

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4003 Computer Vision

Lab 1 Report

Zhang Yuhan

U1823060F



06 October 2020

Table of Contents

Contrast Stretching.....	3
Histogram Equalization	5
Linear Spatial Filtering	8
Median Filtering.....	11
Suppressing Noise Interference Patterns.....	14
Undoing Perspective Distortion of Planar Surface	20

Contrast Stretching

Importing the image:

Input	Figure 1.1 Original Image
<code>Pc=imread('mrttrainbland.jpg');</code> <code>whos Pc</code>	
Output	
Pc 320x443x3	
Input	Figure 1.2 Grayscale Image
<code>P=rgb2gray(Pc);</code> <code>whos P</code>	
Output	
P 320x443	

Although the image seems to be in grayscale (as shown in Figure 1.1), it is still considered an RGB image by matlab, so it is necessary to convert it to grayscale by using the `rgb2gray()` function before processing the image. This allows us to work with a 2-dimensional matrix and simplifies the process.

The original minimum and maximum intensities in the image must be obtained in order to do the contrast stretching.

Input	Outputs
<code>min(P(:)), max(P(:))</code>	Min: 13 Max: 204

Next, the minimum and maximum intensities must be stretched to reach their respective limits at 0 and 255 in order to maximise the range of grey levels in the image.

Input	Outputs
<code>P_sub=imsubtract(P,13);</code> <code>P2=immultiply(P_sub,255/191);</code> <code>min(P2(:)), max(P2(:))</code>	Min: 0 Max: 255

The code above aims to achieve this by using the formula: $P_{new} = \frac{P_{in} - P_{min}}{P_{max} - P_{min}} \times 255$. This way, the intensities of the new image will span the entire dynamic range. Since the input type of equation is `uint8`, the `imsubtract()` and `immultiply()` functions are used.

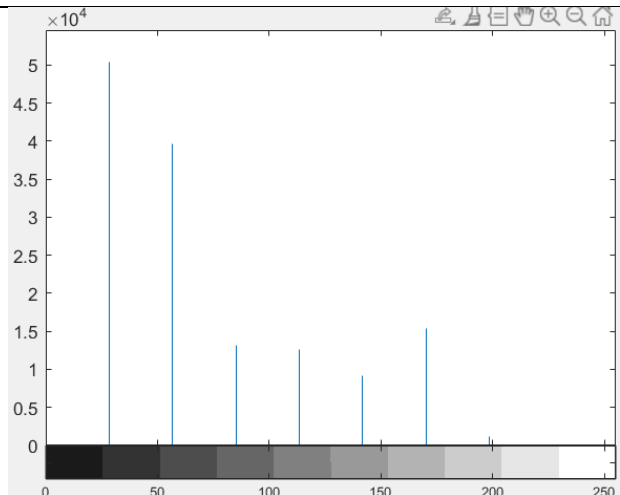
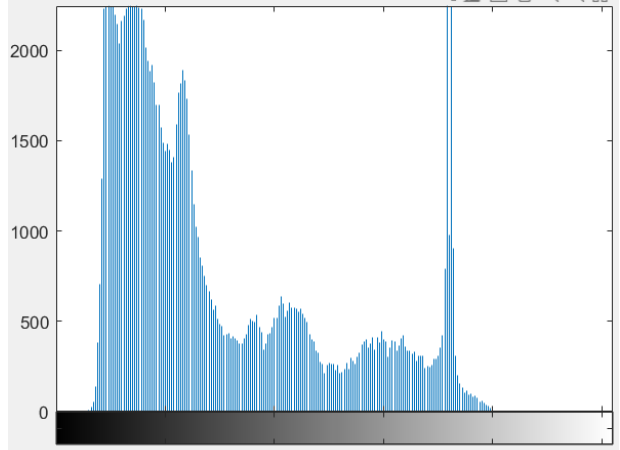
The result of contrast stretching is shown below:



In comparison to the original image in Figure 1.3, the pixels in the processed image in Figure 1.4 are able to occupy the entire grayscale range, thus appearing clearer to the naked eye.

Histogram Equalization

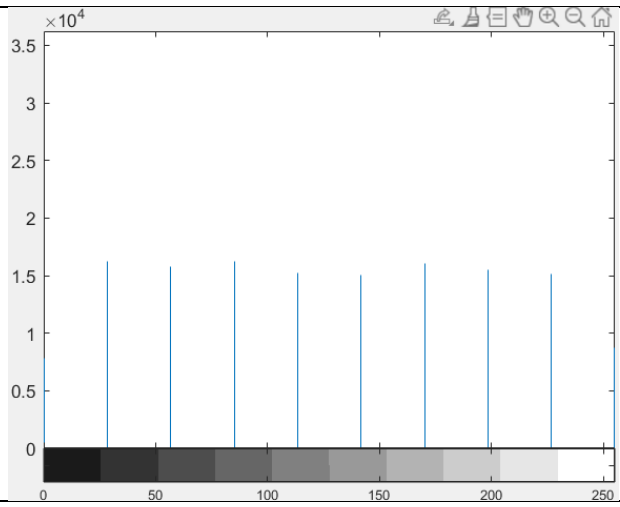
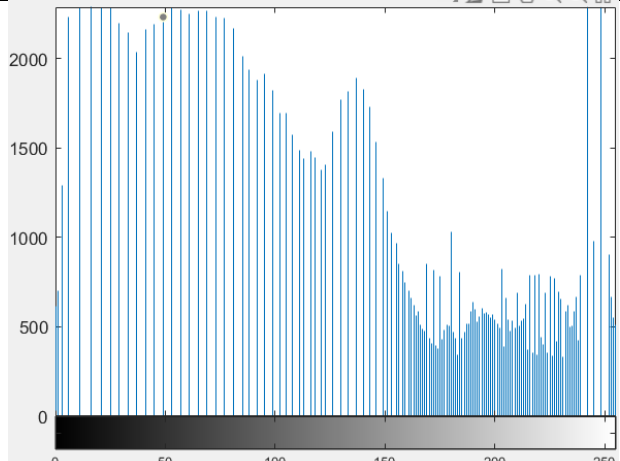
Histogram equalization is another technique for image processing. It aims to spread out the more frequent intensity values to stretch the intensity range of the image. The histograms of the original image are shown below:

Input <code>imhist(P,10);</code>	Output: Figure 2.1 Histogram of original image 10 bins 
Input <code>imhist(P,255);</code>	Output: Figure 2.2 Histogram of original image 255 bins 

a) What are the differences between the histogram with 10 bins versus the histogram with 256 bins?

- The histogram with 256 bins provides more information on the distribution of grey levels in the image than the histogram with 10 bins. This is because the range of grey levels in each bin for 256 bins is much lesser than that of 10 bins.

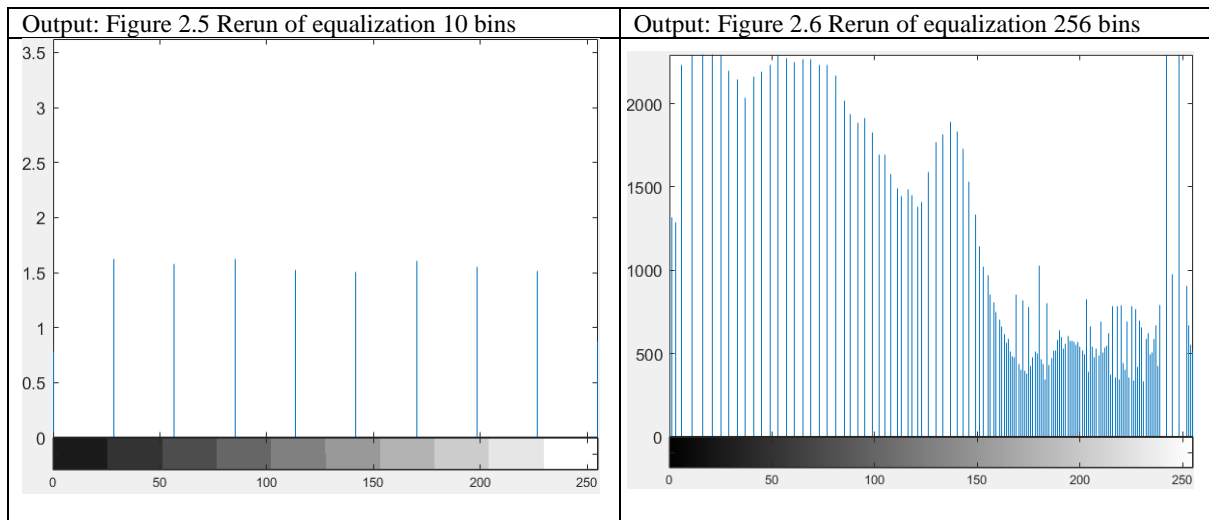
To carry out the histogram equalization, the function `histeq(P,255)` is used. The results are shown below:

Input <pre>P3=histeq(P,255); imhist(P3,10);</pre>	Output: Figure 2.3 Histogram of processed image 10 bins 
Input <pre>P3=histeq(P,255); imhist(P3,256);</pre>	Output: Figure 2.4 Histogram of processed image 255 bins 

b) Are the histograms equalized? What are the similarities and differences between the latter two histograms?

- Yes, after equalization, the histograms for both 10 bins (Figure 2.3) and 256 bins (Figure 2.4) are significantly more equalized than that of the original image.
- Both the equalized histograms occupy the maximum grey level range as can be observed from the Figures above.
- For the equalized 10 bins histogram, the frequency of pixels in each bin and the spacing in between bins is very consistent. However, the bins in the equalized 256 bins histogram is much more concentrated in the grey level range of approximately 160-240. This causes the frequency of pixels to be much higher in grey level ranges with less concentrated bins.

The histogram equalization was executed again in order to analyse its effects:



c) Does the histogram become more uniform? Give suggestions as to why this occurs.

- The histogram does not become more uniform compared to the initial equalization results. This is due to the fact that the pixels are already mapped to each bin such that the pixels are uniformly distributed throughout the grey level range. Thus, it would be redundant to run the same histogram equalization on the image multiple times.

