



**NANYANG**  
**TECHNOLOGICAL**  
**UNIVERSITY**

**CZ4003 Computer Vision**

*Lab 1: Point Processing + Spatial Filtering +  
Frequency Filtering + Imaging Geometry*

Yong Hao

U1722282A

## Contents

1. Objectives.....	3
2. Experiments.....	3
2.1. Contrast Stretching .....	3
2.1.a. Read the image ‘mrt_train’ .....	3
2.1.b. Investigate different methods for improving the image using point processing.....	4
2.1.c. Check the minimum and maximum intensities present in the image .....	4
2.1.d. Applying contrast stretching:.....	5
2.1.e. Showing the image: .....	5
2.2. Histogram Equalization .....	6
2.2.a. Display the intensity histogram of P:.....	6
2.2.b. Histogram Equalization .....	6
2.2.c. Return Histogram Equalization .....	7
2.3 Linear Spatial Filtering.....	8
2.3.a. Generate and Normalize Filters .....	8
2.3.b. View ‘ntu-gn.jpg’ .....	9
2.3.c. Filter the Image.....	10
2.3.d. Read image ‘ntu-sp’ .....	11
2.3.e. Filter ‘ntu-sp’ Using Gaussian Filters.....	11
2.4. Median Filtering .....	12
2.4.a. Filter the Gaussian Noise Image and Speckle Noise Image with Median Filtering .....	12
2.5. Suppressing Noise Interference Patterns .....	15
2.5.a. Display the image ‘pck-int.jpg’ .....	15
2.5.b. Obtain Fourier Transform and Display Power Spectrum .....	15
2.5.c. Redisplay the Power Spectrum without fftshift and Measure Locations of Peaks. ....	16
2.5.d. Set zero to 5*5 neighbourhood elements.....	17
2.5.e. Compute Inverse Fourier Transform and Display Resultant Image .....	18
2.5.f. Free the Primate from a Fence .....	19
2.6. Undoing Perspective Distortion of Planar Surface.....	21
2.6.a. Read in and display the ‘book’ image .....	21
2.6.b. Get 4 Corners’ Positions and Desired Locations.....	21
2.6.c. Set up the Matrix and Estimate Projective Transformation.....	21
2.6.d. Wrap the image and Display it .....	22

## 1. Objectives

This laboratory aims to introduce image processing in MATLAB context. In this laboratory we will:

1. Become familiar with the MATLAB and Image Processing Toolbox software package.
2. Experiment with the point processing operations of contrast stretching and histogram equalization.
3. Evaluate how different Gaussian and median filters are suitable for noise removal.
4. Become familiar with the frequency domain operations
5. Understand imaging geometry.

## 2. Experiments

### 2.1. Contrast Stretching

#### 2.1.a. Read the image 'mrt\_train'

By running the code below, we read in the MRT train image:

```
% read in the image mrt_train
Pc = imread('mrt_train.jpg');
whos Pc
imshow(Pc)

% converting the mrt_train to grayscale image
P = rgb2gray(Pc);
whos P
imshow(P)
```

The 'whos' function in MATLAB lists the information of a variable, including the name, size, bytes, class, and attributes. We can find out the information of the image variable Pc and P (Pc converted to grayscale) is:

Name	Size	Bytes	Class	Attributes
Pc	320x443x3	425280	uint8	
P	320x443	141760	uint8	

We noticed the size of P has become 2-dimensional and the bytes value is smaller than the bytes value of Pc, meaning the conversion to grey scale has been successfully conducted.

The result image representing P is given below:



### 2.1.b. Investigate different methods for improving the image using point processing

The definition of point processing is the pixel value of the enhanced image point is solely determined by the pixel value of the original image point. Three point processing image enhancement techniques introduced in the course: **contrast stretching**, **histogram equalization**, and **power-law transformations**.

1. **Contrast stretching:** Contrast stretching improved the contrast of an image by mapping the current intensity values to span a new desired range of values, for example, mapping a image whose lowest intensity value is 20 and highest intensity value 220 to the full range of 8-bit grayscale image [0, 255]
2. **Histogram equalization:** histogram equalization is a more sophisticated method than contrast stretching as contrast stretching is greatly affected by outliers on the bounds, thus making the adjusted image less representative. Histogram equalization enhances the image by flattening the intensity values in equally splitted bins to a wider range.
3. **Power-law transformation:** Power-law transformations maps the input image intensity to an exponential curve using two parameters,  $b$  and  $c$ . The output is given by taking the  $b$  to the power of the original pixel intensity and multiply it by  $c$ .
- 4.

