r_min);
P2a(:,:) = immultiply(P2a(:,:), im2double(255 / (r_max - r_min)));
%min(P2a(:)), max(P2a(:));
r_min = double(min(P2a(:)));
r_max = double(max(P2a(:)));
disp("Max Value(After Contrast Stretching)=")
disp(r_max);
disp("Min Value(After Contrast Stretching)=")
disp(r_min);
P2a = uint8(P2a);
```

Figure 1.4.1 Code to run

Result:

```
Max Value(After Contrast Stretching)=
   255

Min Value(After Contrast Stretching)=
     0
```

Figure 1.4.2 Result

After contrast stretching, the minimum intensity has become 0 and maximum intensity has become 255 as Contrast Stretching will linearly scale the gray levels such the smallest intensity present in the image maps to 0, and the largest intensity maps to 255.

e. Finally, redisplay the image by

```
>> imshow(P2);
```

Note again that if you choose to convert your byte images to doubles first, you will need to use the uint8 command is necessary to convert the P2 double values back to byte values(Done in part d.). Otherwise, imshow detects P2 to be double and assumes that the intensity range is between 0.0 and 1.0. An alternative is to use the imshow as such: >> imshow(P2,[]); This does automatic contrast stretching of the display but does not affect the input matrix. This allows you to ignore whether you are displaying byte or double-valued images.

Figure 1.5.1 Original Image of Train

Figure 1.5.2 Train Image after Contrast Stretching

From the result, we can see that after contrast stretching, the new image has a better contrast compared to the original image which is what we wanted.

2. Histogram Equalization

Histogram Equalization is a computer image processing technique used **to improve contrast in images** . It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image.

The key idea of where does the k-th bin go:

– Step 1: Ask where is the (k-1)-th bin now

– Step 2: Calculate how many new bins the k-th bin needs, then move over to the last (recall that we cannot split the pixels in the same bin)

Steps:

a. Display the image intensity histogram of P using 10 bins by

>> imhist(P,10);

Next display a histogram with 256 bins. What are the differences?

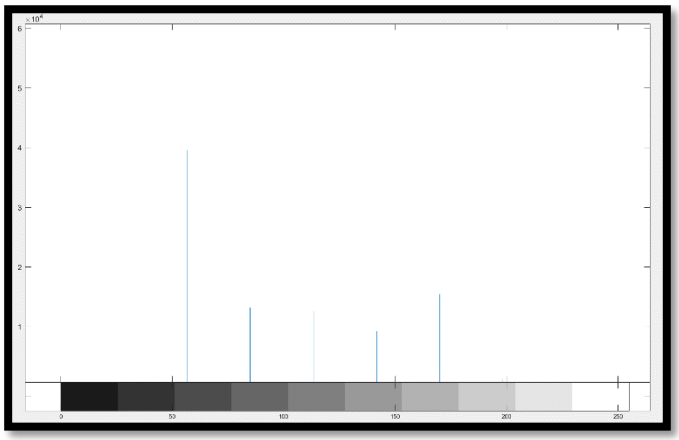

Figure 2.1.1 10 bins Histogram

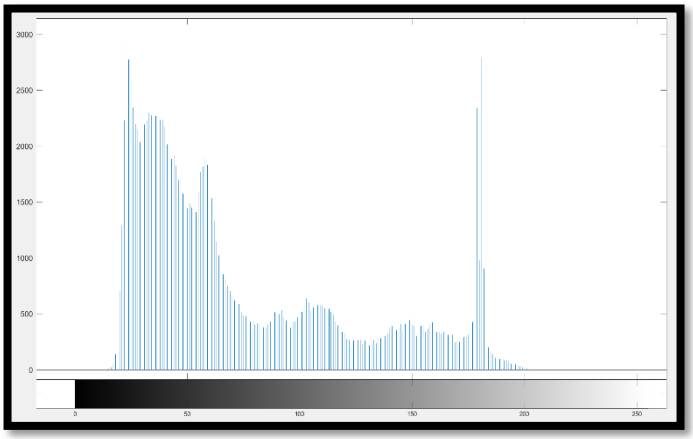

Figure 2.1.2 256bins Histogram

Differences: 256 bins is more detailed than 10 bins histogram

b. Next, carry out histogram equalization as follows:

```
>> P3 = histeq(P,255);
```

Redisplay the histograms for P3 with 10 and 256 bins. Are the histograms equalized?
What are the similarities and differences between the latter two histograms?

Result:

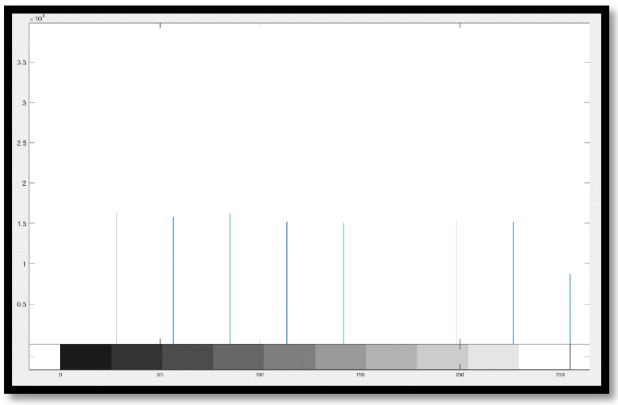

Figure 2.2.1 10 bins Histogram(Equalized)

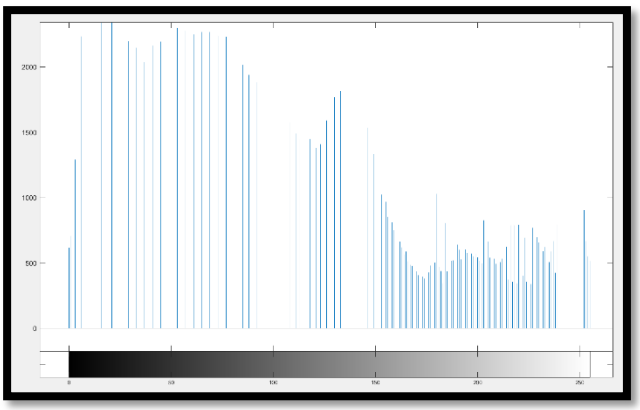

Figure 2.2.2 256 bins Histogram(Equalized)

Answer 1:

Yes, both histogram is equalized but for 256 bins it does not equalize as much as 10 bins histogram

Answer 2:

Similarities: Both histogram gray level is ranged from 0 to 255
Differences: (1) 10 bins histogram is more equalized than 256 bins histogram.
(2) 10 bins histogram frequency is around 1.5*10^4 but 256 bins histogram ranged from around 334 to 2935
(3) 10 bins histogram is evenly spaced out but 256 bins histogram has a higher spaced out from 161 to 239

c. Rerun the histogram equalization on P3. Does the histogram become more uniform? Give suggestions as to why this occurs.

Result:

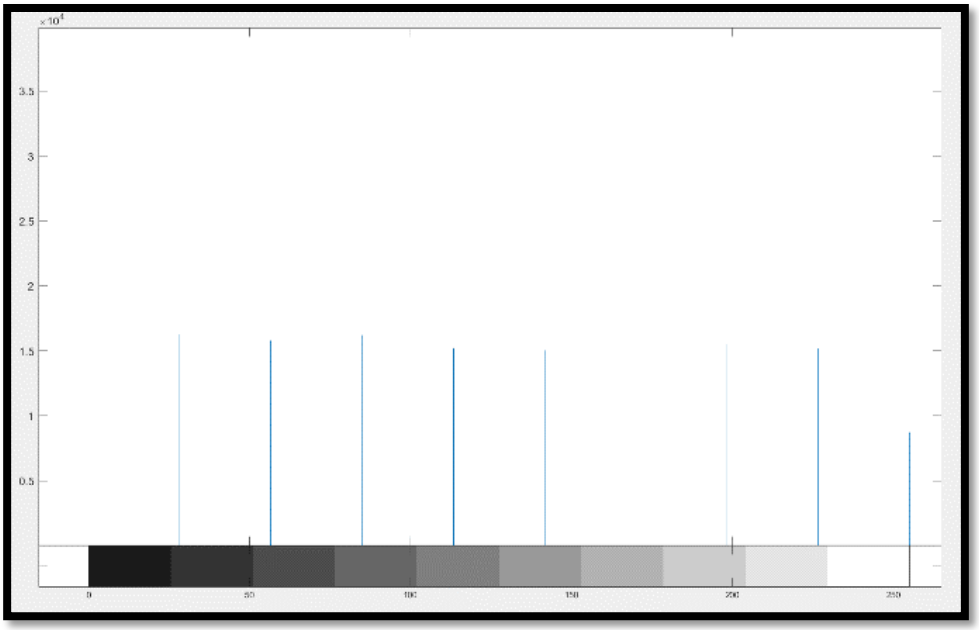

Figure 2.3.1 10 bins Histogram(Rerun Equalized)

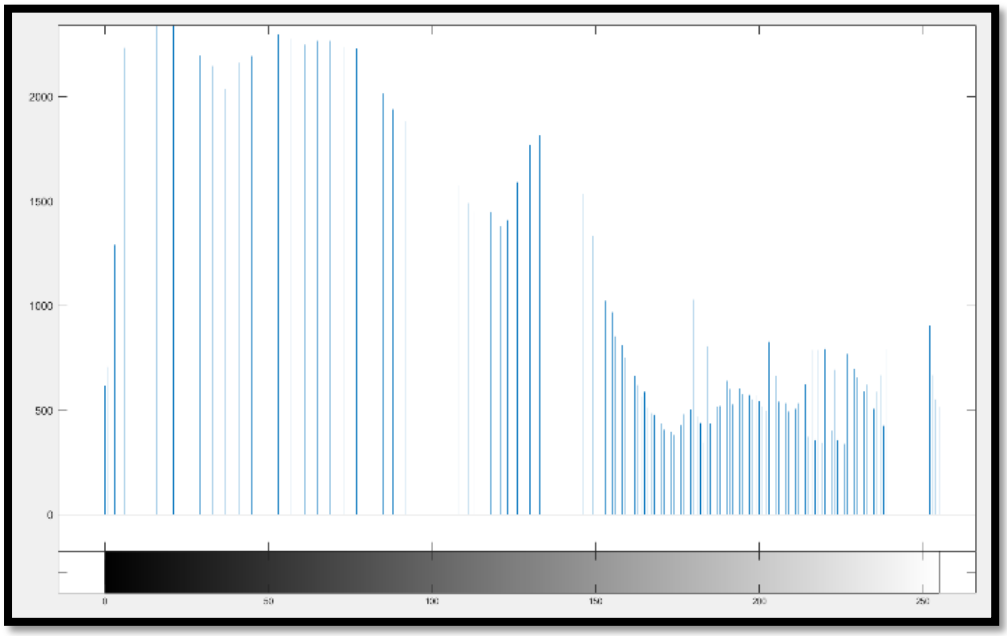

Figure 2.3.2 256 bins Histogram(Rerun Equalized)

Answer :

No, the histogram is the same as before.I think it is due to that the bins have already been assigned to their "designated" area during the first equalization which is by cumulative probability function so rerunning equalization will not have any changes

Checking result image after histogram equalization:

Figure 2.3.3 Original Image

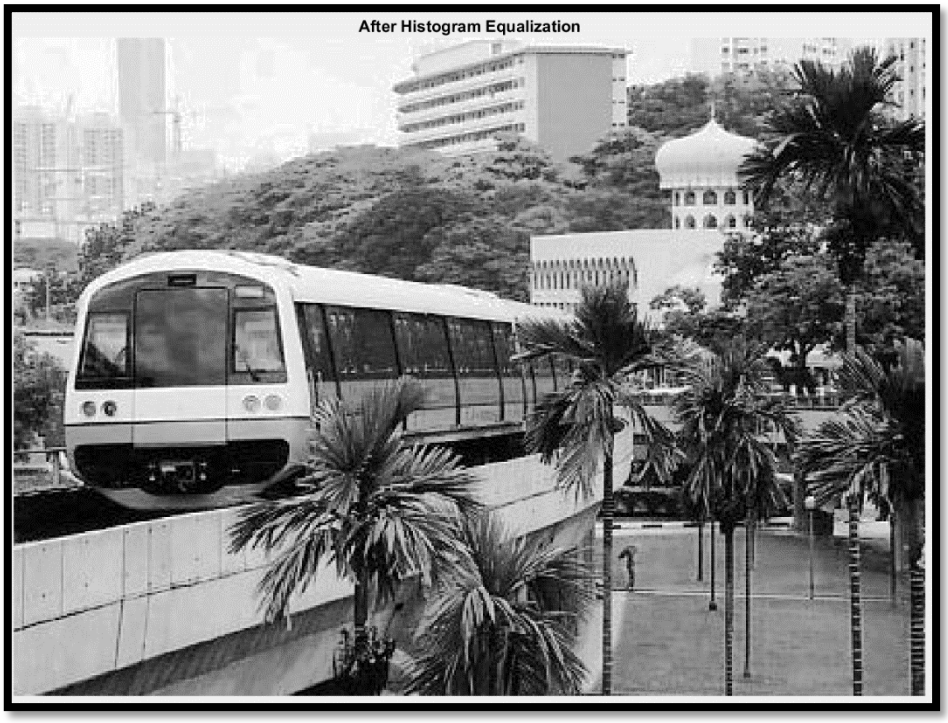

Figure 2.3.4 After Histogram Equalization

Conclusion:

Histogram Equalization is a method that will make the mean brightness of the output image is significantly different from the input image and the overall shape of histogram will change and only reversible if the function for histogram equalization is known whereas contrast stretching will create a contrast that is more similar to the original image and the shape of histogram will remain the same. Which method is better to do contrast stretching depends on what kind of result the user expected to have.

3. Linear Spatial Filtering

Linear spatial filtering is used for smoothening or removing noise on a image, it will **modifies an image f by replacing the value at each pixel with some linear function of the values of nearby pixels**. Moreover, this linear function is assumed to be independent of the pixel's location (i, j), where (i, j) indexes the pixels in f, which is represented as a mr by mc matrix.

In this lab assignment, we are using Gaussian filter for removing noises in picture.

A Gaussian Filter is a low pass filter used for reducing noise (high frequency components) and blurring regions of an image. The filter is implemented as an Odd sized Symmetric Kernel (DIP version of a Matrix) which is passed through each pixel of the Region of Interest to get the desired effect.

Steps:

a. Generate the following filters:

```
%-------------------------------------------------------------------------------------
% a. Generate the following filters:
%-------------------------------------------------------------------------------------
% h(n)=......
h=@(X,Y,std) exp(-(X.^2.+Y.^2)./(std^2.*2))./(2*pi*std^2);
% dimension= ......, -2, -1, 0, 1, 2, ......
dim=5;
x=-(dim-1)/2:(dim-1)/2;y=-(dim-1)/2:(dim-1)/2;
create x,y array
[X,Y] = meshgrid(x,y);


% (i) Y and X-dimensions are 5 and σ = 1.0
h1=h(X,Y,1);
% Normalization
h1_norm=h1./sum(h1,'all');
figure
% 指定值为 0.8 的 FaceAlpha 名称-值对组
mesh(X,Y,h1_norm,'FaceAlpha','0.8')
title('Gaussian Filter (Standard Deviation=1.0)'),xlabel('X'),ylabel('Y'),zlabel('Z');


% (ii) Y and X-dimensions are 5 and σ = 2.0
h2=h(X,Y,2);
% Normalization
h2_norm=h2./sum(h2,'all');
figure
mesh(X,Y,h2_norm,'FaceAlpha','0.8')
title('Gaussian Filter (Standard Deviation=2.0)'),xlabel('X'),ylabel('Y'),zlabel('Z');
```

Figure 3.1.1 Code to run

Gaussian Filter 3D Graphs:

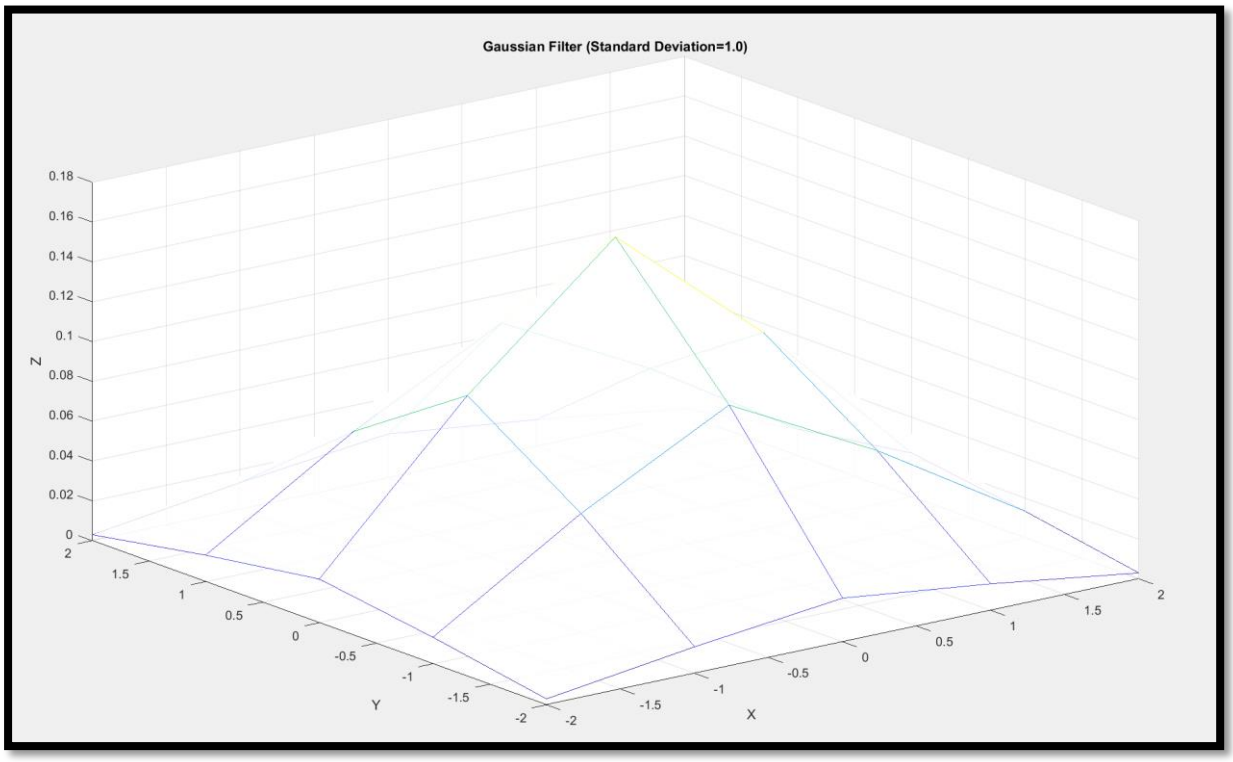

Figure 3.1.2 Gaussian Filter with std=1

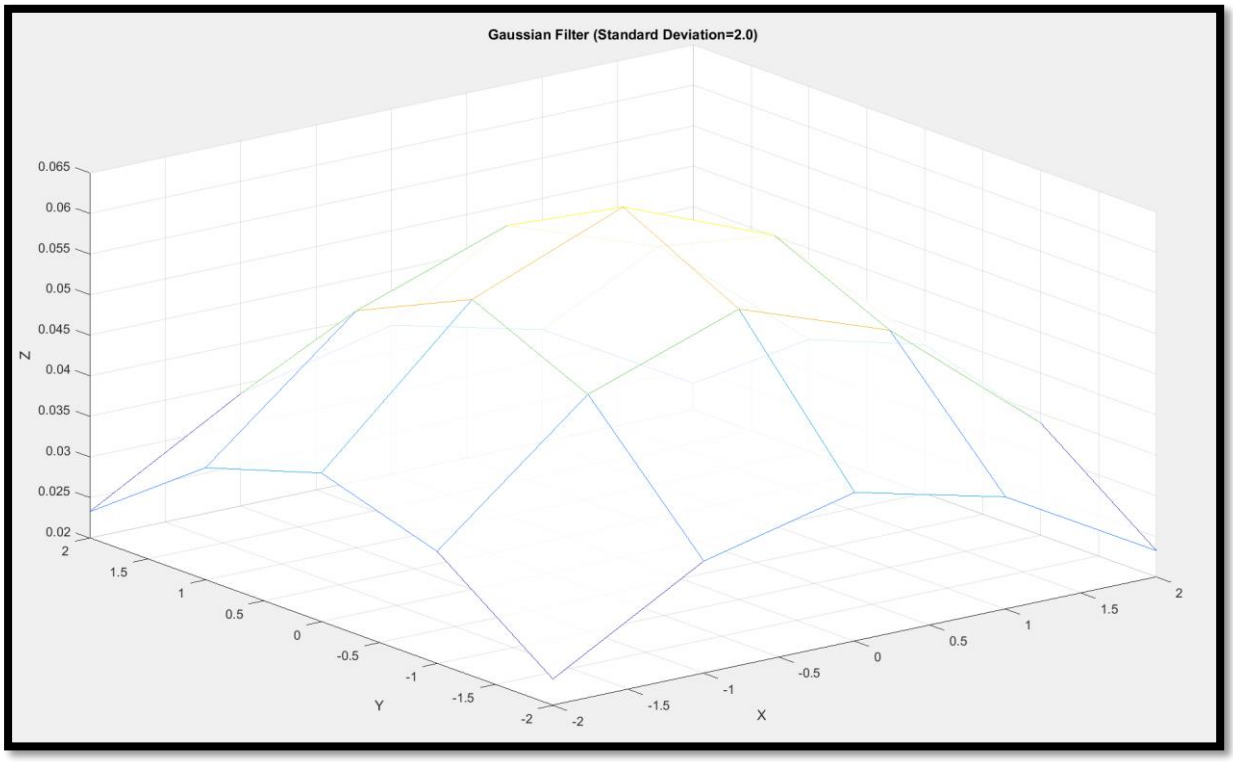

Figure 3.1.3 Gaussian Filter with std=2

b. Download the image ‘ntu-gn.jpg’ from NTULearn and view it. Notice that this image has additive Gaussian noise.

Figure 3.2 NTU Library Picture with Gaussian Noise

c. Filter the image using the linear filters that you have created above using the conv2 function, and display the resultant images. How effective are the filters in removing noise? What are the trade-offs between using either of the two filters, or not filtering the image at all?

```
%--------------------------------------------------------------------------------
% c. Filter the image using the linear filters that you have created above using the conv2
% function, and display the resultant images. How effective are the filters in removing noise?
% What are the trade-offs between using either of the two filters, or not filtering the image at
% all?
%--------------------------------------------------------------------------------

filtered_lib_gn1 = uint8(conv2(lib_gn,h1));
filtered_lib_gn2 = uint8(conv2(lib_gn,h2));

