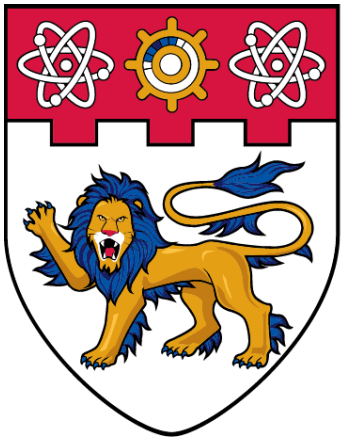


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Lab Report 1 : Image Processing in MATLAB

(Point Processing, Spatial Filtering, Frequency Filtering & Imaging Geometry)

CZ4003 – COMPUTER VISION
(Semester 1, AY 2020/2021)

WILSON THURMAN TENG
U1820540H

1. Introduction

1.1 Overview

In this report, we to explore and implement different algorithms used for image processing in MATLAB. Although various algorithms discussed in this report have already been implemented by the IPT (Image Processing Toolbox) software package, we adopt a first principles approach (unless otherwise specified) and use IPT to check if the algorithm has been correctly implemented.

1.2 Dependencies

As mentioned above in the overview, the following tools will be used:

- MATLAB R2020a (Source code tested on this version)
- Image Processing Toolbox software package

2. Experiments

2.1 Contrast Stretching

(a, b) Display original and grayscale image

The whos command is useful for finding out the image size as well as the pixel datatype.

```
 Name        Size                Bytes  Class     Attributes

 P1a       320x443             141760  uint8
```

Original RGB image is black and white, like the grayscale image. This suggests that every pixel in the RGB image has the same value for all 3 channels (R, G & B). This can be confirmed with the following assertion statements.

```
%% (a) Load image and transform to grayscale
Pc = imread('images/mrt-train.jpg');
whos Pc;
P1a = rgb2gray(Pc);
assert(max(max(Pc(:,:,1) - P1a(:,:))) == 0);
assert(max(max(Pc(:,:,2) - P1a(:,:))) == 0);
assert(max(max(Pc(:,:,3) - P1a(:,:))) == 0);
```

(c) Find Min & Max intensities of grayscale image

Source Code:

```
%% (c) Find Min & Max intensities of grayscale image
min_intensity = min(P1a(:)) % Used as offset
max_intensity = max(P1a(:))
```

Output:

```
min_intensity =     max_intensity =

  uint8               uint8

   13                  204
```

The min and max intensities do not correspond the range of values possible from using the uint8 datatype (0-255). Hence, we can perform contrast stretching to attempt to improve the contrast of the image.

(d) Contrast Stretching

Contrast stretching, or normalization is a simple image enhancement technique used to 'stretch' the range of intensity values present in the image to the range that the pixel data type allows.

```
%% (d) Contrast Stretching

% Native implementation
norm_factor = 255 / (double(max_intensity) - double(min_intensity));
normalised_img = (double(P1a) - double(min_intensity)) * norm_factor;
P1d_ = uint8(normalised_img);

% Using Image Processing Toolbox Library
P1d(:,:) = imsubtract(P1a(:,:), double(min_intensity));
P1d(:,:) = immultiply(P1d(:,:), ...
    255 / (double(max_intensity) - double(min_intensity)));

% Check if P1d and P1d_ has gone through contrast stretching
assert(min(P1d(:)) == 0 && max(P1d(:)) == 255)
assert(min(P1d_(:)) == 0 && max(P1d_(:)) == 255)
% Check if first principles implmentation is correct
assert(max(max(P1d(:,:) - P1d_(:,:))) == 0);
disp("Assertion passed : P1d max and min values are 0 and 255 respectively.");
```

To perform normalization, we must first subtract the minimum value (13) from the entire image matrix before multiplying the resultant image matrix by a factor – new range over initial range (255 / 204-13).

Additionally, assertion statements have been added to check if the min and max values of the resultant image is 0 and 255 respectively. Since the script did not throw any errors, the assertion is true.