The image results of histogram equalization are shown below:

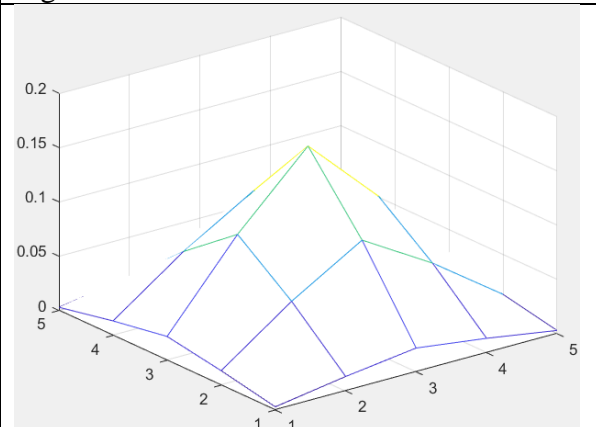
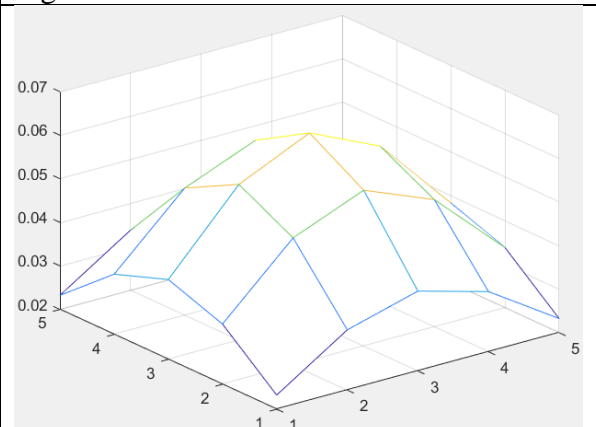


As shown in the results above, histogram equalization was able to significantly increase the contrast of the image. However, a rerun of histogram equalization will not lead to better results.