imshow(filtered_lib_gn1);
title('Filtered Library GN using h(n) with std=1');
imshow(filtered_lib_gn2);
title('Filtered Library GN using h(n) with std=2');
```

Figure 3.3.1 Code to run

Result:

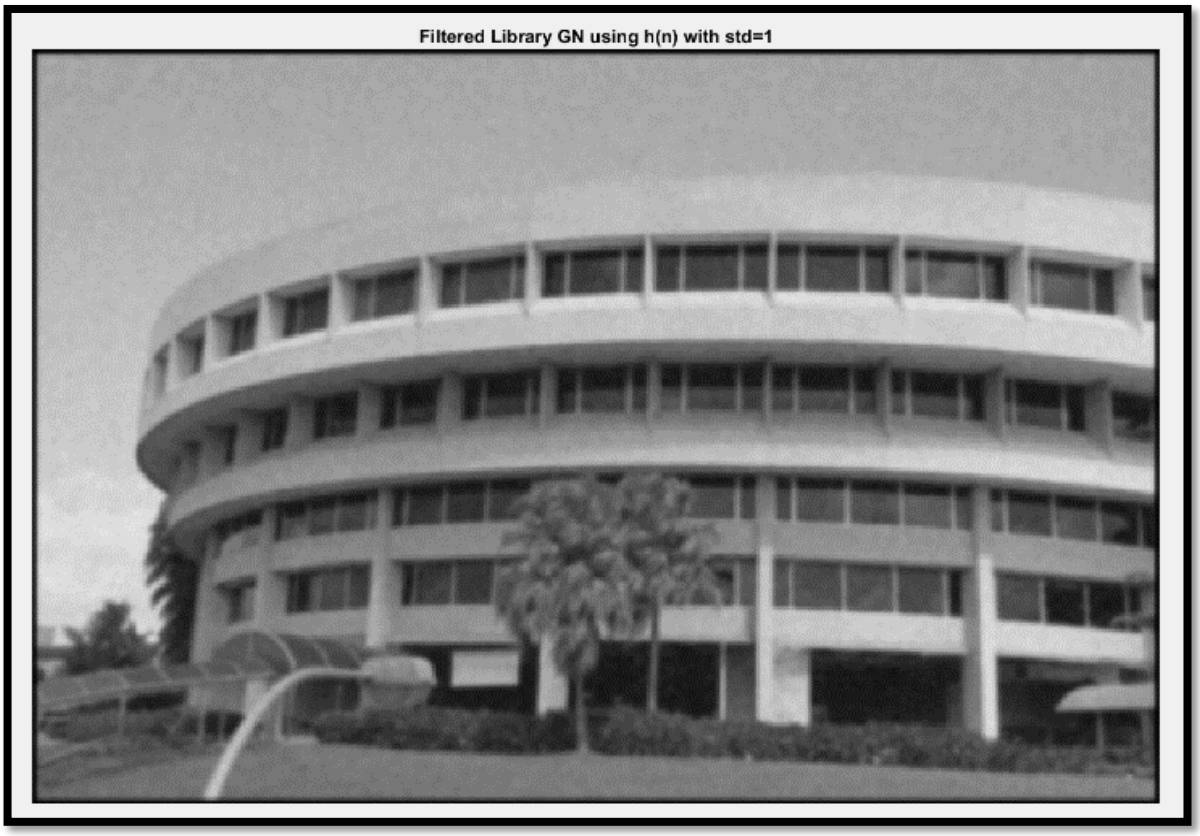

Figure 3.3.2 Filtered Library GN using h(n) with std=1

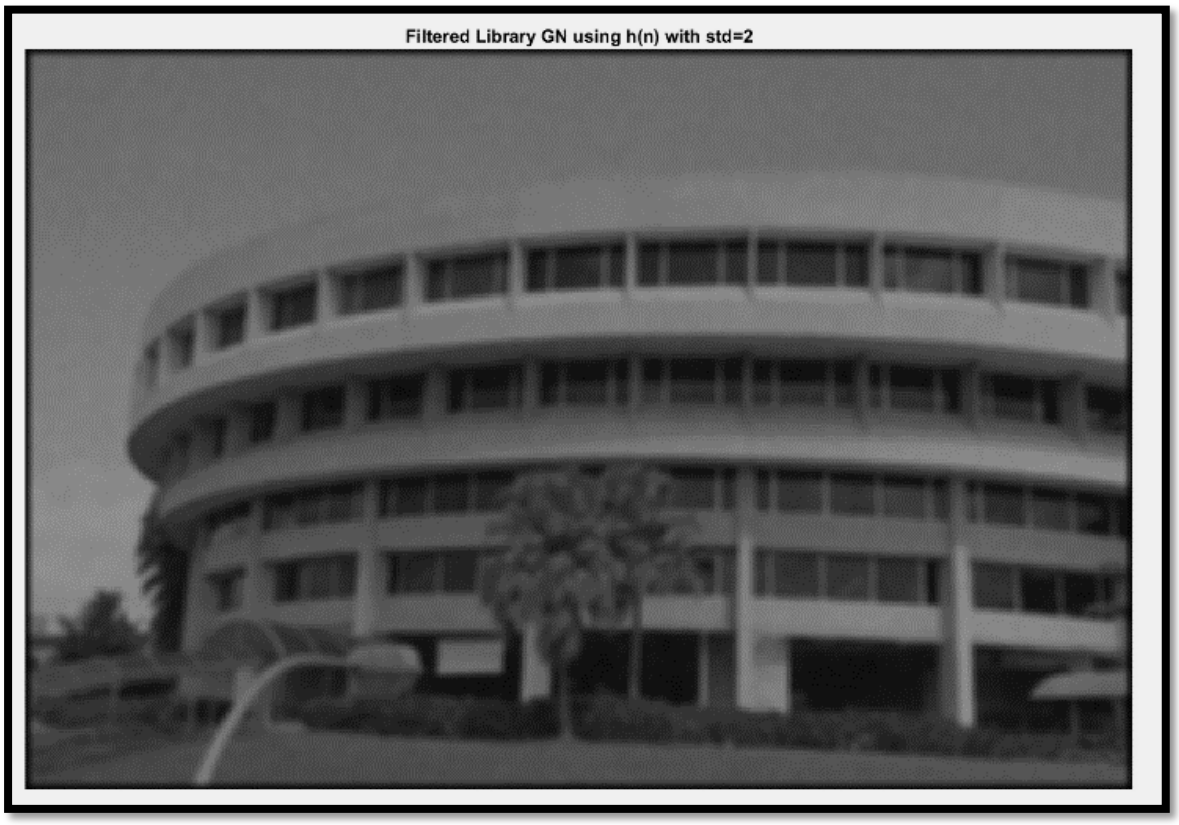

Figure 3.3.3 Filtered Library GN using h(n) with std=2

Answer:

The filters are effective in removing Gaussian noise and the higher the standard deviation the more noise is being removed.

The trade-off of using the filter is the picture will become more blurred because removing noise will blurred the edges also but if picture is not being filtered then we will have a picture that has a lot of noise which make the picture not clear also.

d. Download the image ‘ntu-sp.jpg’ from NTULearn and view it. Notice that this image has additive speckle noise.

Figure 3.4 NTU Library Picture with Speckle Noise

e. Repeat step (c) above. Are the filters better at handling Gaussian noise or speckle noise?

```
lib_sp = imread('lib-sp.jpg');
%imshow(lib_sp);

%-------------------------------------------------------------------------------------
% e. Repeat step (c) above. Are the filters better at handling Gaussian noise or speckle noise?
%-------------------------------------------------------------------------------------
filtered_lib_sp1 = uint8(conv2(lib_sp,h1));
filtered_lib_sp2 = uint8(conv2(lib_sp,h2));

imshow(filtered_lib_sp1);
title('Filtered Library SP using h(n) with std=1');
imshow(filtered_lib_sp2);
title('Filtered Library SP using h(n) with std=2');

% Answer: The filters are able to remove the speckle noise also but it is
%         not as effective as removing gaussian noise as there is still
%         some visible but not very clear small white dot in the pictures
```

Figure 3.5.1 Code to run

Result:

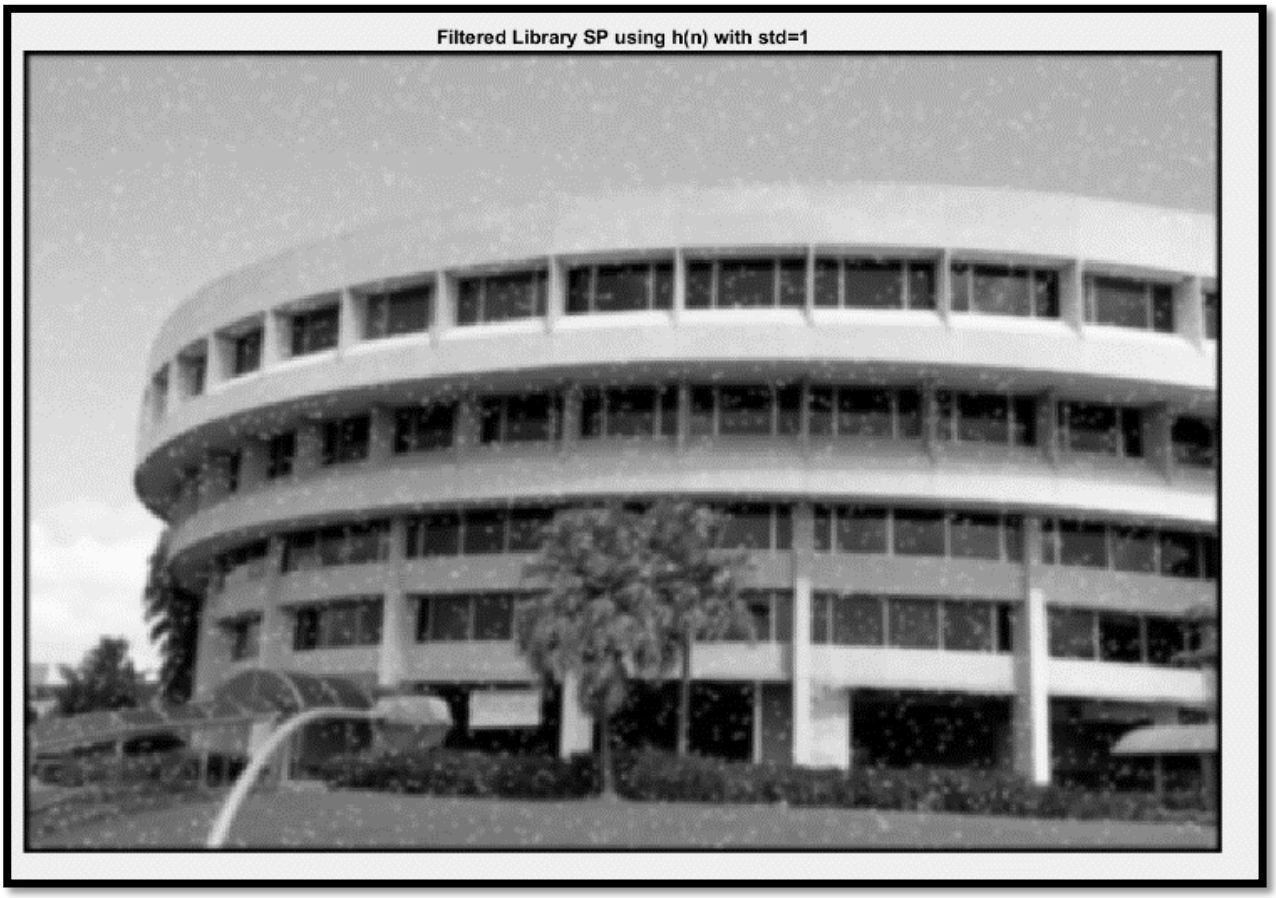

Figure 3.5.1 Filtered Library SP using h(n) with std=1

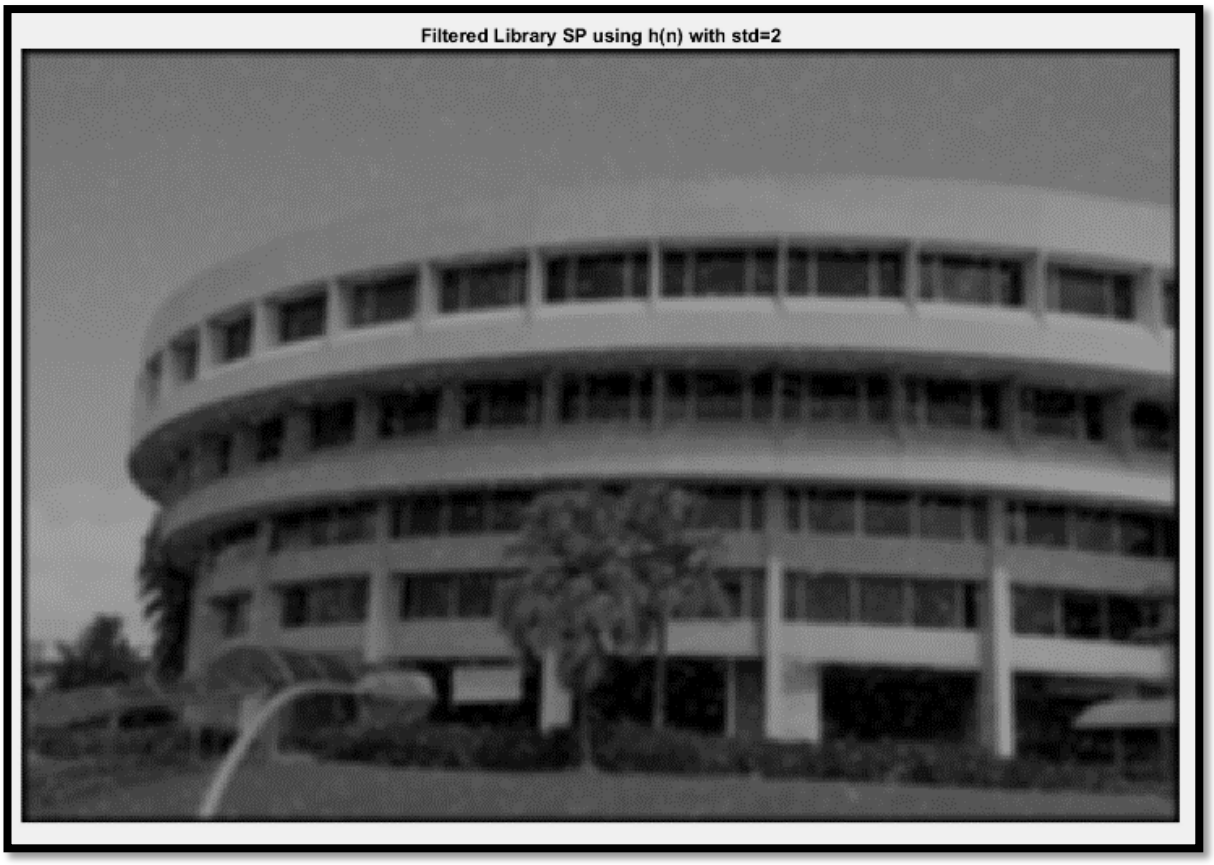

Figure 3.5.2 Filtered Library SP using h(n) with std=2

Answer:

The filters are able to remove the speckle noise also but it is not as effective as removing gaussian noise as there is still some visible but not very clear small white dot in the pictures.

4. Median Filtering

Median filtering is a special case of order-statistic filtering. For each pixel, the set of intensities of neighbouring pixels (in a neighbourhood of specified size) are ordered. Median filtering involves replacing the target pixel with the median pixel intensity. Repeat steps (b)-(e) in Section 2.3, except instead of using h(x,y) and conv2, use the command medfilt2 with different neighbourhood sizes of 3x3 and 5x5. How does Gaussian filtering compare with median filtering in handling the different types of noise? What are the tradeoffs?

Code:

```
mfiltered_libgn1 = medfilt2(lib_gn,[3,3]);
mfiltered_libgn2 = medfilt2(lib_gn,[5,5]);
imshow(mfiltered_libgn1);
title('Filtered Library GN using Median Filter(3X3)');
imshow(mfiltered_libgn2);
title('Filtered Library GN using Median Filter(5X5)');

mfiltered_libsp1 = medfilt2(lib_sp,[3,3]);
mfiltered_libsp2 = medfilt2(lib_sp,[5,5]);
imshow(mfiltered_libsp1);
title('Filtered Library SP using Median Filter(3X3)');
imshow(mfiltered_libsp2);
title('Filtered Library SP using Median Filter(5X5)');
```

Figure 4.1.1 Code to run

Result:

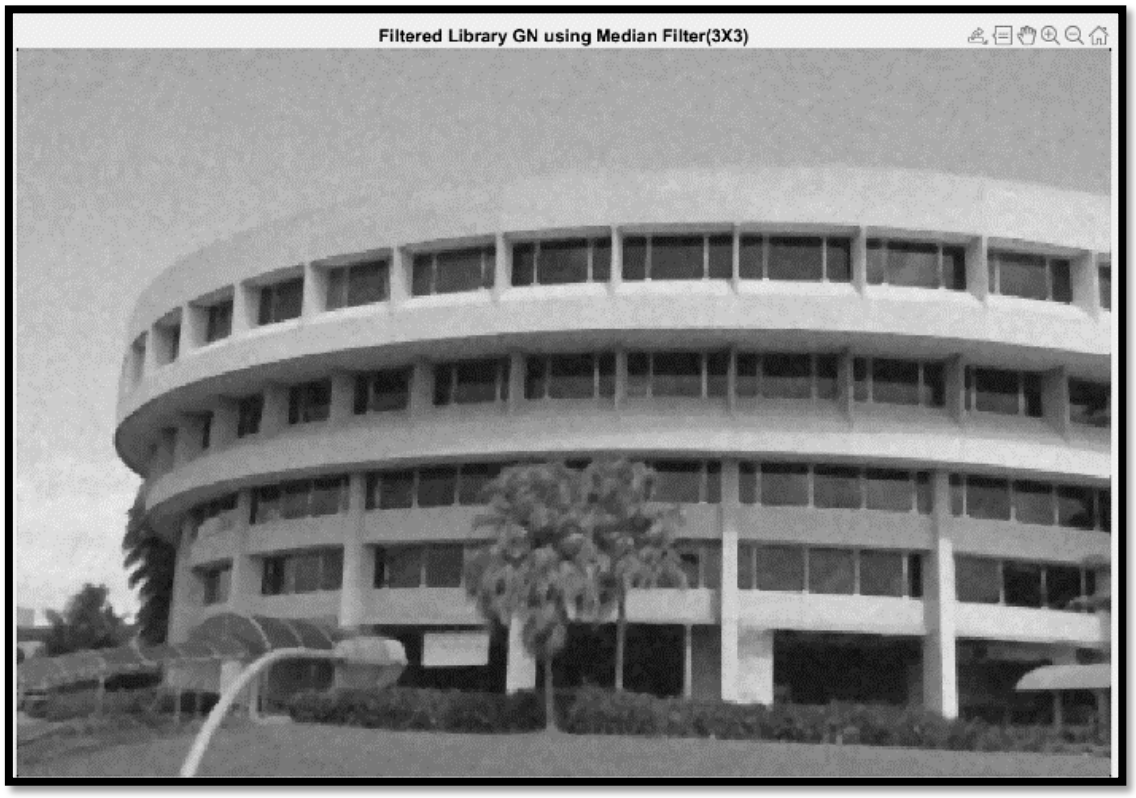

Figure 4.1.2 Library GN after 3x3 Median Filtering

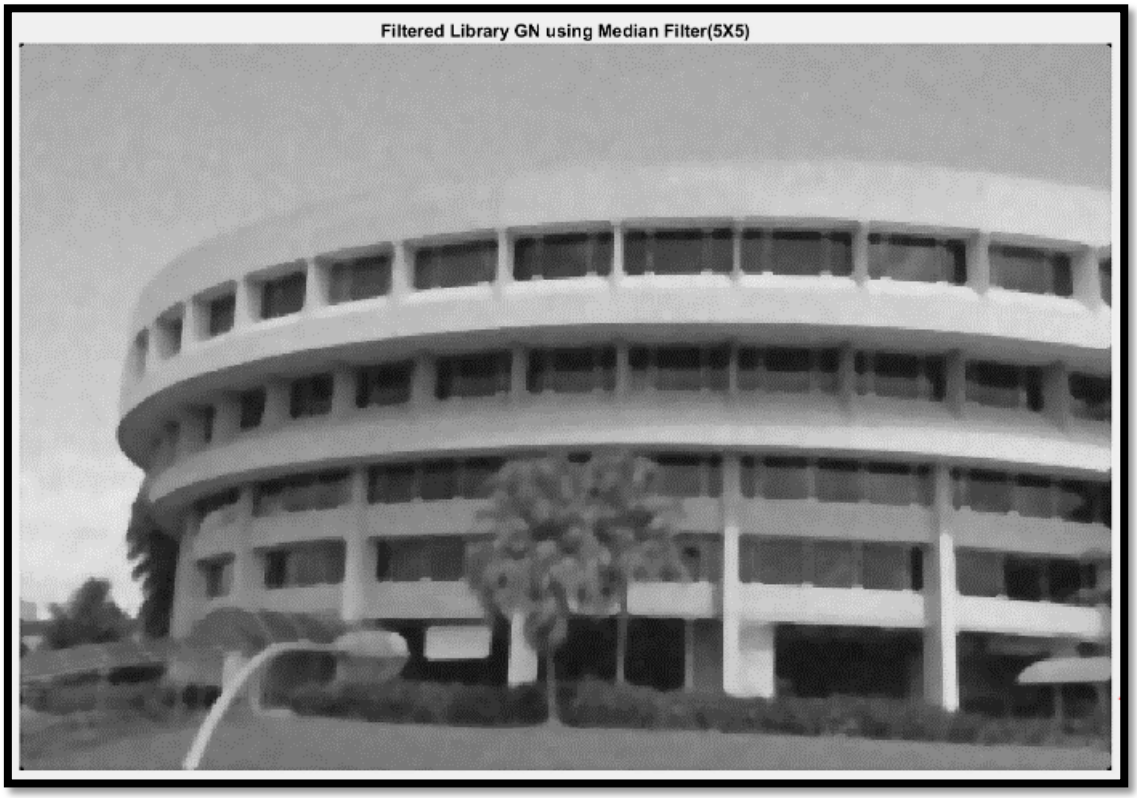

Figure 4.1.3 Library GN after 5x5 Median Filtering

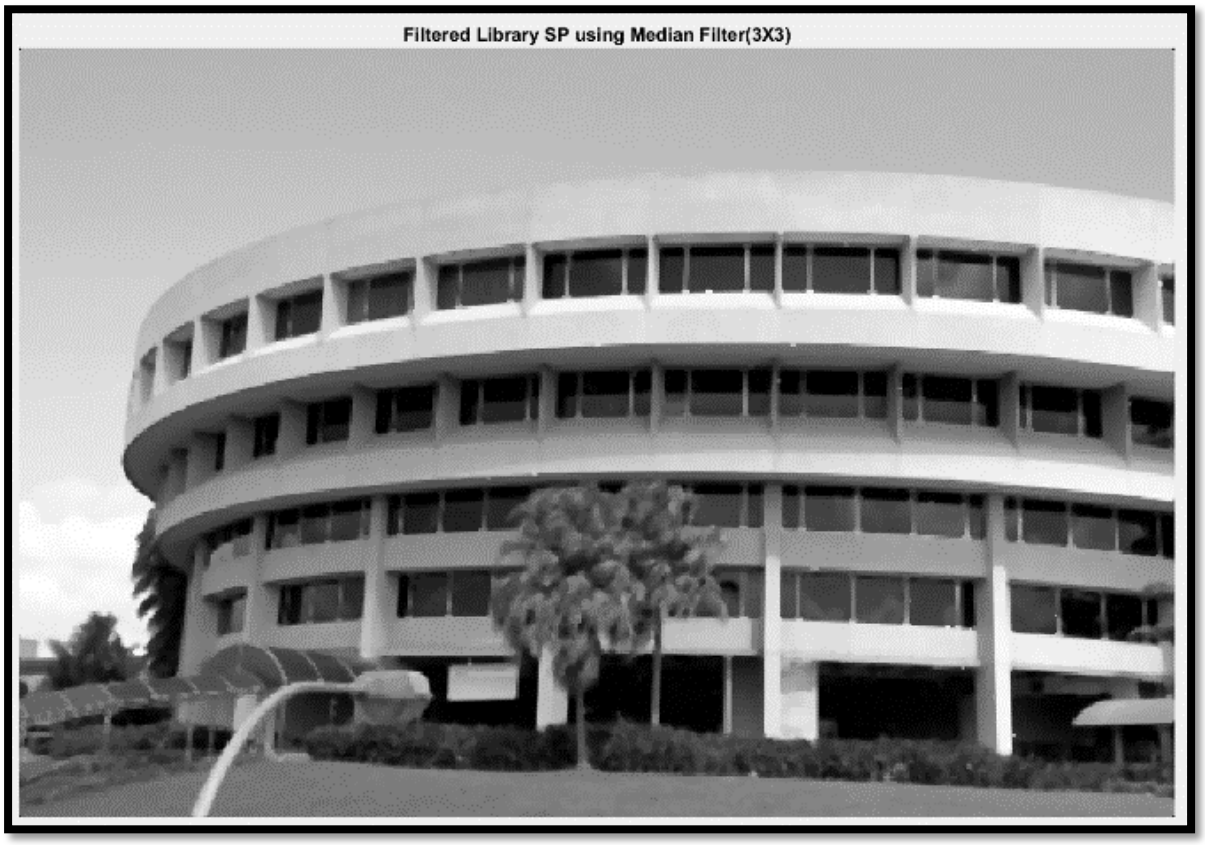

Figure 4.1.4 Library SP after 3x3 Median Filtering

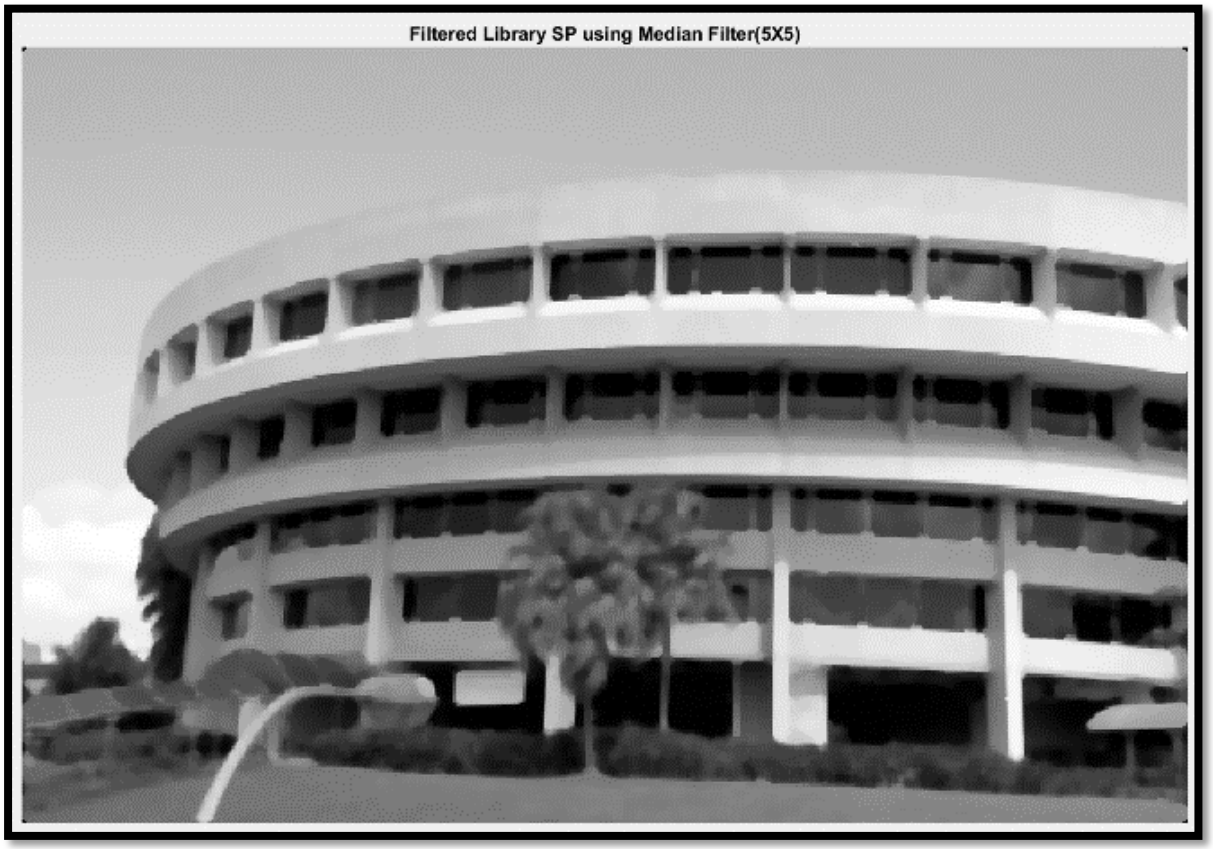

Figure 4.1.5 Library SP after 5x5 Median Filtering

Answer :

Gaussian Filtering is better at removing gaussian noise compared to Median Filtering whereas Median Filtering is better at removing speckle noise compared to Gaussian Filtering and the more the neighbourhood size the better the removing of noise for both noises.

Using Median Filtering, speckle noise which is the white dot in the picture can be removed efficiently and the edges can be preserved way better than using Gaussian Filtering but in trade-off which the gaussian noise is not able to be removed easily. Increasing of neighbourhood size will increase the efficiency of removing the noises in trade-off of edges will not be preserved that well.

5. Suppressing Noise Interference Patterns

Occasionally with poor television reception, parallel lines will be seen on the screen corrupting the desired image. These are interference patterns. Here we explore how bandpass filtering can be used to suppress such interference.

In this lab assignment, we are using Fourier Transform for suppressing interference.

The Fourier transform can be used to interpolate functions and to smooth signals. For example, in the processing of pixelated images, the high spatial frequency edges of pixels can easily be removed with the aid of a two-dimensional Fourier transform.

Steps:

a. Download the image 'pck-int.jpg' from NTULearn and display it from MATLAB. Notice the dominant diagonal lines destroying the quality of the image.

Figure 5.1 Pck-int

b. Obtain the Fourier transform F of the image using fft2, and subsequently compute the power spectrum S. Note that F should be a matrix of complex values, but S should be a real matrix. Display the power spectrum by