(e) Display result and automatic contrast stretching for display

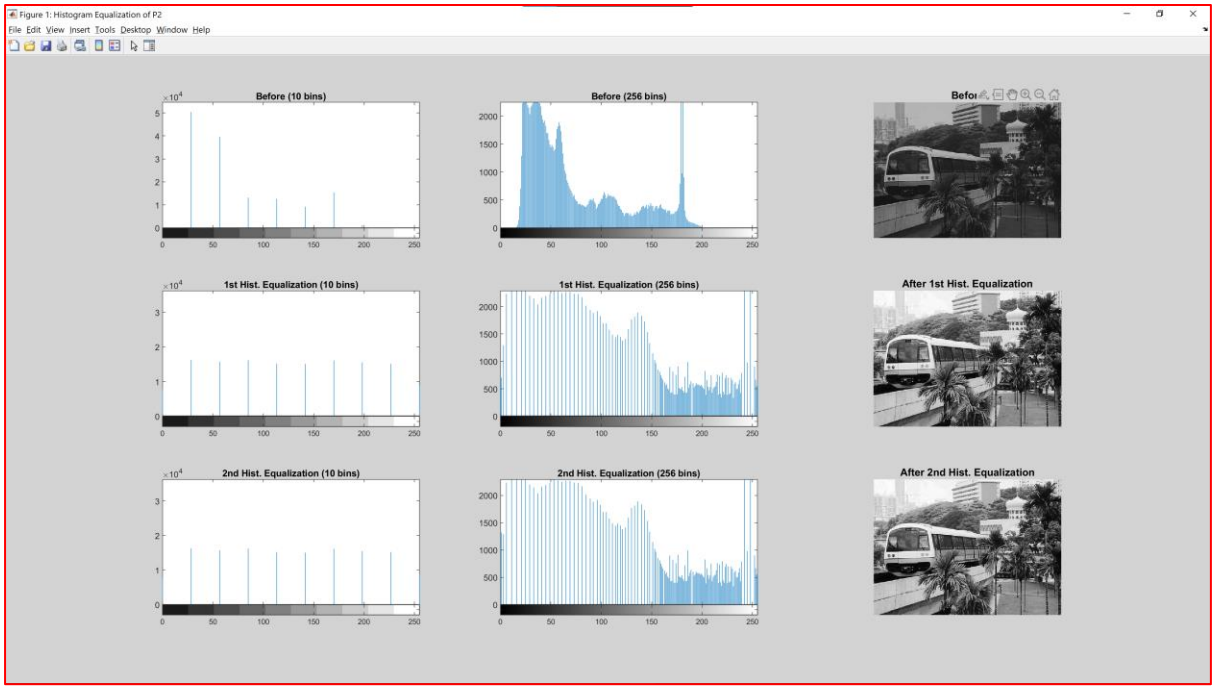
The image on the left shows the original image while the image on the right shows the image after contrast stretching has been performed. We can see that the resultant image displays a greater range of colours as compared to the original image.

Next, we do a comparison between imshow(img, []) and imshow(img).

imshow(img, []) is performed on P1a, the original image and imshow(img) is performed on P1d, the resultant image. Since the 2 images do not have any noticeable differences in contrast, we can conclude that imshow(img, []) performs automatic contrast stretching to display the image. It is important to note, however, that it does not change the underlying matrix.