### 2.1.c. Check the minimum and maximum intensities present in the image

By running the following function, we acquired the maximum and minimum intensity values of the original image:

```
% checking the maximum and minimum intensity values of P
min(P(:))
max(P(:))
```

The maximum and minimum intensity values are given by:

<code>min(P(:))</code>	<code>ans = uint8 13</code>
<code>max(P(:))</code>	<code>ans = uint8 204</code>

#### 2.1.d. Applying contrast stretching:

To map the original intensity values to range [0,255], we apply the following formula. 's' denotes the output intensity. 'r' is the intensity of the original input image point. 'rmin' is the minimum intensity value in the original image, 'rmax' is the maximum intensity value of the original image.

$$s = \frac{255(r - r_{min})}{r_{max} - r_{min}}$$

The formula can be implemented by breaking it into one subtraction and one multiplication as below:

```
% implementing contrast stretching by using one subtraction and one
% multiplication
P2 = imsubtract(P, 13);
P2 = immultiply(P2, (255/(204-13)));

% checking the maximum and minimum intensity of the enhanced image
min(P2(:))
max(P2(:))
```



After checking the minimum and maximum intensities, the values are given by:

<code>min(P(:))</code>	<code>ans = uint8 0</code>
<code>max(P(:))</code>	<code>ans = uint8 255</code>

#### 2.1.e. Showing the image:

```
imshow(P2)
```

The comparison between the result image after contrast stretching and the original image is given below:

Original Image	Result Image after Contrast Stretching
	

## 2.2. Histogram Equalization

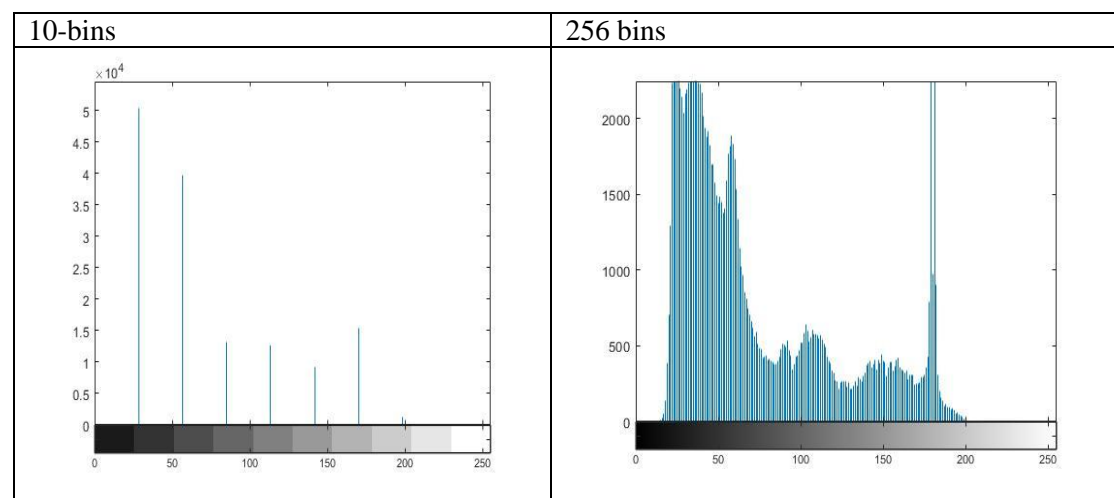
### 2.2.a. Display the intensity histogram of P:

By using the following functions, we can display the intensity histogram using 10 bins and 256 bins respectively:

```
% read in the image
Pc = imread('mrt_train.jpg');
P = rgb2gray(Pc);

% displaying the histogram into 10 and 256 bins
figure;
imhist(P, 10);
figure;
imhist(P, 256);
```

The results are given below:



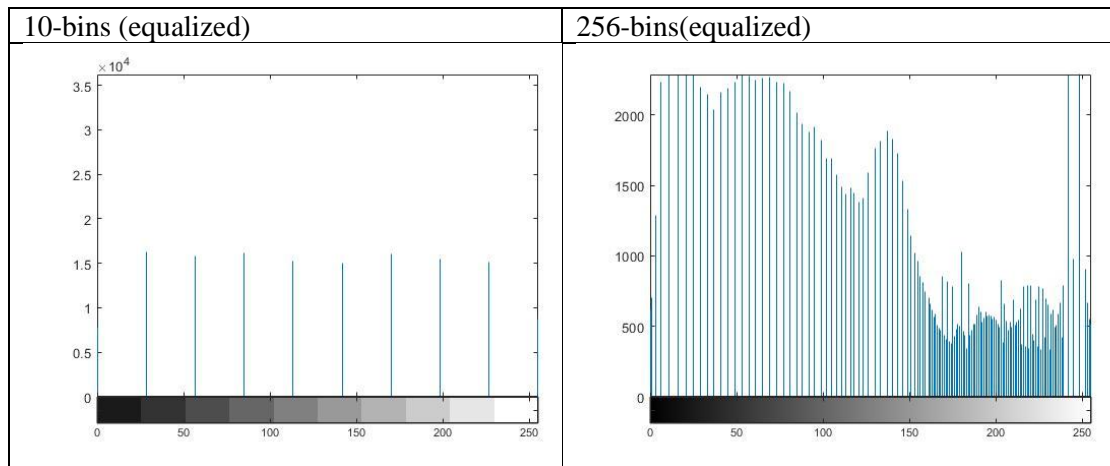
The differences of dividing the intensities into 256 bins and 10 bins include:

1. The 10-bin histogram divides all pixels into 10 equal-interval bins, while 256-bin histogram divides them into 256.
1. Compared to 256-bin histogram, the average number of pixels in each bin in the 10-bin histogram is significantly greater than the average number of pixels in the 256-bin image.
2. Compared to 10-bin histogram, the 256-bin image contains more detailed information regarding the intensity distribution of the image.

### 2.2.b. Histogram Equalization

By running the function below and displaying the output histograms, we acquired the following outputs:

```
% applying histogram equalization and displaying the output histogram
P3 = histeq(P, 255);
figure;
imhist(P3, 10);
figure;
imhist(P3, 256);
figure;
imshow(P3)
```

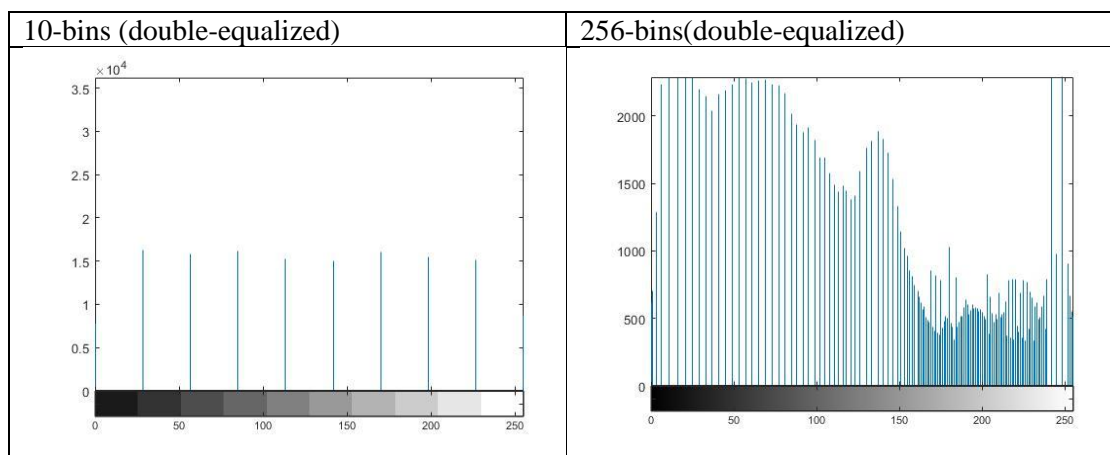


By observing the equalized histograms with their original ones, we can confirm that the histograms are equalized since we can observe the histograms are more flattened than the original ones. In the equalized 10 bin image, each bin has almost the same number of pixels. In the equalized 256-bin images, the number of pixels in each bin has certain variances, but is still more “flattened” than the original histogram, and we can observe that some pixel intensities are mapped to intensities that does not exist in the original histogram.

### 2.2.c. Return Histogram Equalization

By reapplying the histogram equalization function to P3, we acquired the following double-equalized histograms:

```
%applying the histogram equalization function again and check if they
are more uniform
P4 = histeq(P3, 255);
figure;
imhist(P4, 10);
figure;
imhist(P4, 256);
figure;
imshow(P4)
```



We can observe that re-equalized histograms are the same with single-equalized histograms.

The reason why this occurs is because histogram equalization is idempotent, because the output intensity  $s$  follows the following formula:

$$s = (L - 1) \int_0^r P(r) dr$$

Where  $L$  is the maximum intensity of the output image,  $r$  is the intensity of the input image, and  $P(r)$  is the probability of the intensity value  $r$ . If we reapply the formula to  $s$ , we will get the new output  $s'$  as follows:

$$s' = (L - 1) \int_0^s P(s) ds$$

Since  $r$  is mapped to  $s$  according to its cumulative distribution, thus  $\int_0^s P(s) ds$  has the same value as  $\int_0^r P(r) dr$ , since  $s$  is only a linear transformation of  $\int_0^r P(r) dr$ . Thus,  $s'$  is equal to  $s$ , which implies the double-equalized output is the same as the single-equalized output. The output image is as below:



## 2.3 Linear Spatial Filtering

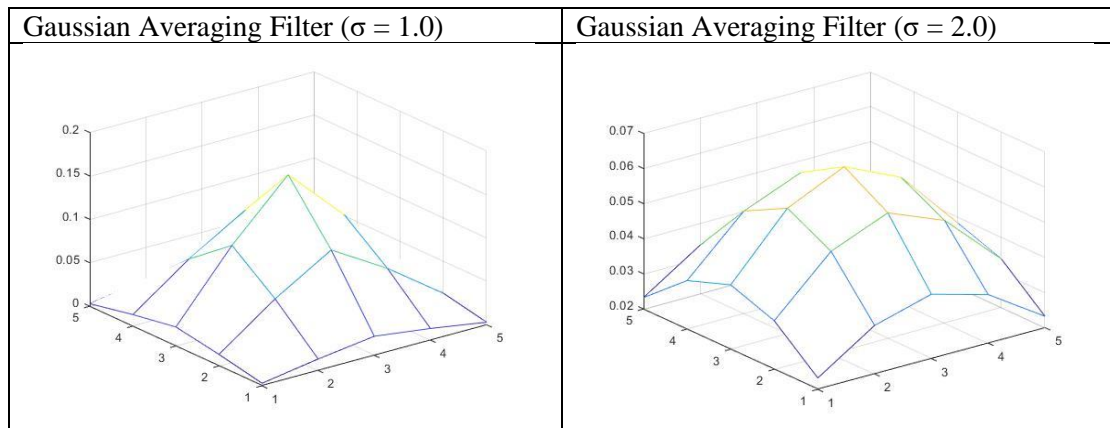
### 2.3.a. Generate and Normalize Filters

By running the function below, we can generate and normalize two Gaussian averaging filters with standard deviation 1 and 2 respectively, and view them as 3-D graphs:



```
% crasting the filters, the second argument in fspecial is regarded as
a matrix [5 5]
h1 = fspecial('gaussian',5,1);
h1 = (h1/sum(h1(:)));
figure;
mesh(h1);

h2 = fspecial('gaussian',5,2);
h2 = (h2/sum(h2(:)));
figure;
mesh(h2);
```



### 2.3.b. View 'ntu-gn.jpg'

By running the functions below, we can read in and view the 'ntu-gn' image:

```
% read in the image
gn = imread('ntu_gn.jpg');
figure;
imshow(gn);
```

The result image is:



### 2.3.c. Filter the Image

By running the functions below, we can filter the image ('ntu-gn') using the Gaussian averaging filters generated above:

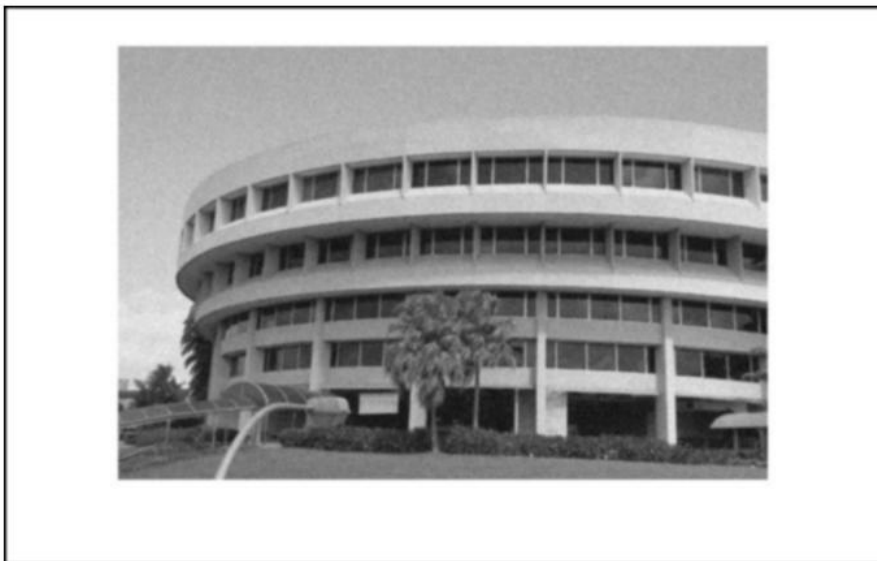
```
% filter the image using filters created above
P1 = uint8(conv2(gn, h1));
figure;
imshow(P1);

P2 = uint8(conv2(gn, h2));
figure;
imshow(P2);
```

Result Image Filtered by Gaussian Averaging Filter ( $\sigma = 1.0$ )



Result Image Filtered by Gaussian Averaging Filter ( $\sigma = 2.0$ )



The Gaussian filters are effective in removing the Gaussian noise. The higher the standard deviation, the better the filter is for removing Gaussian noise.