```
>> imagesc(fftshift(S.^0.1));
>> colormap('default');
```

fftshift is used to shift the origin of the Fourier transform to the centre of the image. Note that the origin corresponds to the DC (or zero frequency) component. The power to 0.1 is only used to nonlinearly scale the power spectrum such that it is easier to visualize the frequency components.

Notice that there are two distinct, symmetric frequency peaks that are isolated from the central mass. These frequency components correspond to the interference pattern.

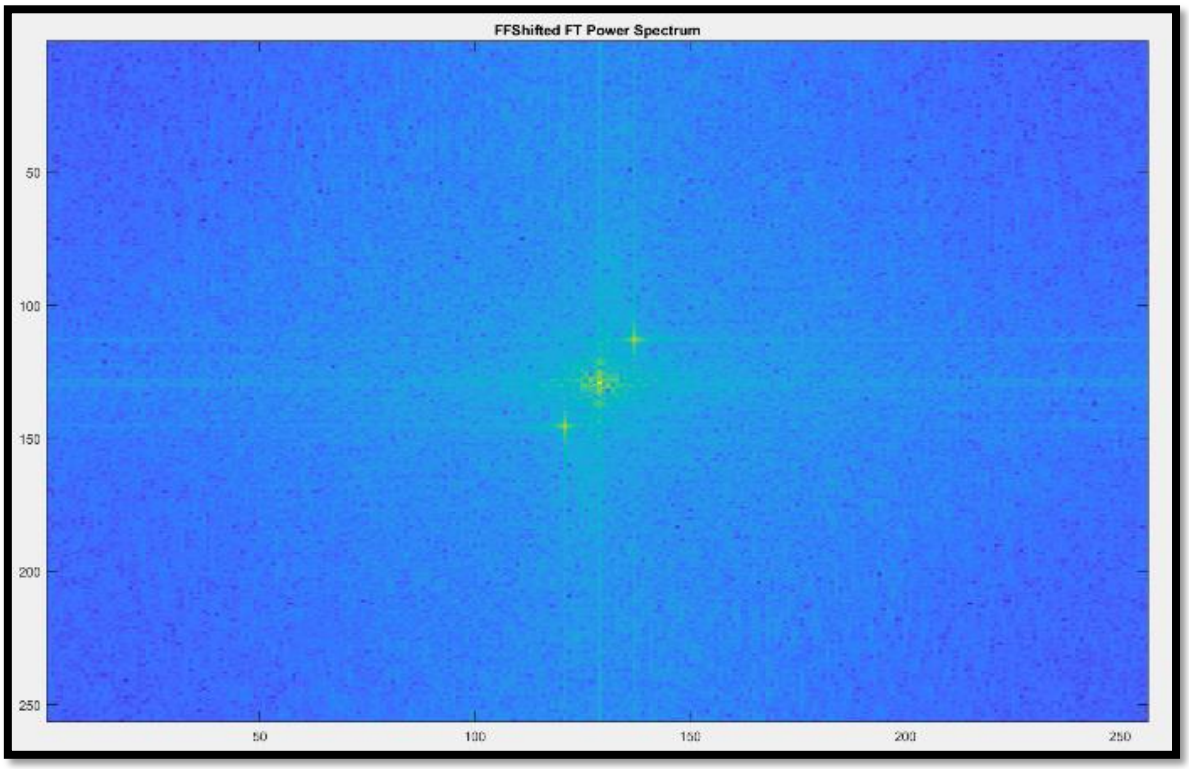

Figure 5.2.1 FFTShifted of Power Spectrum

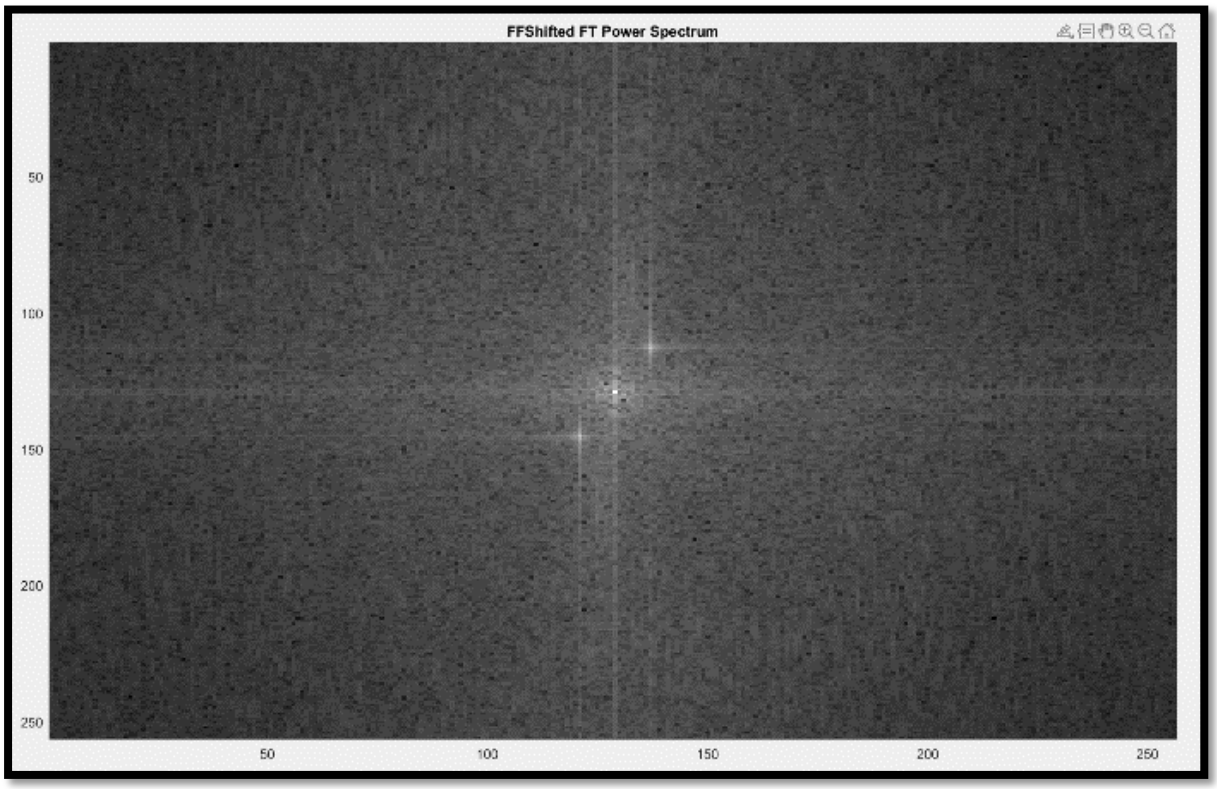

Figure 5.2.2 FFTShifted of Power Spectrum(Grey)

c. Redisplay the power spectrum without fftshift. Measure the actual locations of the peaks. You can read the coordinates off the x and y axes, or you can choose to use the ginput function.

```
figure;
imagesc(S.^0.1);
colormap('default');
title("FT Power Spectrum")
figure;
imagesc(S.^0.1);
colormap('gray');
title("fft (Grey)")