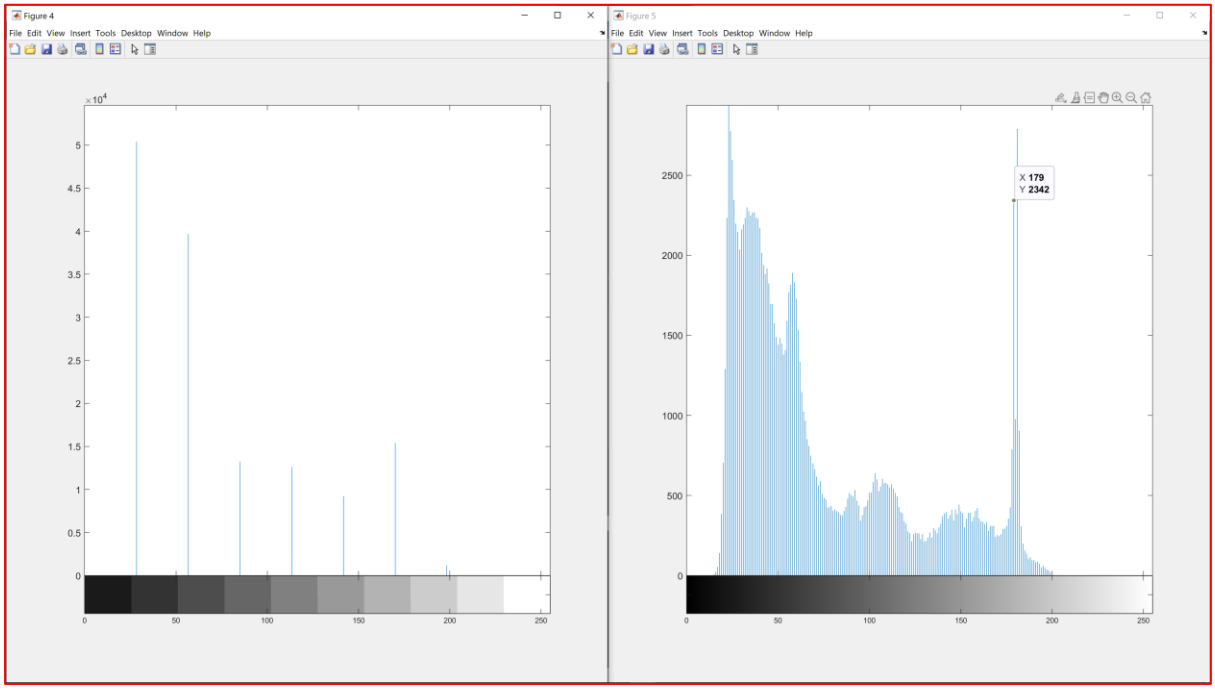
2.2 Histogram Equalization

The rows in the diagram below represent the image before histogram equalization (HE), after performing HE once, and after performing HE twice, in that order. The columns represent the image histogram with 10 bins, 256 bins and finally the image.




```
%% 2.2 Histogram Equalization

%% (a) Show histogram of P
Pc = imread('images/mrt-train.jpg');
P2 = rgb2gray(Pc);
figure('Name', 'Histogram Equalization of P2', 'Color', '#D3D3D3');

subplot(3,3,1), imhist(P2, 10), title('Before (10 bins)');
subplot(3,3,2), imhist(P2, 256), title('Before (256 bins)');
subplot(3,3,3), imshow(P2), title('Before');

%% (b) Histogram Equalization on P
P2b = histeq(P2, 256);
subplot(3,3,4), imhist(P2b, 10), title('1st Hist. Equalization (10 bins)');
subplot(3,3,5), imhist(P2b, 256), title('1st Hist. Equalization (256 bins)');
subplot(3,3,6), imshow(P2b), title('After 1st Hist. Equalization');
%

%% (c) Repeat Histogram Equalization on P
P2c = histeq(P2b, 256);
subplot(3,3,7), imhist(P2c, 10), title('2nd Hist. Equalization (10 bins)');
subplot(3,3,8), imhist(P2c, 256), title('2nd Hist. Equalization (256 bins)');
subplot(3,3,9), imshow(P2c), title('After 2nd Hist. Equalization');

diff = imsubtract(P2b(:,:), P2c(:,:));
figure('Name', 'P2b subtract P2c', 'Color', '#D3D3D3'), imshow(diff), title('Pixels are all 0, indicating P2b == P2c');
% figure('Name', 'Ignore', 'Color', '#D3D3D3'), imshow(diff), title('Ignore');

assert(max(diff(:)) == 0) % Check if P2 has gone through contrast stretching
disp("Assertion passed : P2b == P2c");

% Same Histogram even after repeated Histogram Equalization.
```