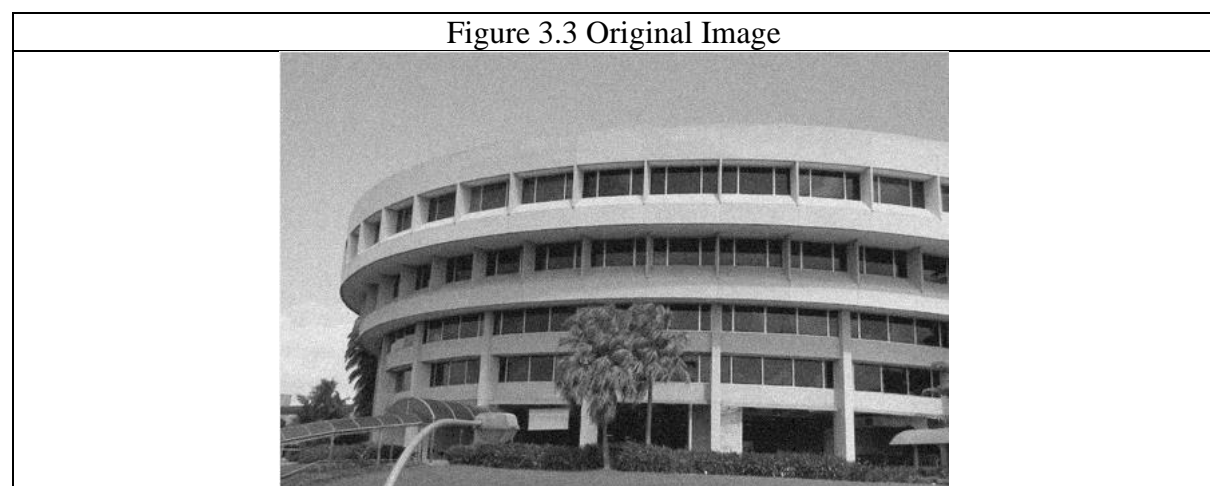
Linear Spatial Filtering

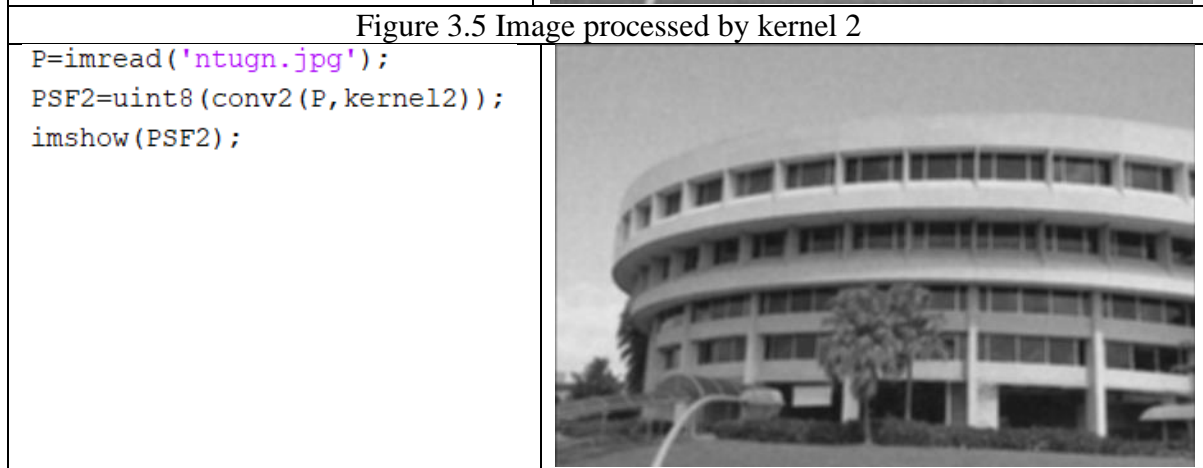
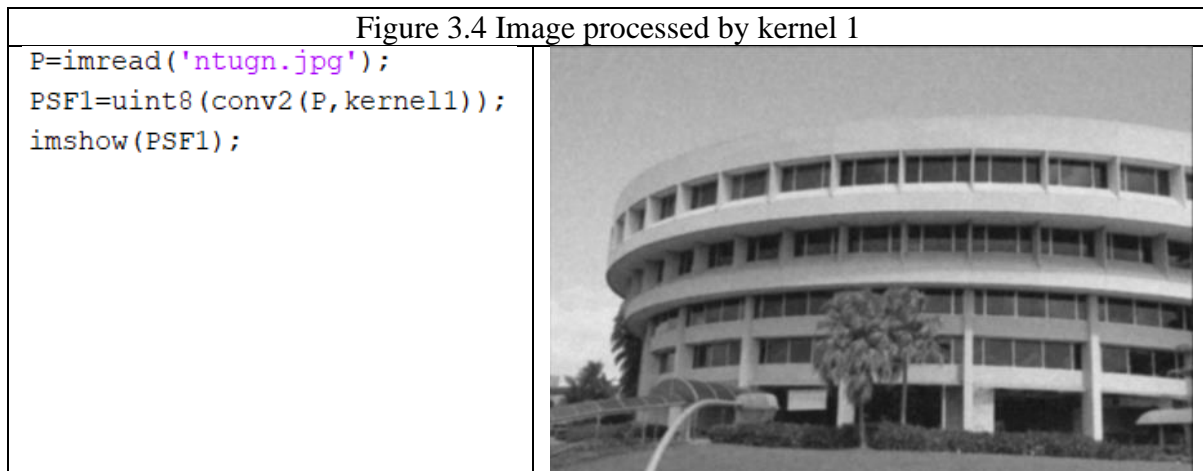
Linear spatial filtering is a type of image processing that replaces pixels with a linear function using the values of nearby pixels.

Representations and meshes of Gaussian filters are shown below:

Y and X dimensions are 5 and $\sigma = 1.0$	
Input <pre>kernel1=fspecial('gaussian',[5 5],1); figure() mesh(kernel1);</pre>	Figure 3.1 Mesh of kernel 1 
Y and X dimensions are 5 and $\sigma = 2.0$	
Input <pre>kernel2=fspecial('gaussian',[5 5],2); figure() mesh(kernel2);</pre>	Figure 3.2 Mesh of kernel 2 

Results of Gaussian filter on image with Gaussian noise using the conv2() function:

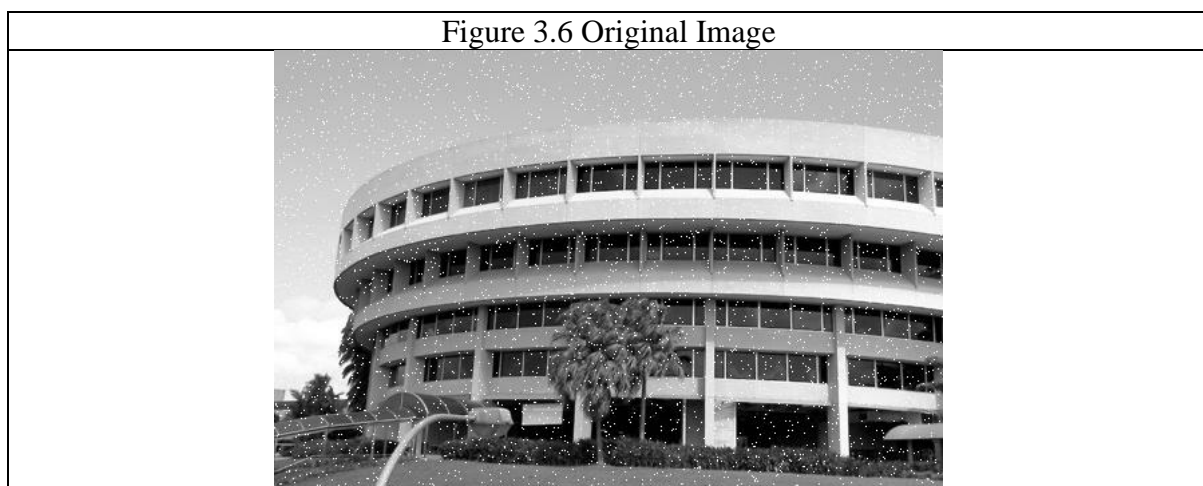




c) How effective are the filters in removing noise? What are the trade-offs between using either of the two filter, or not filtering the image at all?

- The Gaussian filters are fairly effective in reducing Gaussian noise. By comparing Figure 3.4 and 3.5 against the original image in Figure 3.3, it can be observed that increasing the value of σ will increase the effectiveness of the filter on reducing Gaussian noise, but the resultant image will become more blurred. Thus, more Gaussian noise will be filtered out at the cost of more information in the image.

Results of Gaussian filter on image with speckled noise using the conv2() function:





d) How effective are the filters in removing noise? What are the trade-offs between using either of the two filter, or not filtering the image at all?