% Locate coordinate (x,y) using ginput() function
% https://www.mathworks.com/help/matlab/ref/ginput.html
[n,m] = ginput(1);
```

Figure 5.3.1 Code to run

Power Spectrum:

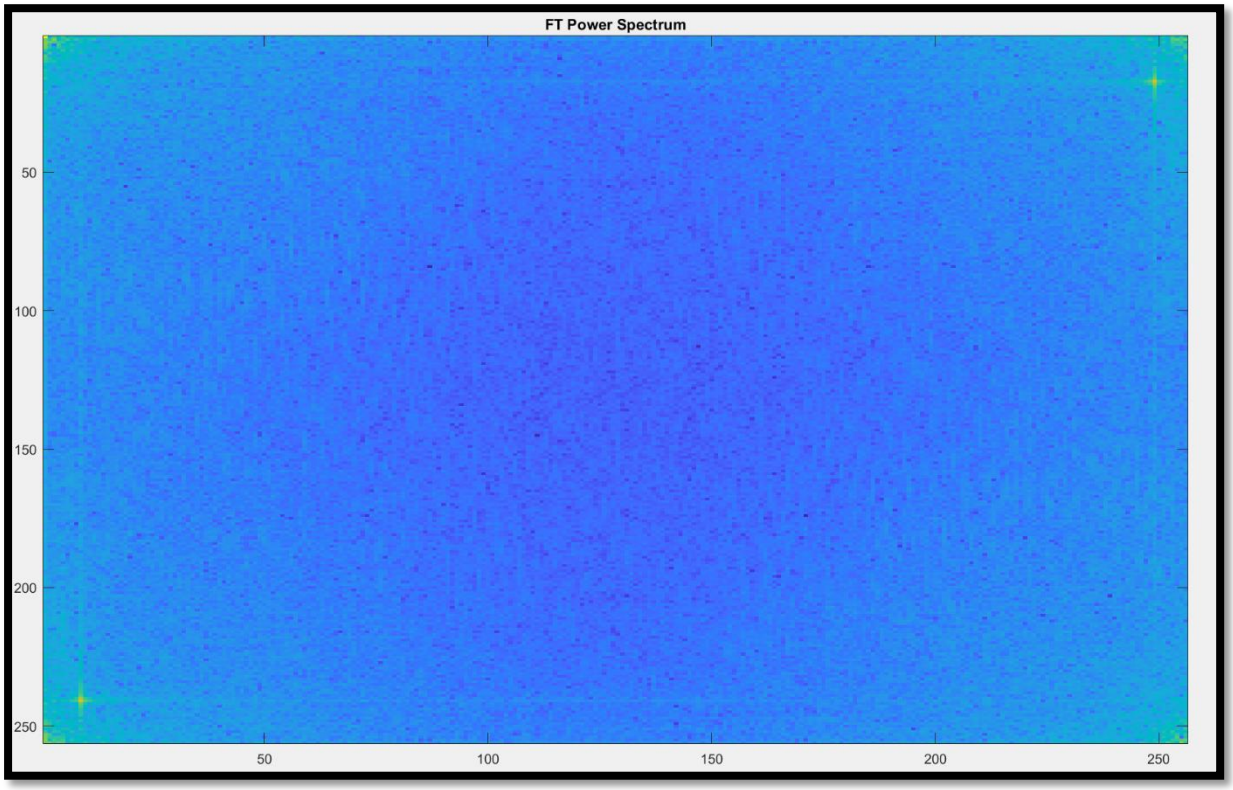

Figure 5.3.2 Power Spectrum

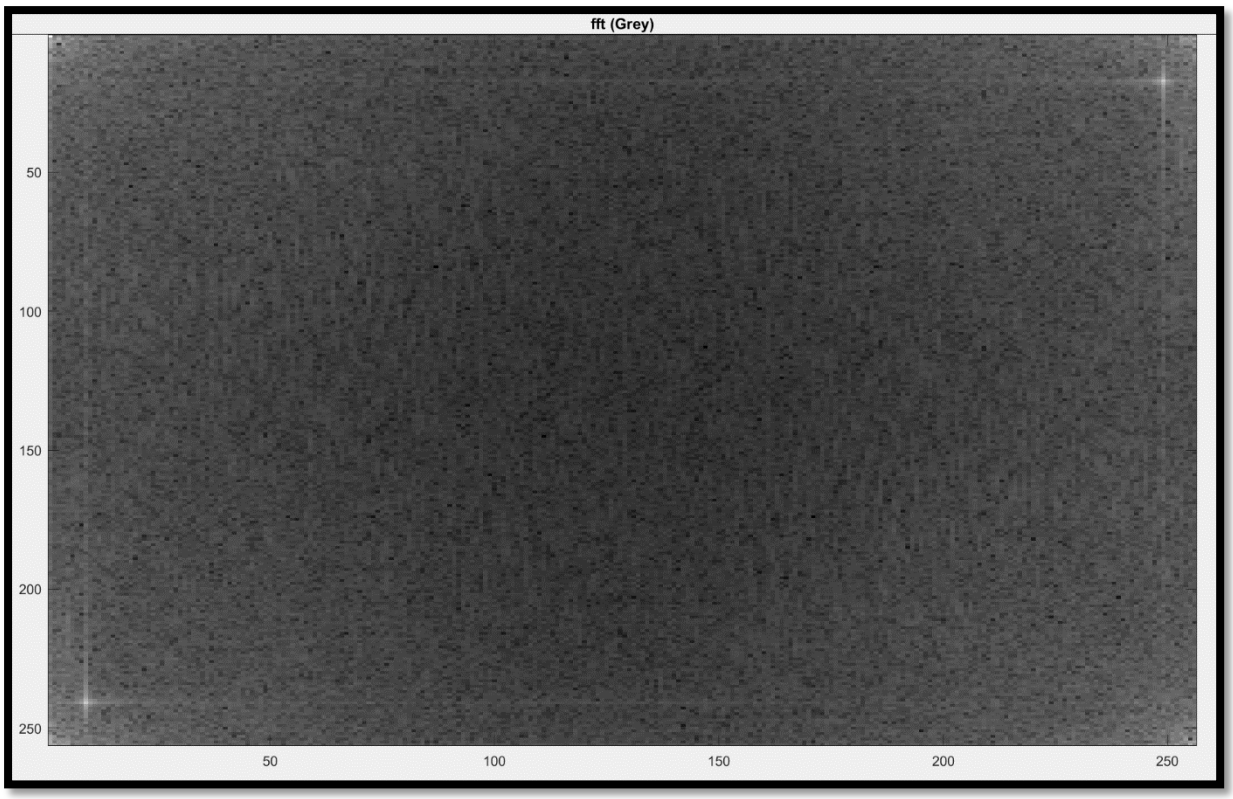

Figure 5.3.3 Power Spectrum(Grey)

Coordinates of power spectrum:

Top Right FFT Power Spectrum
n=[15, 19]
m=[247,251]

Bottom left FFT Power Spectrum
n=[239,243]
m=[7,11]

d. Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks in the Fourier transform F, not the power spectrum.

Recompute the power spectrum and display it as in step (b).

```
F(15:19,247:251)=0;
F(239:243,7:11)=0;
S=abs(F).^2;
figure;
imagesc(fftshift(S.^0.1));
colormap('default');
title("FFShifted FT Power Spectrum(Recomputed)")
figure;
imagesc(fftshift(S.^0.1));
colormap('gray');
title("FFShifted FT Power Spectrum(Recomputed)")
```

Figure 5.4.1 Code to run

Result:

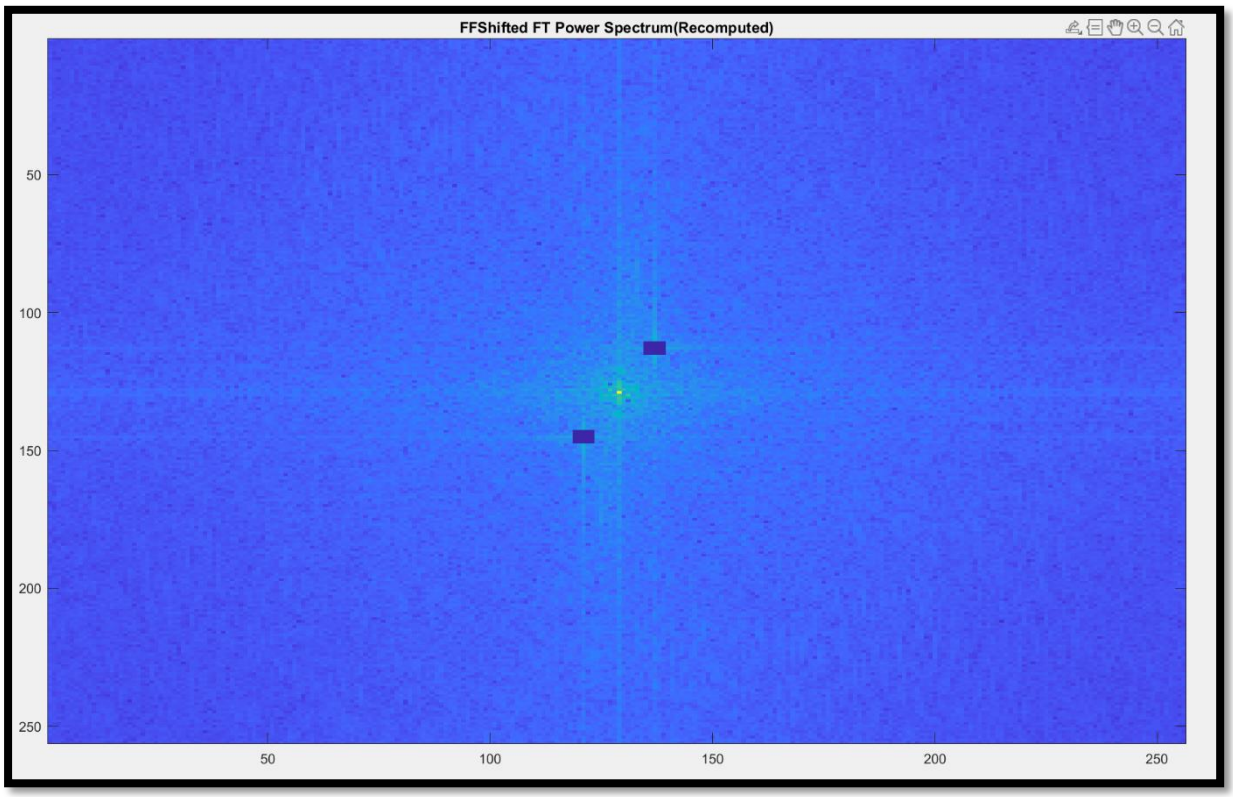

Figure 5.4.2 Power Spectrum After set 0

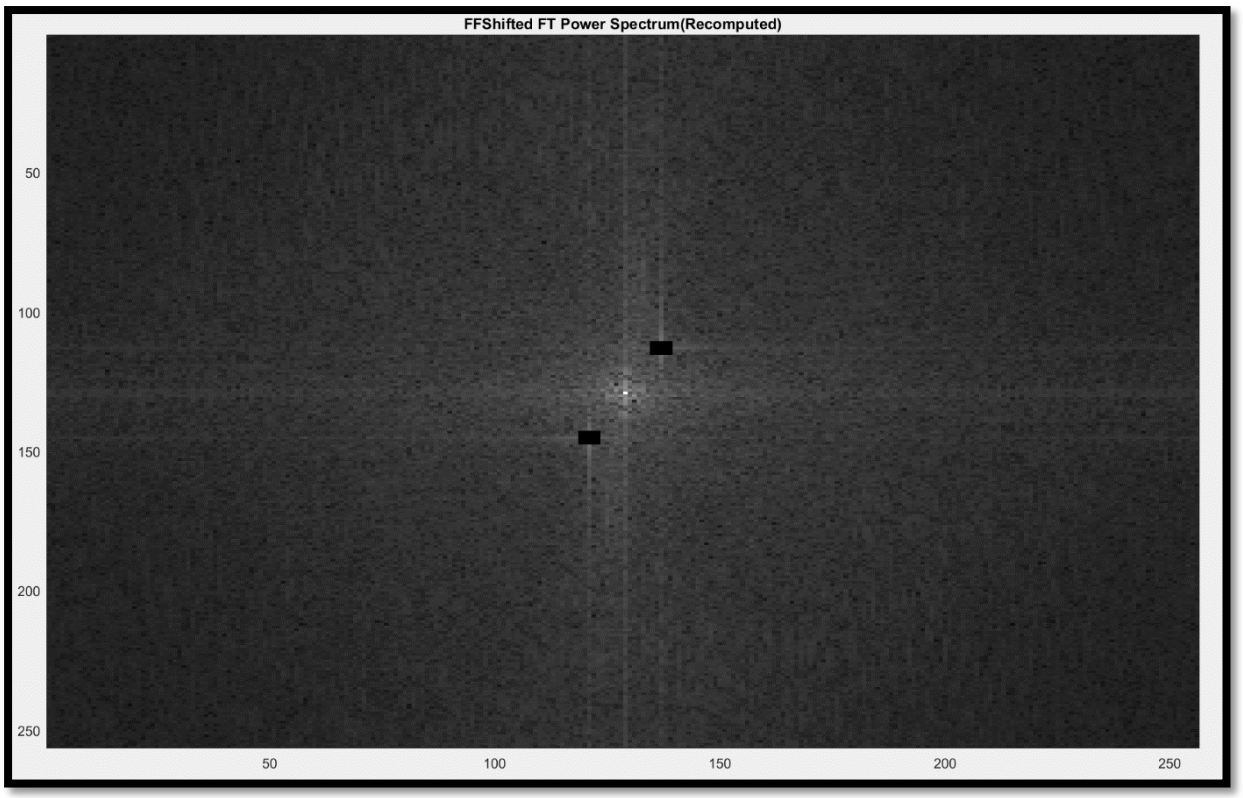

Figure 5.4.3 Power Spectrum After set 0(Grey)

e. Compute the inverse Fourier transform using ifft2 and display the resultant image. Comment on the result and how this relates to step (c). Can you suggest any way to improve this?