(a) Difference between 10 bins and 256 bins in a histogram

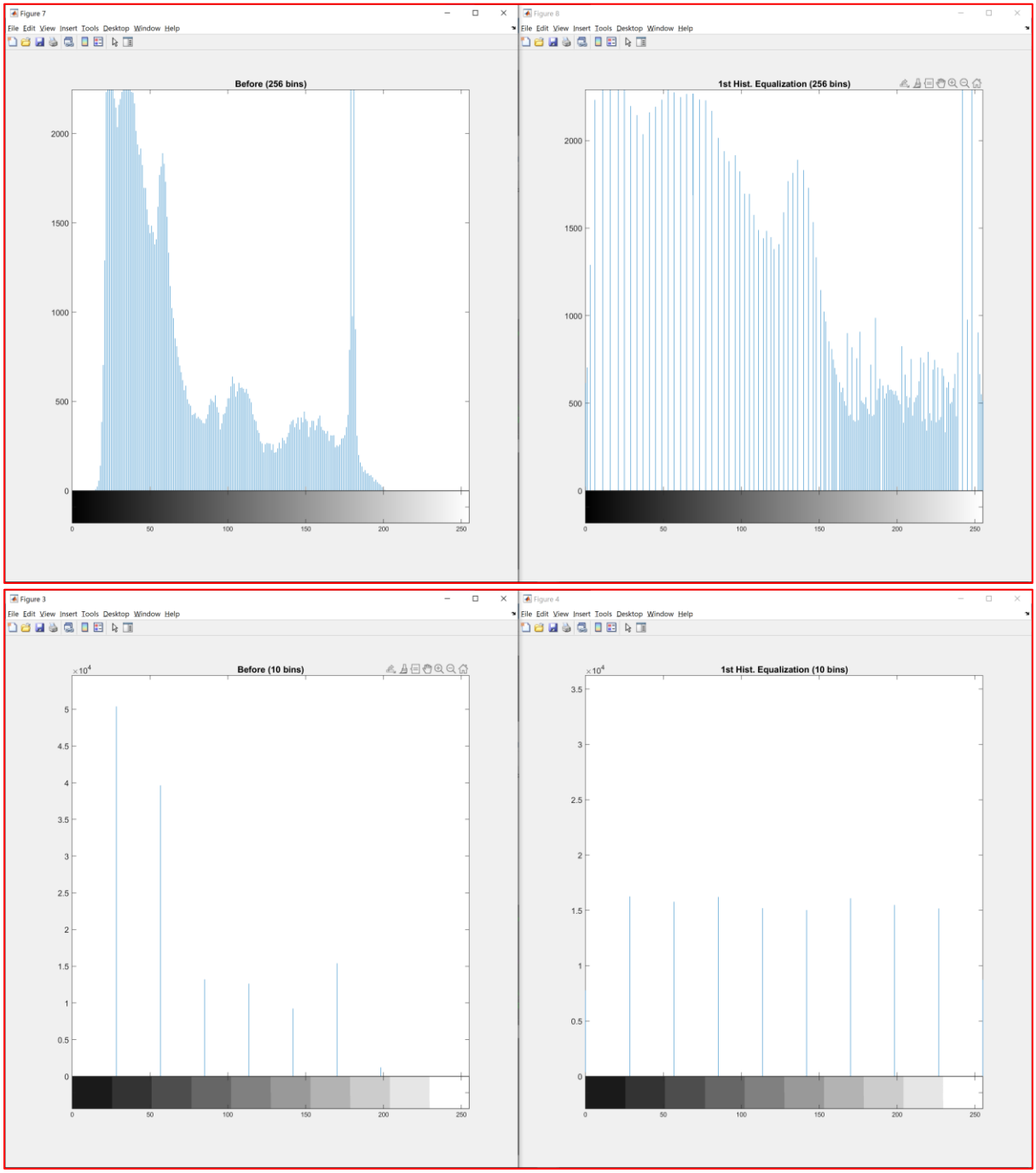


In a histogram, the x-axis represents the pixel intensity while the y-axis represents the pixel count. Regardless of the number of bins, the total number of 141760 pixels (320x443=141760) will be distributed to all the bins.

A 10-bin histogram shows an aggregated form of the 256-bin histogram. On average, every bin in the 10-bin histogram aggregates the pixel count of 25.6 (256/10) pixel intensities. While the aggregated form of the 10-bin histogram allows us to see the general trend much quicker, some details are lost as a trade-off. An example would be the sharp spike in pixel count for pixel intensities between 54-62 as well as the sudden increase in pixel count for pixel intensities 179 and 181.

To conclude, a 10-bin histogram is suitable to assess the general trend of the histogram quickly while a 256-bin histogram is suitable for more in-depth analysis of the image.