The trade-offs of using the two filters or do not filter at all, is that the filter with higher standard deviation we use, we can have a better effect in removing the Gaussian noise, but the result image will be more blurred, thus suffer information loss.

#### 2.3.d. Read image 'ntu-sp'

By running the function below we can read in the image 'ntu-sp' and display it:

```
%read in the image 'ntu-sp' and display it
sp = imread('ntu_sp.jpg');
figure;
imshow(sp);
```

The result image is:



#### 2.3.e. Filter 'ntu-sp' Using Gaussian Filters

By running the functions below, we can filter the image ('ntu-sp') using the Gaussian averaging filters generated above:

```
% filter the image 'ntu-sp' and display it
P3 = uint8(conv2(sp, h1));
figure;
imshow(P3);

P4 = uint8(conv2(sp, h2));
figure;
imshow(P4);
```

Result Image Filtered by Gaussian Averaging Filter ( $\sigma = 1.0$ )



Result Image Filtered by Gaussian Averaging Filter ( $\sigma = 2.0$ )



Since we can still observe apparent speckle noise after filtering, we can conclude that Gaussian filters are better at filtering Gaussian noise than speckle noise.

## 2.4. Median Filtering

### 2.4.a. Filter the Gaussian Noise Image and Speckle Noise Image with Median Filtering

By running the functions below, we can acquire the filtered image with different median filtering settings:

```

% reading in the image gn, sp and filter them with medfilt2 size 3 and
5
gn = imread('ntu_gn.jpg');
figure
imshow(gn);

P1 = medfilt2(gn,[3 3]);
figure
imshow(P1);

P2 = medfilt2(gn,[5 5]);
figure
imshow(P2);

sp = imread('ntu_sp.jpg');
figure
imshow(sp);

P3 = medfilt2(sp,[3 3]);
figure
imshow(P3);

P4 = medfilt2(sp,[5 5]);
figure
imshow(P4);

```

'ntu-gn' filtered by median filter with size 3



'ntu-gn' filtered by median filter with size 5



'ntu-sp' filtered by median filter with size 3



'ntu-sp' filtered by median filter with size 5





As we can observe from the result image, median filter is effective for removing speckle noise since speckle noise tend to be white dots (intensity value 0) which can be mediated by taking the mean from its neighborhoods.

Compared with Gaussian filter, median filter is better at filtering speckle noise than Gaussian filter, while median filter performs less impressive when filtering Gaussian noise than Gaussian filter as we can observe that the Gaussian noise remain detectable after applying median filter.

The trade-off of whether to use a median filter or Gaussian filter includes:

1. Median filter and Gaussian filter works differently regarding different types of noises. Median filter works better for speckle noise and Gaussian filter works better for Gaussian noise.
2. Gaussian filters with higher standard deviation tend to have better effect when filtering both speckle noise and Gaussian noise but will make the entire image more blurred. More details and information might be lost with higher standard deviation.
3. Median filters with higher filter size tend to have better effect when filtering both speckle noise and Gaussian noise but will make the edges of the image more blurred, thus loss of information occurs.

## 2.5. Suppressing Noise Interference Patterns

### 2.5.a. Display the image 'pck-int.jpg'

By running the functions below, we can display the image:

```
% read in and display the image
P = imread('pck_int.jpg');
figure
imshow(P)
```

The result image is:

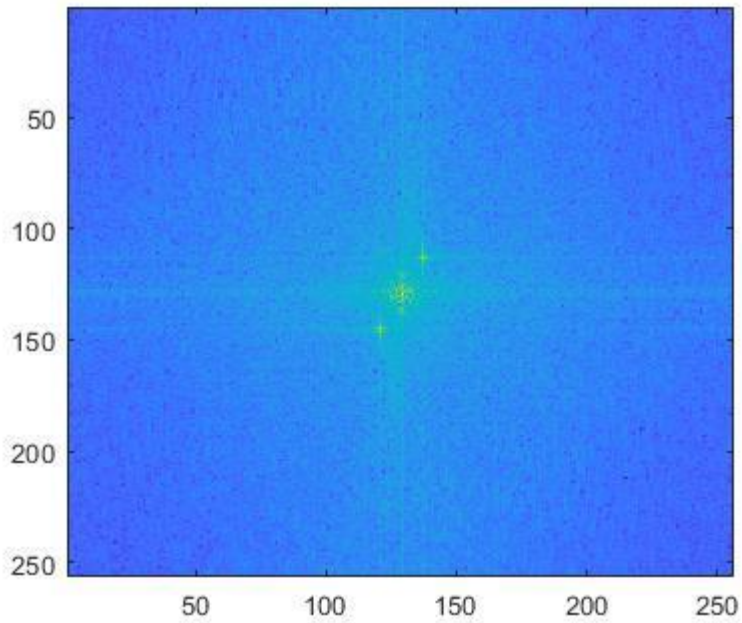


### 2.5.b. Obtain Fourier Transform and Display Power Spectrum

By running the functions below, we can obtain the Fourier transform  $F$  and its power spectrum  $P$ :

```
% Obtain Fourier transform F of image P, and compute spectrum
F = fft2(P);
S = abs(F);
figure;
imagesc(fftshift(S.^0.1));
colormap('default');
```