```
figure;
colormap('default');
imshow(uint8(ifft2(F)))
```

Figure 5.5.1 Code to run

Result:

Figure 5.5.2 pck-int removed interference

Answer :

Most of the interference pattern are removed when we set F(n,m)=0 which n and m are the coordinates of the peak. When F(n,m) is set to 0, the specific sets of building block('atom') of the image that is corresponds to the interference patterns is being removed and when we use the inverse Fourier Transform the specific atom with 'weight'=0 will have no influence on the image

Improvement Suggestion:

In this question, I only used two peaks that is most visible for fourier transform and actually if we look closer we might able to see some of the possible peaks that is not very visible, we can try out by adding those different peaks, or use different combination of the peaks to do fourier transform and see which result we get is the best result.

f. Download the image `primate-caged.jpg' from NTULearn which shows a primate behind a fence. Can you attempt to "free" the primate by filtering out the fence? You are not likely to achieve a clean result but see how well you can do.

Original Image:

Figure 5.6.1 Primate-Caged

Power Spectrum:

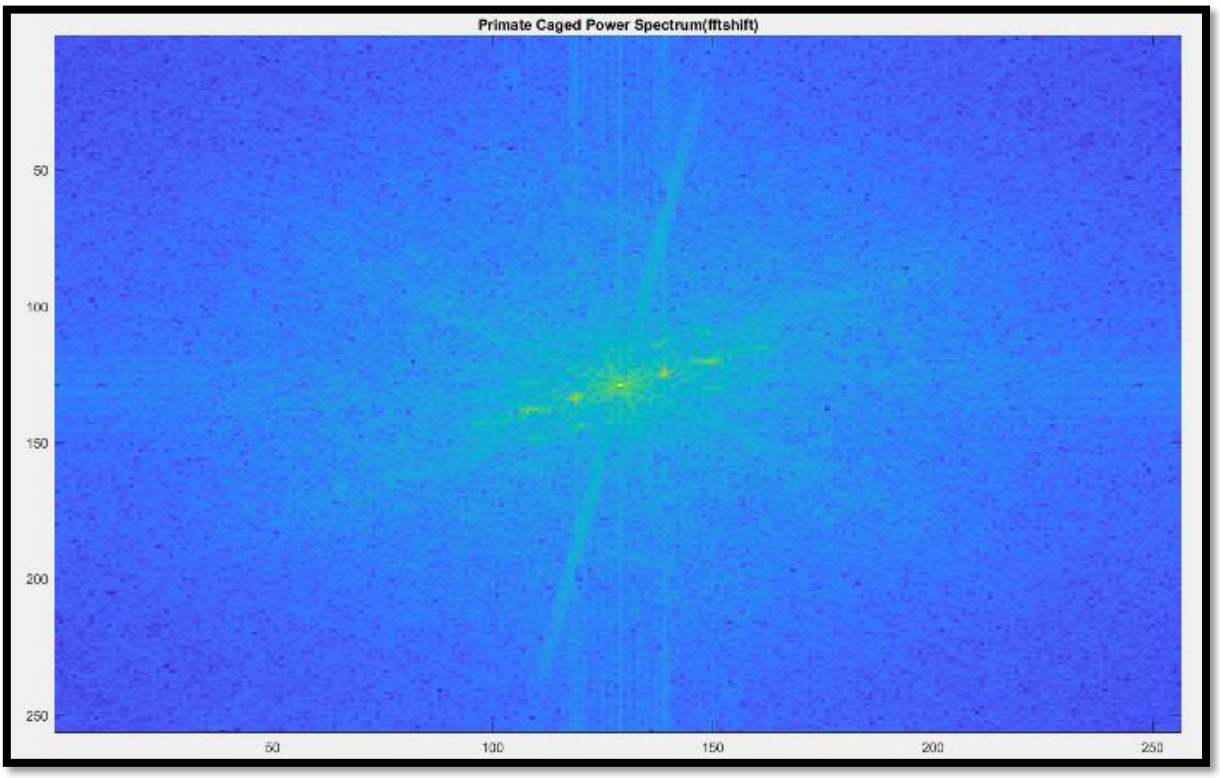

Figure 5.6.2 Power Spectrum

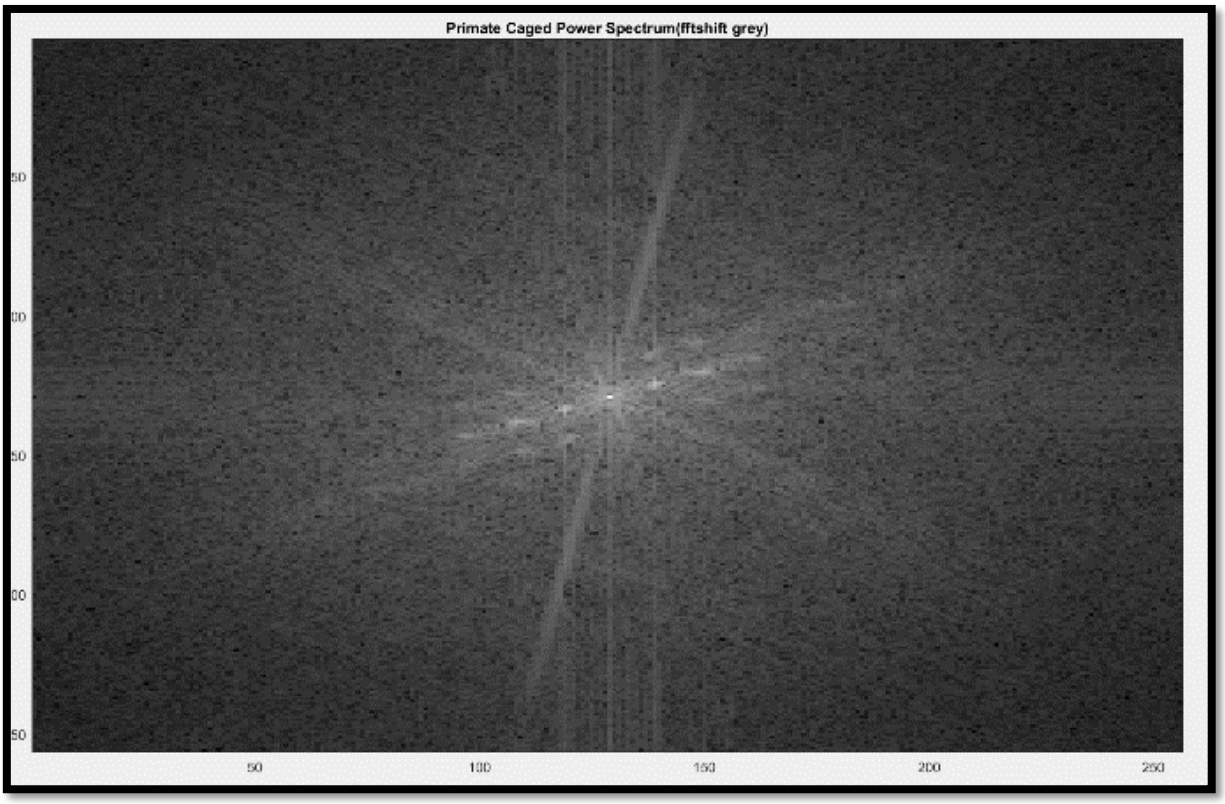

Figure 5.6.3 Power Spectrum(Grey)

Coordinates of the peaks:

Top Right:

x1 = 6;

y1 = 247;

x2 = 10;

y2 = 237;

Bottom Left :

x3 = 252;

y3 = 11;

x4 = 248;

y4 = 22;

Result after the peaks' 5x5 neighbourhood elements have been set to 0:

Figure 5.6.4 Primate-caged result after removed interference

Conclusion:

From the result, we can observe that the line of the cage has been significantly lesser than the original image after removing the peaks identified.

However, the lines is still more visible compared to the result we get from pck-int and this lead to a conclusion that a more complex and powerful technique is required to fully removed the line if the line is not straight. We can also be observed that the peaks are less obvious from the spectrum map when compared to the pck-int power spectrum.

This conclude that the fourier transform is unable to detect the line patterns as clearly when the line is not in straight line and we are required to use another method to tackle this issue if we want to fully remove the line of the cage.

6. Undoing Perspective Distortion of Planar Surface

The following paragraph provides the basic knowledge on 2D planar projective transformation. A 2D planar projective transform is expressed via a 3x3 matrix:

$$\begin{bmatrix} kx_{im} \\ ky_{im} \\ k \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix}$$

where Xw and Yw are the coordinates of a point in the input image, while xim and yim are the coordinates of the transformed point in the output image. This can be alternatively expressed in algebraic form:

$$x_{im} = \frac{m_{11}X_w + m_{12}Y_w + m_{13}}{m_{31}X_w + m_{32}Y_w + 1}, \mathrm{y}_{im} = \frac{m_{21}X_w + m_{22}Y_w + m_{23}}{m_{31}X_w + m_{32}Y_w + 1}.$$

Since there are only 8 degrees of freedom in the projective matrix (i.e., 8 unknowns), we can compute this matrix if we know the correspondence between 4 or more points in the input image and the output image. Using the correspondences and the above equations, the following matrix equation may be set up

$$\begin{bmatrix} X_w^1 & Y_w^1 & 1 & 0 & 0 & 0 & -x_{im}^1 X_w^1 & -x_{im}^1 Y_w^1 \\ 0 & 0 & 0 & X_w^1 & Y_w^1 & 1 & -y_{im}^1 X_w^1 & -y_{im}^1 Y_w^1 \\ X_w^2 & Y_w^2 & 1 & 0 & 0 & 0 & -x_{im}^2 X_w^2 & -x_{im}^2 Y_w^2 \\ 0 & 0 & 0 & X_w^2 & Y_w^2 & 1 & -y_{im}^2 X_w^2 & -y_{im}^2 Y_w^2 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . \\ X_w^n & Y_w^n & 1 & 0 & 0 & 0 & -x_{im}^n X_w^n & -x_{im}^n Y_w^n \\ 0 & 0 & 0 & X_w^n & Y_w^n & 1 & -y_{im}^n X_w^n & -y_{im}^n Y_w^n \end{bmatrix} \begin{bmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{31} \\ m_{32} \end{bmatrix} = \begin{bmatrix} x_{im}^1 \\ y_{im}^1 \\ x_{im}^2 \\ y_{im}^2 \\ . \\ . \\ x_{im}^n \\ y_{im}^n \end{bmatrix} . \quad (*)$$

AU=V

We can write the above equation in the form of Au = v, where A and v are the 8x8 matrix and 8x1 vector respectively, containing corner data, while u is the 8x1 vector of projective transformation parameters to be computed. This equation allows the computation of an unknown projective transformation through simple matrix inversion (for 4 correspondences) or pseudo-inversion (for >4 correspondences).

Steps:

a. Download `book.jpg' from the NTULearn website as a matrix P and display the image. The image is a slanted view of a book of A4 dimensions, which is 210 mm x 297 mm.

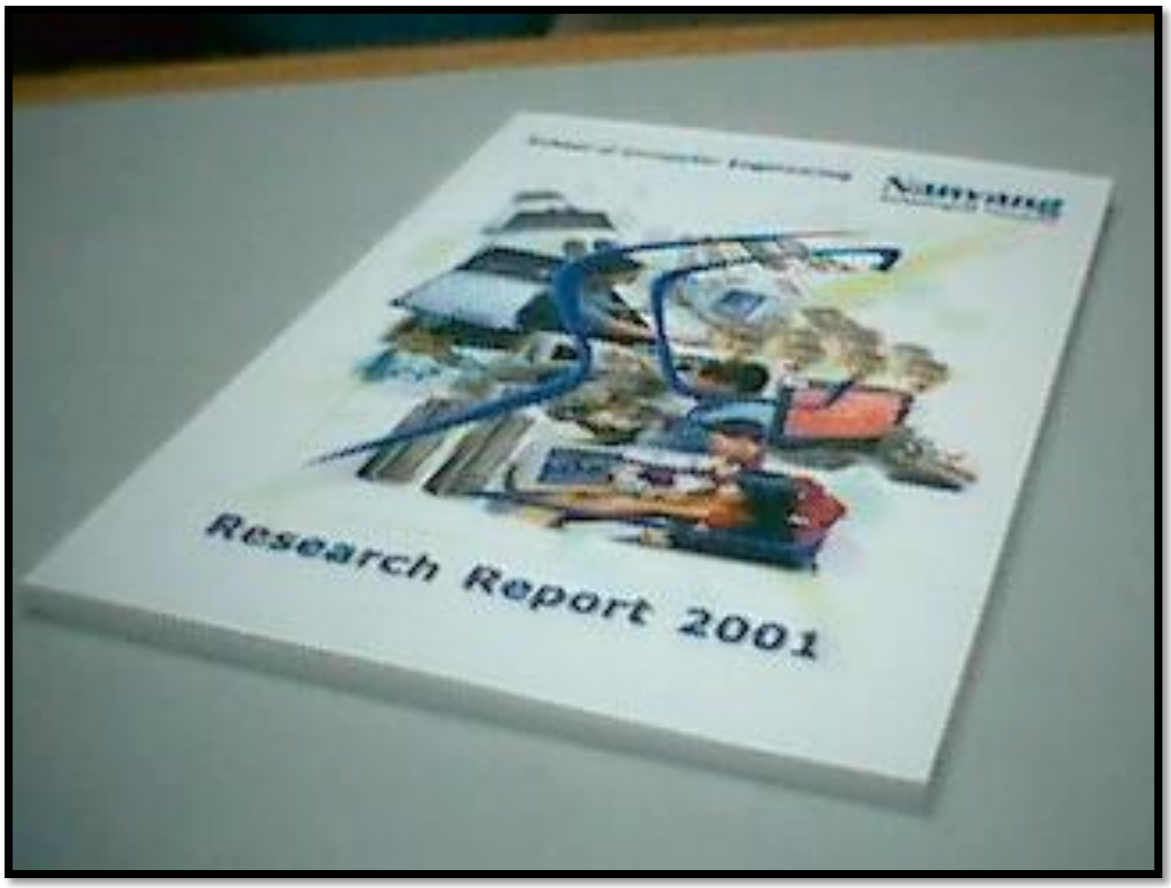

Figure 6.1 Book.jpg

b. The ginput function allows you to interactively click on points in the figure to obtain the image coordinates. Use this to find out the location of 4 corners of the book, remembering the order in which you measure the corners.