(b) Similarities and differences after performing HE



Similarities:

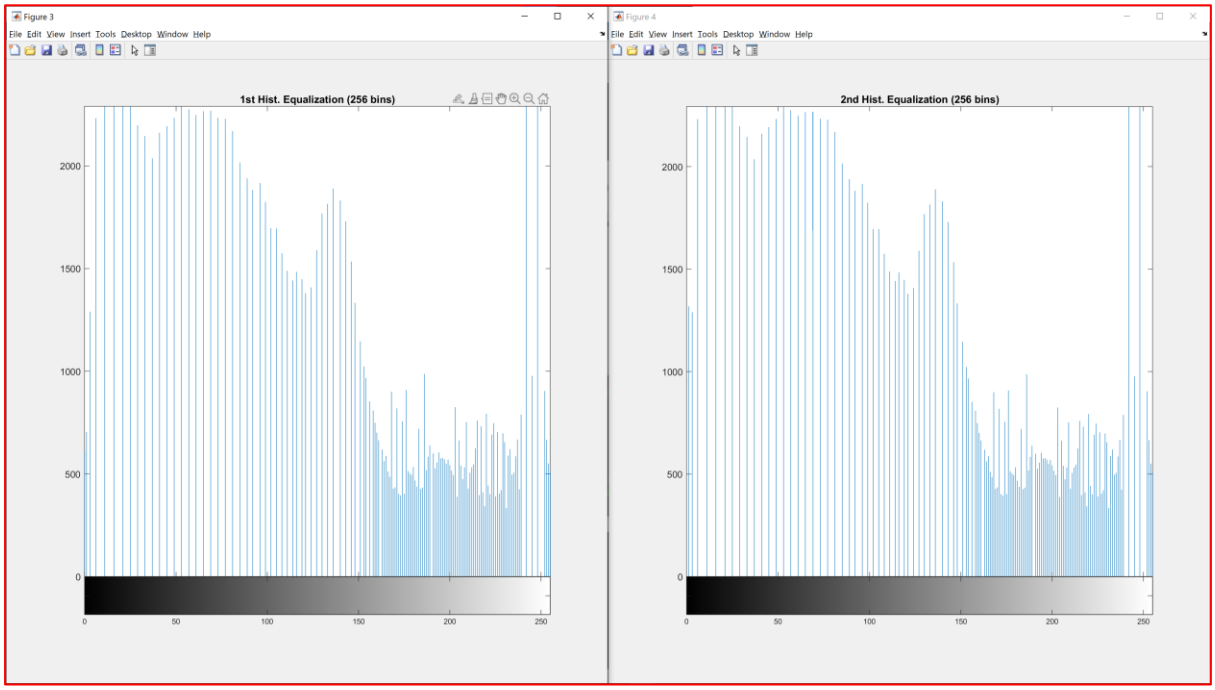
- The total pixel count in both histograms are still the same.
- The range of possible pixel intensities constrained by the uint8 datatype are the same. (0-255)

Differences:

- By analyzing the 10-bin histogram of both images, it is observed that the distribution of the pixel intensities is different. The image before has a higher distribution of pixels in the lower range (<50) while the image after HE has an almost uniform distribution of pixel intensities.

- By analyzing the 256-bin histogram of both images, it is observed that both images have different pixel intensity ranges. The image before has a shorter range (204-13=191) while the image after HE has a larger range.