- The Gaussian filters are not very effective in filtering speckled noise. By comparing the resultant images in Figures 3.7 and 3.8 against the original image in Figure 3.6, it is apparent that increasing the value of σ will result in a blurrier image, but the speckled noise in the background is still present. This means that the speckles on the image will not appear as jarring as the ones in the original image, but there will be a significant loss of information as a result.

e) Are the filters better at handling Gaussian noise or speckle noise?

- The Gaussian filters are significantly more effective in filtering Gaussian noise as compared to speckled noise.

Median Filtering

Median filtering is a case of order-statistic filtering that changes each pixel's intensity depending on the intensities of its neighbouring pixels.

The following are results of median filtering on an image with Gaussian noise using the `medfilt2()` command:

Figure 4.1 Original Image



Figure 4.2 Image processed median filtering with neighbourhood size of [3x3]

```
P=imread('ntugn.jpg');  
PMF1=medfilt2(P,[3 3]);  
imshow(PMF1);
```



Figure 4.3 Image processed median filtering with neighbourhood size of [5x5]

```
P=imread('ntugn.jpg');  
PMF2=medfilt2(P,[5 5]);  
imshow(PMF2);
```



The following are results of median filtering on an image with speckled noise using the `medfilt2()` command:

Figure 4.4 Original Image



Figure 4.5 Image processed median filtering with neighbourhood size of [3x3]

```
P2=imread('ntusp.jpg');  
PMF3=medfilt2(P2,[3 3]);  
imshow(PMF3);
```







Figure 4.6 Image processed median filtering with neighbourhood size of [5x5]

```
P2=imread('ntusp.jpg');  
PMF4=medfilt2(P2,[5 5]);  
imshow(PMF4);
```



Comparison of Gaussian and Median filtering on different types of noise:

Gaussian Noise, Gaussian Filtering ($\sigma=2$)	Speckle Noise, Gaussian Filtering ($\sigma=2$)
	
Gaussian Noise, Median Filtering (5x5)	Speckle Noise, Median Filtering (5x5)
	


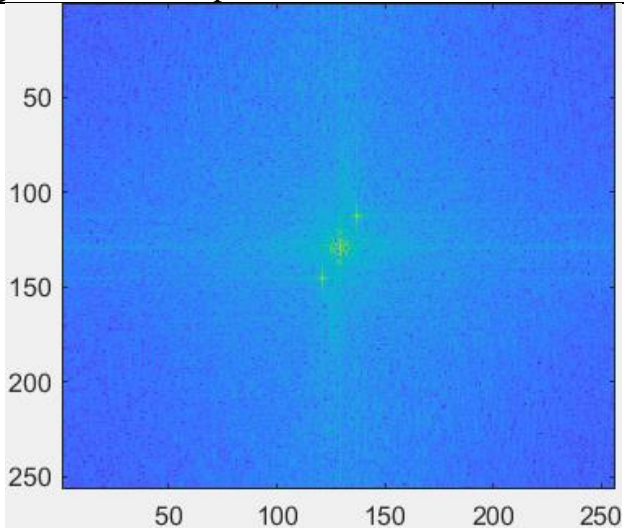
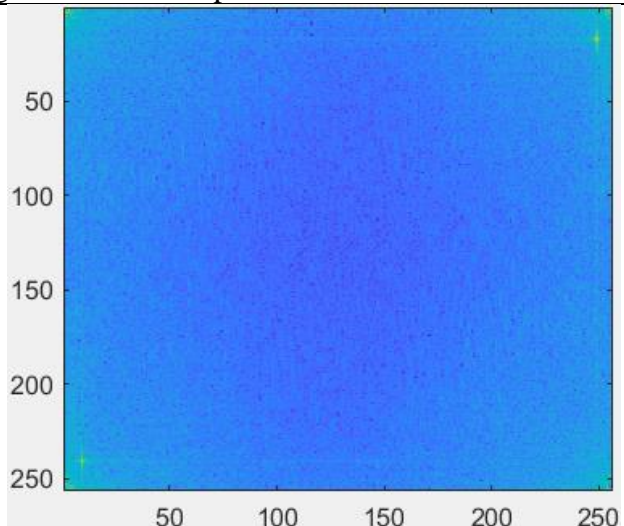
How does Gaussian filtering compare with median filtering in handling the different types of noise? What are the trade-offs?

- Median filtering is fairly effective at handling Gaussian noise, but the edges of details in the image will become more blurred as the neighbourhood size increases. However, it is extremely effective at handling speckled noise, as shown in Figure 4.5 and 4.6. It can almost perfectly filter out the speckled noise, at the cost of some blurring in the details of the image.
- It can be observed that Gaussian filtering is still slightly better at handling Gaussian noise as compared to Median filtering since it is able to retain slightly more information. However, Median filtering is significantly better than Gaussian filtering at filtering speckle noise.

Suppressing Noise Interference Patterns

Bandpass filters in image processing are used to suppress unwanted signals.