The result power spectrum is:

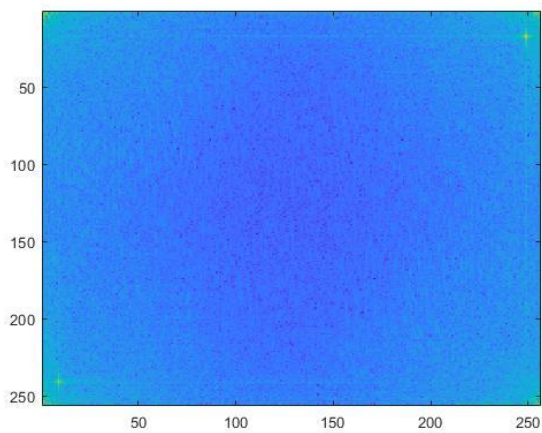


### 2.5.c. Redisplay the Power Spectrum without fftshift and Measure Locations of Peaks.

By running the function below, we can obtain the power spectrum without fftshift:

```
%display the power spectrum without fftshift
figure;
imagesc(S.^0.1);
colormap('default')
[x, y] = ginput(2);
```

The result power spectrum is:





By using **ginput**, we can find the locations of the peaks:

Name ▲	Value
ans	[248.5369,15.8003;9.0...
F	256x256 complex dou...
gn	333x500 uint8
h1	5x5 double
h2	5x5 double
P	256x256 uint8
P1	333x500 uint8
P2	333x500 uint8
P3	333x500 uint8
P4	333x500 uint8
Pc	320x443x3 uint8
S	256x256 double
sp	333x500 uint8
x	[249.0161;8.8441]
y	[16.3166;240.6834]

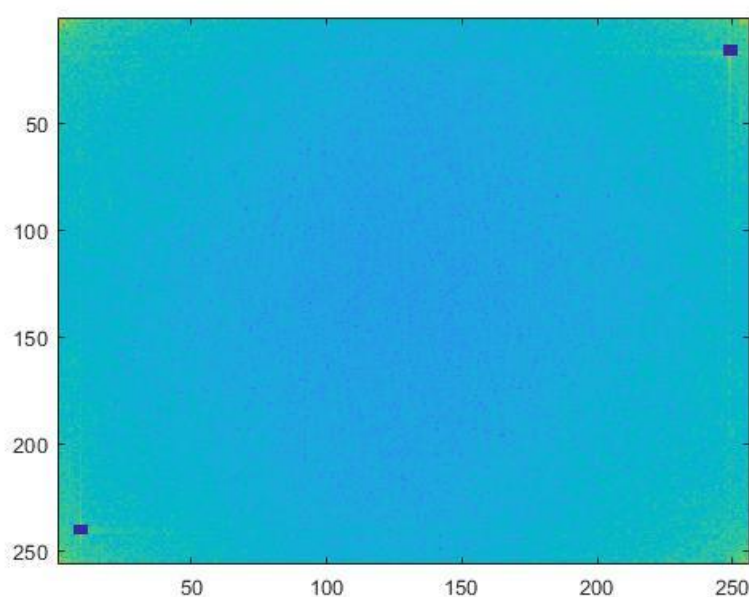
Thus, we can derive the positions of the two peaks are (249, 16) and (9, 240).

#### 2.5.d. Set zero to 5\*5 neighbourhood elements

By running the following functions, we set the 5\*5 neighborhood of peaks to zeros, then recompute the power spectrum and display it:

```
% set zeros to neighboring elements
x1 = 249; y1 = 16;
x2 = 9; y2 = 240;
F1 = F;
F1(y1-2:y1+2, x1-2:x1+2) = 0;
F1(y2-2:y2+2, x2-2:x2+2) = 0;
S1 = abs(F1);
figure;
imagesc(S1.^0.1);
colormap('default');
```

The result power spectrum is:



### 2.5.e. Compute Inverse Fourier Transform and Display Resultant Image

By running the functions below, we compute the inverse Fourier transform using **ifft2** and display the image:

```
% inverse the FT and display the image
P2 = uint8(ifft2(F1));
figure;
imshow(real(P2))
```