(c) Comments on image and histogram after performing 2nd HE

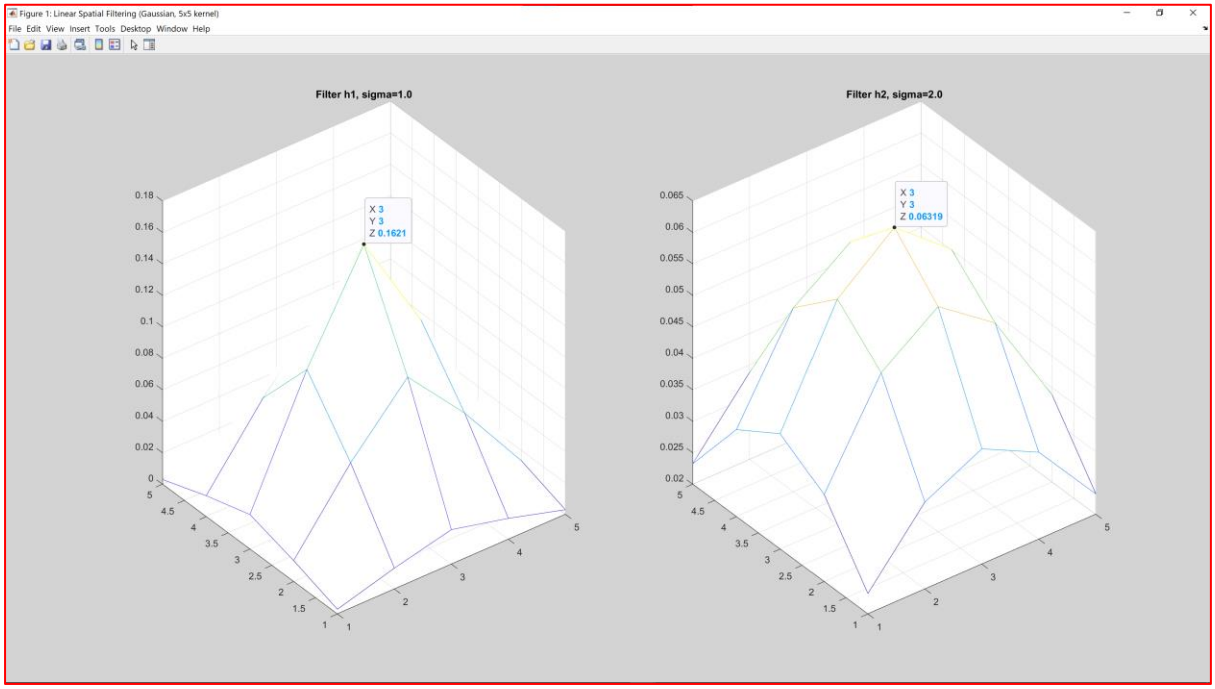


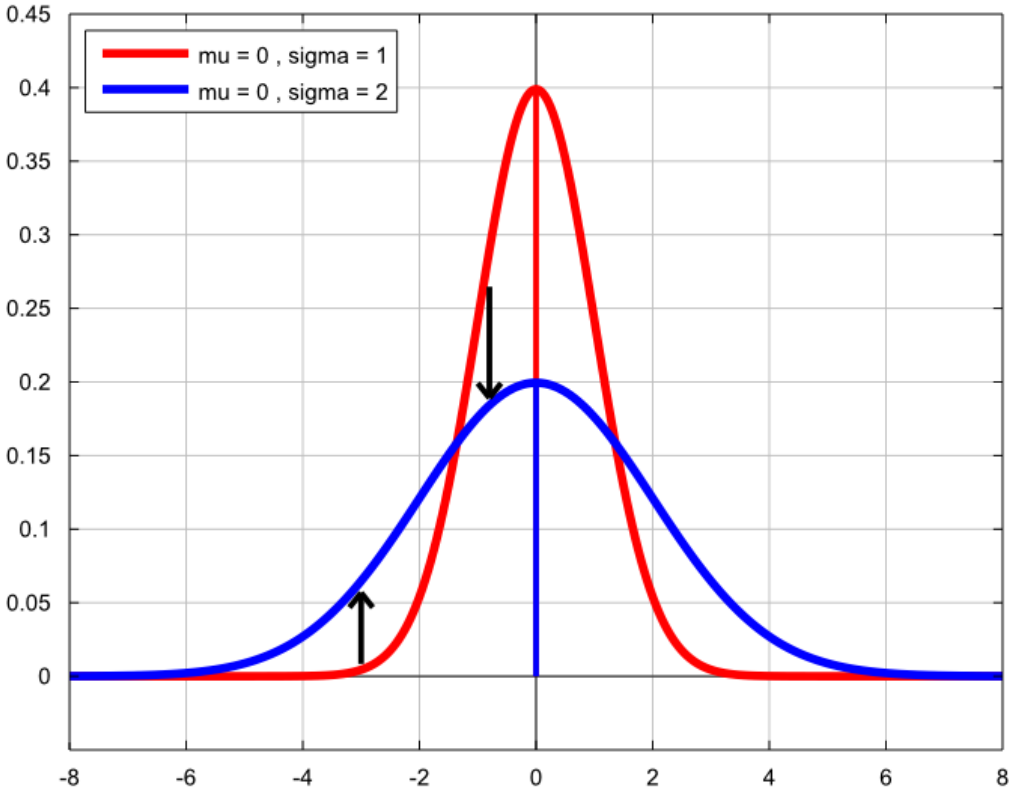
The assertion statement at the end is takes the image after the 1st and 2nd HE and computes the difference. If the maximum value of the difference matrix is 0, we can conclude that the 2 matrices are the same. Since the script executes without any errors, we can conclude that the image is the same even after performing HE the second time.