```
>> [X Y] = ginput(4)
```

Also, specify the vectors x and y to indicate the four corners of your desired image (based on the A4 dimensions), in the same order as above.

```
X_im = [0; 210; 210; 0];
Y_im = [0; 0; 297; 297];
```

Figure 6.2 Desired output of matrix V

c. Set up the matrices required to estimate the projective transformation based on the equation (*) above.

```
A = [
    [X(1), Y(1), 1, 0, 0, 0, -X_im(1) * X(1), -X_im(1) * Y(1)];
    [0, 0, 0, X(1), Y(1), 1, -Y_im(1) * X(1), -Y_im(1) * Y(1)];
    [X(2), Y(2), 1, 0, 0, 0, -X_im(2) * X(2), -X_im(2) * Y(2)];
    [0, 0, 0, X(2), Y(2), 1, -Y_im(2) * X(2), -Y_im(2) * Y(2)];
    [X(3), Y(3), 1, 0, 0, 0, -X_im(3) * X(3), -X_im(3) * Y(3)];
    [0, 0, 0, X(3), Y(3), 1, -Y_im(3) * X(3), -Y_im(3) * Y(3)];
    [X(4), Y(4), 1, 0, 0, 0, -X_im(4) * X(4), -X_im(4) * Y(4)];
    [0, 0, 0, X(4), Y(4), 1, -Y_im(4) * X(4), -Y_im(4) * Y(4)];
];
```

Figure 6.3.1 Matrix A

```
v = [X_im(1); Y_im(1); X_im(2); Y_im(2); X_im(3); Y_im(3); X_im(4); Y_im(4)];
```

Figure 6.3.2 Matrix V

```
u = A\v;
```

Figure 6.3.3 u=(A^-1)v

Calculate for the unknown u matrix(projective transformation parameters) for transformation:

u=(A^-1)*v

Reshape projective transformation parameters, u back to normal matrix form:

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

Figure 6.3.4 Reshape

Write down this matrix. Verify that this is correct by transforming the original coordinates:

```
w = U*[X'; Y'; ones(1,4)];
```

```
w = w ./ (ones(3,1) * w(3,:));
```

Figure 6.3.5 Code for checking correctness

Transform the original picture to frontial view:

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

Figure 6.3.6 Code to convert to frontial view

Result:

Figure 6.3.7 Book after transform

Answer:

The quality of the book picture is not as good as directly take a new picture of frontial view of the book because the picture is cropped and warp into new dimension of picture which the top part of the picture is more blurry than the bottom part of the picture, the exactly same as the original picture used.

f. In your new image, identify the big rectangular pink area, which seems to be a computer screen, at about middle place between “Nanyang” and “2001”. You may use any methods you wish.

For this question, to identify the big rectangular pink(orange) area, I will approach it using a mask to identify which area is in orange. The threshold of the RGB is shown as below:

190<=R<=225
110<=G<=150
110<=B<=120

The threshold of the orange is based on Figure 6.3.7, I uploaded it to a online image RGB checking tools and try to check for the RGB around the computer screen area and the RGB value I have gotten is around this value.

The image RGB checking tools link:
(https://imagecolorpicker.com/en)

Code:

```
img = P2;
red_layer   = img(:,:,1);
green_layer = img(:,:,2);
blue_layer  = img(:,:,3);
% Define RGB code range for orange color

% Apply thresholds
orange_pos = red_layer>=180 &red_layer<=220 & ...
             green_layer>=100&green_layer<=150 & ...
             blue_layer>=90&blue_layer<=150;

BW = orange_pos;
maskedRGBImage = img;
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

imshow(maskedRGBImage);

[r, c] = find(BW);
row1 = min(r);
row2 = max(r);
col1 = min(c);
col2 = max(c);
disp(row1);
disp(row2);
disp(col1);
disp(col2);

croppedImage = imcrop(img,[col1 row1 col2-col1 row2-row1]);
imshow(croppedImage);
```

Figure 6.4.1 Code to run

Mask of orange:

Figure 6.4.2 Orange Mask of the Frontial View of Book

The mask seems to cover some of the orange part at the computer screen, which seems that this mask is usable for indicating which position of image should be cropped, the only things that is not very good is there is a orange dot under the

cluster of orange dot which will affect the accuracy of finding the position to crop the image but this is the best mask that I can come out with, but the mask may be improved if the threshold is set at a more accurate value.

Result:

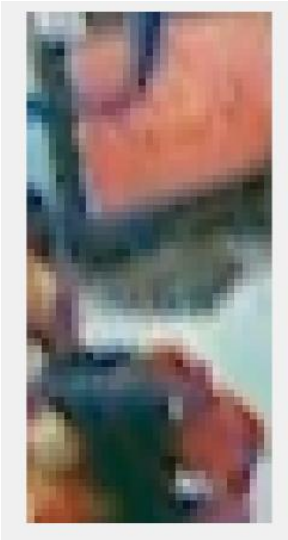

Figure 6.4.3 Orange Computer Screen

And this is the computer screen to be identified.

Retry for getting a better mask:

Code:

```
img = P2;
red_layer   = img(:,:,1);
green_layer = img(:,:,2);
blue_layer  = img(:,:,3);
% Define RGB code range for orange color

% Apply thresholds
orange_pos = red_layer>=196 &red_layer<=220 & ...
             green_layer>=114&green_layer<=150 & ...
             blue_layer>=90&blue_layer<=150;

BW = orange_pos;
maskedRGBImage = img;
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

imshow(maskedRGBImage);

[r, c] = find(BW);
row1 = min(r);
row2 = max(r);
col1 = min(c);
col2 = max(c);
disp(row1);
disp(row2);
disp(col1);
disp(col2);
```

Figure 6.4.4 Code to run(retry)

I tried to readjust the threshold to find orange color position to make sure that the value is not from the orange or red color from the sofa in the picture.

The threshold of the RGB is shown as below:

196<=R<=220
114<=G<=150
90<=B<=120

Result of the mask:

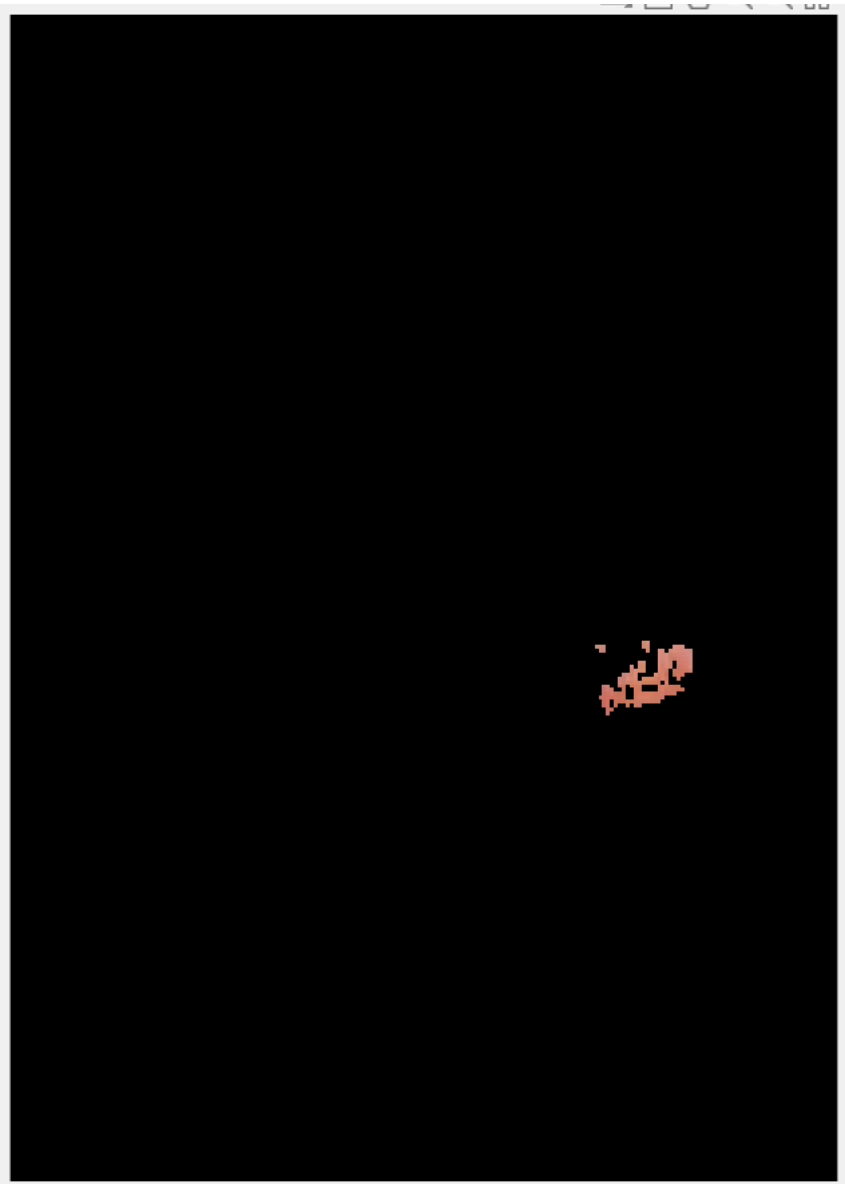

Figure 6.4.5 Orange Mask(Retry)

Result of the cropped image:

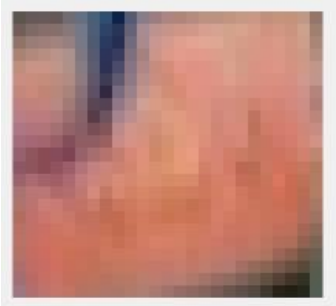

Figure 6.4.6 Orange Computer Screen

And we are able to crop out the correct position of the orange computer screen. *Do note that even thought the mask is incomplete I am able to crop out the orange screen since we only need the minimum and maximum position of x and y for cropping of image.*

Removing of blue part in orange screen:

```
% Trying to remove the blue line via changing to other color
%%
originalImg = croppedImage;

% red_layer   = img(:,:,1);
% green_layer = img(:,:,2);
% blue_layer  = img(:,:,3);

red_layer   = originalImg(:,:,1);
green_layer = originalImg(:,:,2);
blue_layer  = originalImg(:,:,3);
% Define RGB code range for orange color

% Apply thresholds for finding blue part
blue_pos = red_layer>=30 &red_layer<=150& ...
             green_layer>=30&green_layer<=150 & ...
             blue_layer>=60&blue_layer<=255;

BW = blue_pos;

imshow(BW);

red_layer(BW)=217;
green_layer(BW)=120;
blue_layer(BW)=107;

rgbImage = cat(3, red_layer, green_layer, blue_layer);
```

Figure 6.4.7 Code to run

Result:

Figure 6.4.8 After removing blue part

In result, we can see that inside the computer screen has a shape of:

_____END OF REPORT_____