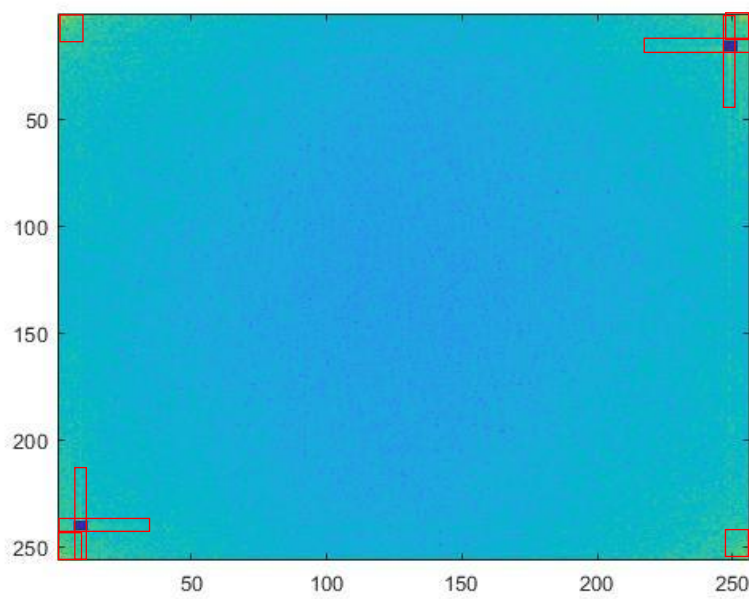
The result image is:



We can observe the interference pattern has been significantly reduced. This is related to part ‘c’ because we removed the values surrounding peak values by setting the values to 0, thus removing the interference pattern represented by the peaks. Thus, when we do the inverse Fourier transform, we can observe that the interference pattern has been significantly enhanced.

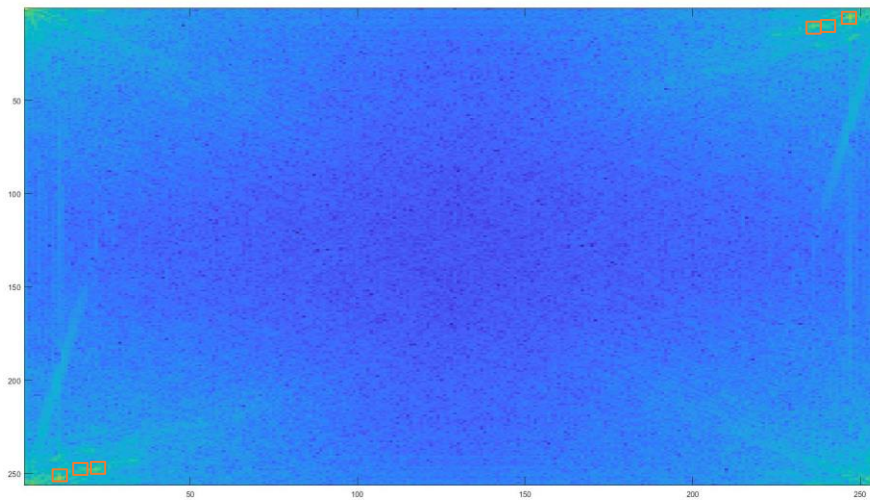
Suggestions of enhancements:

Observing the power spectrum after removing the 5\*5 neighborhood of peaks, we can notice that some peak areas still exist in the cross shape around the peaks. Also, some peaks reside in the front left corner and bottom right corner. By removing the aforementioned parts (marked in red circles below), we can expect a further improvement to the interference.



#### 2.5.f. Free the Primate from a Fence

By running the following code, we can acquire the Fourier transform of the primate image, and locate the peaks (marked in orange squares below) using **ginput**:



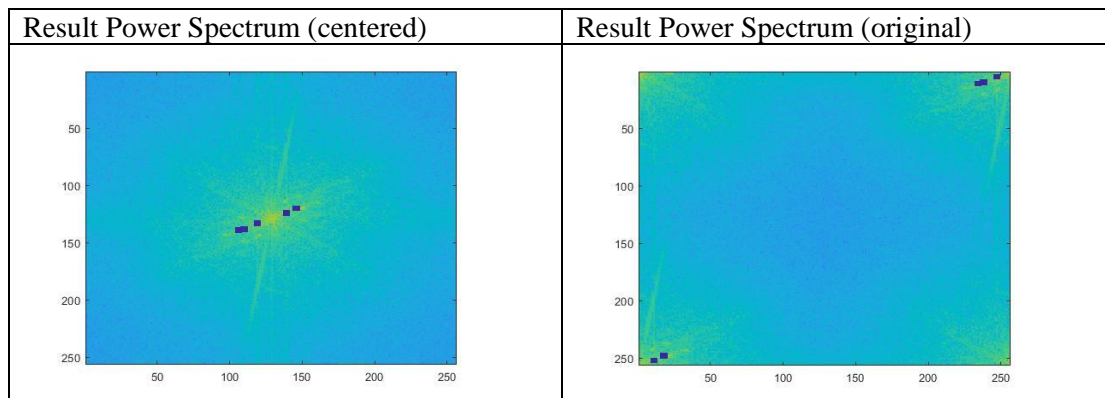
From the above image we located 6 peak points:

x	y
11	252
22	248
18	248
247	5
238	10
234	11

Then we remove the peaks and display the power spectrum:

```
% remove the peaks
x1 = 11; y1 = 252;
x2 = 22; y2 = 248;
x3 = 18; y3 = 248;
x4 = 247; y4 = 5;
x5 = 238; y5 = 10;
x6 = 234; y6 = 11;
F1 = F;
F1(y1-2:y1+2, x1-2:x1+2) = 0;
F1(y2-2:y2+2, x2-2:x2+2) = 0;
F1(y3-2:y3+2, x3-2:x3+2) = 0;
F1(y4-2:y4+2, x4-2:x4+2) = 0;
F1(y5-2:y5+2, x5-2:x5+2) = 0;
F1(y6-2:y6+2, x6-2:x6+2) = 0;
S1 = abs(F1);
figure;
imagesc(fftshift(S1.^0.1));
colormap('default');
figure;
imagesc(S1.^0.1);
colormap('default');

% inverse the FT and display the image
P2 = uint8(iff2(F1));
figure;
imshow(real(P2))
```



Then we reverse the Fourier transform and obtain the result image:



We can observe that the cages have been partially lightened for the primate.

## 2.6. Undoing Perspective Distortion of Planar Surface

### 2.6.a. Read in and display the 'book' image

By running the functions below, we can read in the 'book' image and display it:

```
% read in the image
P = imread('book.jpg');
P = rgb2gray(P);
figure;
imshow(P);
```

The result image is:



### 2.6.b. Get 4 Corners' Positions and Desired Locations

By running the functions below, we can find the 4 corners of the image and set up the A4 desired location:

```
% get the corners of the image
[x,y] = ginput(4);
x_desired = [0;210;210;0];
y_desired = [0;0;297;297];
```

The output coordinates are:

x	[142.8631;308.5115;255.9467;3.4971]
x_desired	[0;210;210;0]
y	[27.6470;47.0130;216.1196;159.0591]
y_desired	[0;0;297;297]

The corners locations are:

x	y
143	28
309	47
256	216
3	159

### 2.6.c. Set up the Matrix and Estimate Projective Transformation

By running the code below, we set up the matrix and examine if the transformation is correct by verifying the 4 corners of the image:

```
% set_up the matrix and verify the result
A = [x(1),y(1),1,0,0,0,-x_target(1)*x(1),-x_target(1)*y(1)];
    [0,0,0,x(1),y(1),1,-y_target(1)*x(1),-y_target(1)*y(1)];
    [x(2),y(2),1,0,0,0,-x_target(2)*x(2),-x_target(2)*y(2)];
    [0,0,0,x(2),y(2),1,-y_target(2)*x(2),-y_target(2)*y(2)];
    [x(3),y(3),1,0,0,0,-x_target(3)*x(3),-x_target(3)*y(3)];
    [0,0,0,x(3),y(3),1,-y_target(3)*x(3),-y_target(3)*y(3)];
    [x(4),y(4),1,0,0,0,-x_target(4)*x(4),-x_target(4)*y(4)];
    [0,0,0,x(4),y(4),1,-y_target(4)*x(4),-y_target(4)*y(4)];];
v =
[x_target(1);y_target(1);x_target(2);y_target(2);x_target(3);y_target(3)
;x_target(4);y_target(4)];
u =A\v;

U = reshape([u;1], 3, 3)';
w = U*[x'; y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
disp(w);
```

The w of w is given by:

```
w =

    0.0000    210.0000    210.0000     0.0000

         0     0.0000    297.0000    297.0000

    1.0000     1.0000     1.0000     1.0000
```

The value of w is the same as desired.

#### 2.6.d. Wrap the image and Display it

By running the functions below, we can wrap up the image and display it:

```
% wrap up in image and diaplay it
T = maketform('projective', U);
P2 = imtransform(P, T, 'XData', [0 210], 'YData', [0 297]);
figure;
imshow(P2);
```

The result image is:



We can observe that the upper part of the image is more blurred than the lower part of the image. This may be caused by in the original image, the upper part occupies smaller space than the lower part and must be stretched to match the target image size, thus causing the blur.