Below are the interfered image and its power spectrum:

Figure 5.1 Original Image	
	
Input with fftshift	Figure 5.2 Power Spectrum with fftshift
<pre>P=imread('pckint.jpg'); F=fft2(P); S=abs(F); imagesc(fftshift(S.^0.1)); colormap('default');</pre>	
Input without fftshift	Figure 5.3 Power Spectrum without fftshift
<pre>P=imread('pckint.jpg'); F=fft2(P); S=abs(F); imagesc(S.^0.1); colormap('default');</pre>	

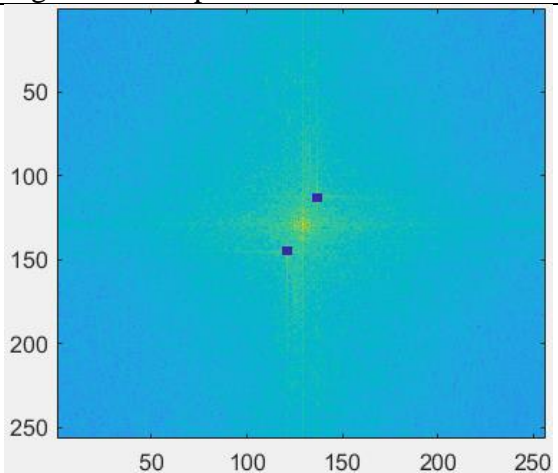
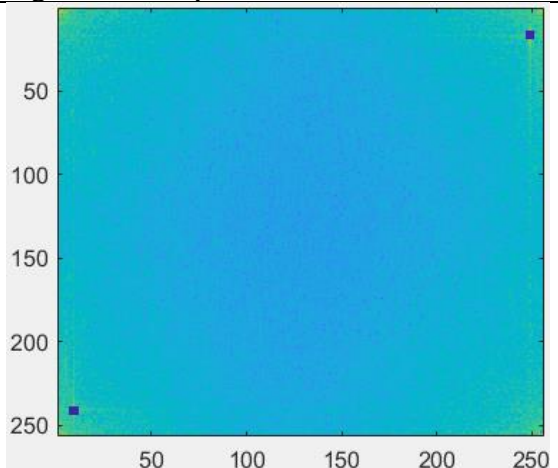
As can be observed from Figure 5.2 and 5.3, there are two distinct frequency peaks that are symmetrical to each other and isolated from the central mass. These peaks are the cause of the interference pattern in the original image (Figure 5.4). Using the `ginput` function as shown below, the coordinates of the peaks can be found.

```
[x,y]=ginput(2);
```


The x and y coordinates of the peaks are found to be:

x	y
9	241
249	17

In an attempt to suppress these interferences, a 5x5 neighbourhood of pixels around the peaks (at the coordinates stated above) are set to zero. The following shows the implementation of this action:


Code for setting neighbourhood pixels to 0	
<pre>F(239:243,7:11)=0; F(15:19,247:251)=0;</pre>	
Input	Figure 5.4 Power Spectrum with fftshift, neighbourhood pixels set to 0
<pre>S2=abs(F); imagesc(fftshift(S2.^0.1)); colormap('default');</pre>	
Input	Figure 5.5 Power Spectrum without fftshift, neighbourhood pixels set to 0
<pre>S3=abs(F); imagesc(S3.^0.1); colormap('default');</pre>	

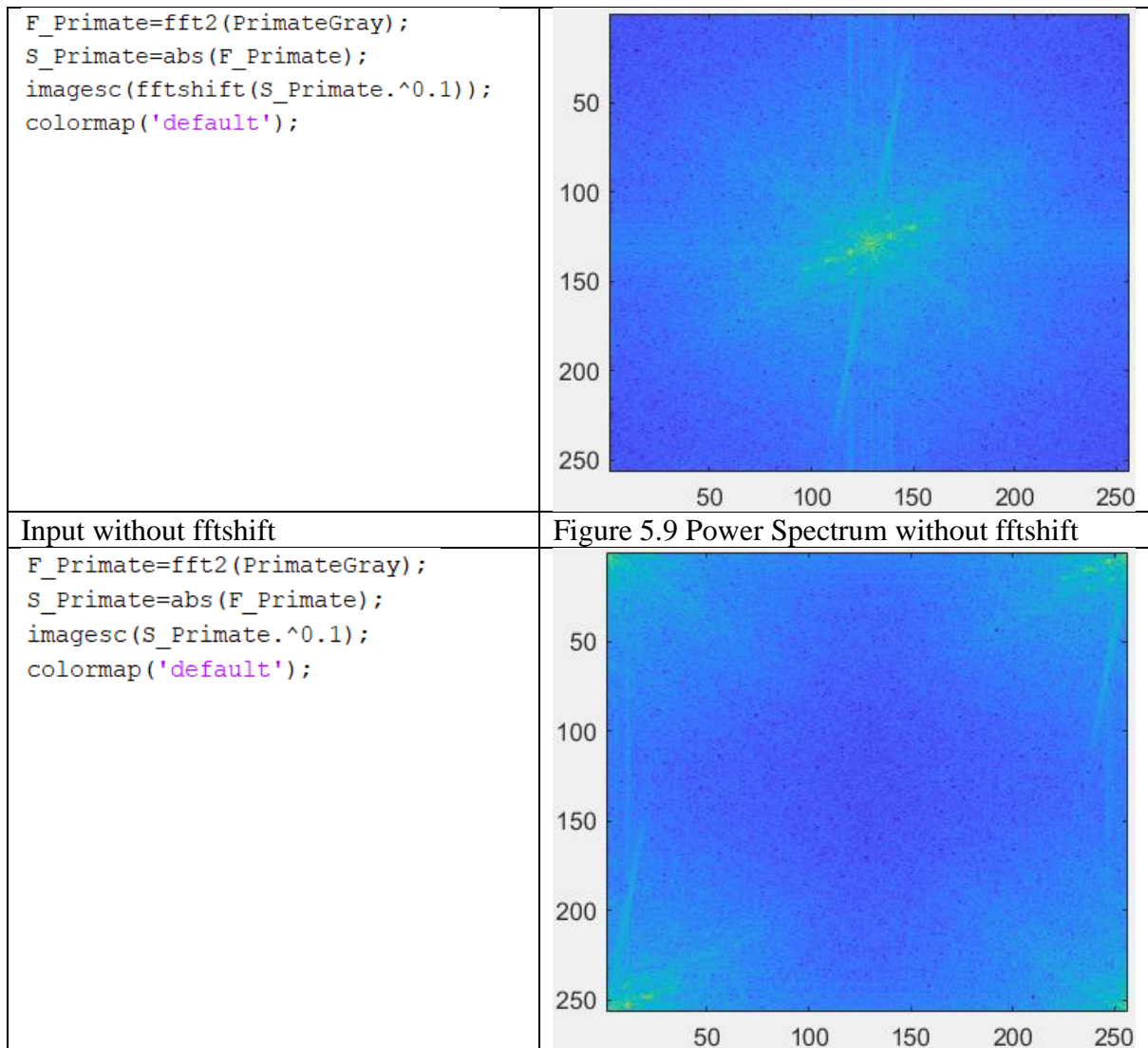
In order to display the resultant image, the `ifft2()` function is used, as shown below:

Input	Figure 5.6 Resultant image
<pre>invF=uint8(ifft2(F)); imshow(invF);</pre>	

As shown in Figure 5.6, the resultant image has significantly less interference as compare to the original image in Figure 5.1. This is because a 5x5 pixel wide area around the interference peaks are set to 0, thus effectively removing most of the interference. However, it can also be observed that there are still traces of interference left on the image. This is due to the fact that a 5x5 area is not sufficient for removing all of the interference, since there are still colours 'leaking out' from under the 5x5 patches as observed in Figures 5.4 and 5.5. A more precise job can be done (perhaps changing the shape of the patches to match that of the interference pattern) by identifying and removing areas of interference which may reduce the interference on the image even further.

In an attempt to remove the fence on the primate image, the above processes are repeated:

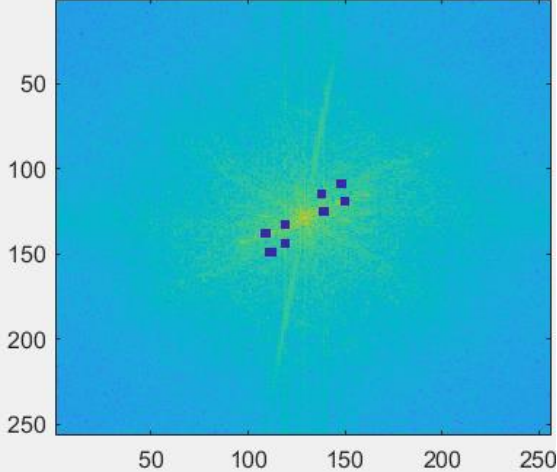
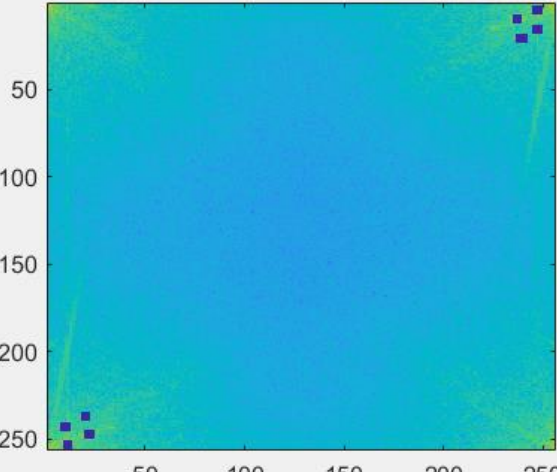
Input	Figure 5.7 Original Image
<pre>Primate=imread('primatecaged.jpg'); whos Primate PrimateGray=rgb2gray(Primate); imshow(PrimateGray);</pre>	
Input with fftshift	Figure 5.8 Power Spectrum with fftshift




It can be observed that there are peaks of interference in a linear fashion in the centre of Figure 5.8, this pattern of peaks corresponds to the ones in the bottom left corner and top right corner of the power spectrum in Figure 5.9. These are the peaks that cause the interference in the image. The `ginput` function was used to find the coordinates of most of the peaks:

x	y
11	255
247	5
237	10
22	247
10	243
20	237
247	16
240	21



In an attempt to suppress these interferences, a 5x5 neighbourhood of pixels around the peaks (at the coordinates stated above) are set to zero. The following shows the implementation of this action:

Code for setting neighbourhood pixels to 0	
<pre> F_Primate(251:255,9:13)=0; F_Primate(3:7,245:249)=0; F_Primate(8:12,235:239)=0; F_Primate(245:249,20:24)=0; F_Primate(241:245,8:12)=0; F_Primate(235:239,18:22)=0; F_Primate(14:18,245:249)=0; F_Primate(19:23,237:242)=0; </pre>	
Input	Figure 5.10 Power Spectrum with fftshift, neighbourhood pixels set to 0
<pre> S2_Primate=abs(F_Primate); imagesc(fftshift(S2_Primate.^0.1)); colormap('default'); </pre>	
Input	Figure 5.11 Power Spectrum without fftshift, neighbourhood pixels set to 0
<pre> S2_Primate=abs(F_Primate); imagesc(S2_Primate.^0.1); colormap('default'); </pre>	

In order to display the resultant image, the `ifft2()` function is used, as shown below:

Input	Figure 5.12 Resultant image
<pre>invF_Primate=uint8(real(ifft2(F_Primate))); imshow(invF_Primate);</pre>	

The resultant image compared to the original image:

Original	Resultant
	

It can be observed from the comparison above that the fence in the resultant image is much less prominent than the fence in the original image. However, this result is not perfect since the outlines of the fence is still present in the resultant image. This may be due to the fact that the fence pattern of the fence is not perfectly straight, thus making it difficult to determine where the interference peaks are in the power spectrum.

Undoing Perspective Distortion of Planar Surface

One way to fix distortion on an image is with matrix transformations. A 2D planar transform can be expressed as:

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


Equation 6.1 2D Planar projective transform

In the equation 6.1, the desired x and y values can be expressed in the following form:

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

Equation 6.2 projective transform in terms of desired x and y

Given an image of a distorted book, the above equations can be used to undo the perspective distortions:

Input	Figure 6.1 Original Image
<pre>book=imread('book.jpg'); whos book imshow(book); Book=rgb2gray(book); imshow(Book);</pre>	

In order to find the approximated coordinates of the four corners of the distorted book,

[X Y]=ginput(4) is used, the results as shown below:

X	Y
142	28
309	47
257	216
4	160

Since the final X and Y values must adhere to the dimensions of an A4 page (210mm x 297mm), the desired X and Y values can be set as:

X final	Y final
0	0
210	0
210	297
0	297


Knowing the above values, the matrix below (Equation 6.3) can be used to transform the original image to the desired final image:

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

Equation 6.3 matrix for projective transformation, $Au=v$

The implementation of Equation 6.3 in matlab is as follows:



A4 paper dimensions	
<pre>%A4 paper dimensions (210*297mm) x_final=[0,210,210,0]; y_final=[0,0,297,297];</pre>	
Implementation of vector v	
<pre>v=[x_final(1); y_final(1); x_final(2); y_final(2); x_final(3); y_final(3); x_final(4); y_final(4)];</pre>	
Implementation of vector A and Equation 6.3	
<pre>A=[X(1),Y(1),1,0,0,0,-x_final(1)*X(1),-x_final(1)*Y(1); 0,0,0,X(1),Y(1),1,-y_final(1)*X(1),-y_final(1)*Y(1); X(2),Y(2),1,0,0,0,-x_final(2)*X(2),-x_final(2)*Y(2); 0,0,0,X(2),Y(2),1,-y_final(2)*X(2),-y_final(2)*Y(2); X(3),Y(3),1,0,0,0,-x_final(3)*X(3),-x_final(3)*Y(3); 0,0,0,X(3),Y(3),1,-y_final(3)*X(3),-y_final(3)*Y(3); X(4),Y(4),1,0,0,0,-x_final(4)*X(4),-x_final(4)*Y(4); 0,0,0,X(4),Y(4),1,-y_final(4)*X(4),-y_final(4)*Y(4)]; u=A\v;</pre>	
Convert resultant u into normal matrix form U	
Input	Output

<code>U=reshape([u;1],3,3)';</code>	1.4554 1.5347 -249.9900 -0.4248 3.6813 -42.0748 0.0002 0.0052 1.0000
Verify the correctness of U	
Input	Output
<code>w=U*[X';Y';ones(1,4)];</code> <code>w=w./(ones(3,1)*w(3,:))</code>	$w =$ 0.0000 210.0000 210.0000 0.0000 0 0.0000 297.0000 297.0000 1.0000 1.0000 1.0000 1.0000
Warp and display the resultant image	
Input	Output: Figure 6.2 Resultant image
<code>T=makeform('projective',U);</code> <code>Book2=imtransform(Book,T,'XData',[0 210],'YData',[0 297]);</code> <code>imshow(Book2);</code>	

c) Does the transformation give you back the 4 corners of the desire image?

- Yes, w gives back $[0 \ 210 \ 210 \ 0; 0 \ 0 \ 297 \ 297; 1 \ 1 \ 1 \ 1]$, which is the 4 coordinates of the desired result.

Comparisons of the resultant image versus the original image:

Original	Result
	

e) Is this what you expect? Comment on the quality of the transformation and suggest reasons.

- Yes, this is similar to the expected result of the transformation. The four corners of the distorted image are mapped to the correct locations in the resultant image and the result is in A4 dimensions.
- Overall, the proportions of the resultant image is adequately close to the actual proportions of the original book
- However, the quality of the resultant image seems to be subpar, especially towards the upper left-hand corner. This may be due to resolution being worse in that corner of the book in the original image, which causes this error to be present in the resultant image as well. This problem may be resolved if the original image was better resolved.