This shows that the histogram equalization algorithm is idempotent. This is expected as the range of pixel intensity range (0-255) and number of pixels remain the same for both passes of the HE algorithm. Hence, the same equilibrium will be reached.

2.3 Linear Spatial Filtering

(a, b) 5x5 Gaussian PSF (Point Spread Function) with different sigma values





Normal Distribution (https://www.statlect.com/probability-distributions/normal-distribution)

5x5 represents the size of the kernel while sigma represents the standard deviation of the Gaussian distribution imposed on the kernel.

```
% Functions to generate Gaussian PSF
sqrt_dist = @(x,y) ...
    x.^2 + y.^2;
h = @(x, y, sigma) ...
    (1 / (2 * pi * sigma^2)) * (exp(-1*sqrt_dist(x,y) / (2 * sigma^2)));
normalize_h = @(h) ...
    h ./ sum(h(:));
```

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

To obtain the gaussian kernel, we first calculate the L2 norm (Euclidean distance) of the kernel position to the centre of the kernel. After which, we calculate the values using the formula above. Lastly, the kernel undergoes normalization to ensure that the weights given to each kernel position is equivalent.

As shown in the normal-distribution curve, when the sigma is increased from 1 to 2, the density is shifted from centre to the tails of the curve. As such, the mesh plot generated is expected as the mesh plot's peak decreases from 0.1621 to 0.06319 and has a flatter distribution when sigma=2.0.

```
x = -2:2;
y = -2:2;
sigma_a1 = 1.0; % Standard Deviation of distribution.

[X,Y] = meshgrid(x,y);
h1 = h(X, Y, sigma_a1);
h1_norm = normalize_h(h1);
assert(max(max(h1_norm - fspecial('gaussian', 5, 1)))<10^-10)
disp("Assertion passed : h1_norm is correct.");
```

To test if the implementation was correct, an assertion statement was used to compare the output of the in-built fspecial('gaussian', kernel_size, sigma) function from the PTI library with our implemented kernel. As expected, both kernels are almost similar, with differences of less than 10^-10 resulting from floating point calculation differences.

(b, c) Gaussian filter on image with additive Gaussian Noise

(d, e) Gaussian filter on image with additive Speckle Noise

(e) Conclusions

The Gaussian filter is suitable for filtering out Gaussian noise as all the pixels in the kernel are giving weights. This is especially useful for removing/reducing noise when the noise is spread across several pixels. Consequently, this causes the gaussian filter to not be as effective for removing speckle noise as speckles are considered outlier data.

In addition, this noise removal is performed at the cost of a blurrier image and lost edges (E.g. tree in the bottom centre of image has lost its definition). (Higher sigma will generate a blurrier image). Both filters also reduce the contrast of the image at sigma=2.0 which is not ideal although we can apply contrast stretching to combat this.

2.4 Median Filtering

(b, c) Median filter on image with additive Gaussian Noise

(d, e) Median filter on image with additive Speckle Noise

(e) Conclusion

The Median filter is very effective at removing speckle noise as it is insensitive to outliers. We can see that a 3x3 Median filter performs better than a 5x5 Gaussian filter at removing speckle noise. This is so as the median filter only selects the median value of the kernel and ignores the outlier speckle noise.

2.4.5 Conclusion on Gaussian filter vs Median filter

Although both Gaussian and Median filters have their own characteristics and strengths, median filters will work better generally as the Gaussian filter tends to blur the image due to gaussian averaging and hence, information is lost at the expense of noise removal. Median filters, on the other hand, preserve edges and hence would be much useful to preserve information.

2.5 Suppressing Noise Interference Patterns

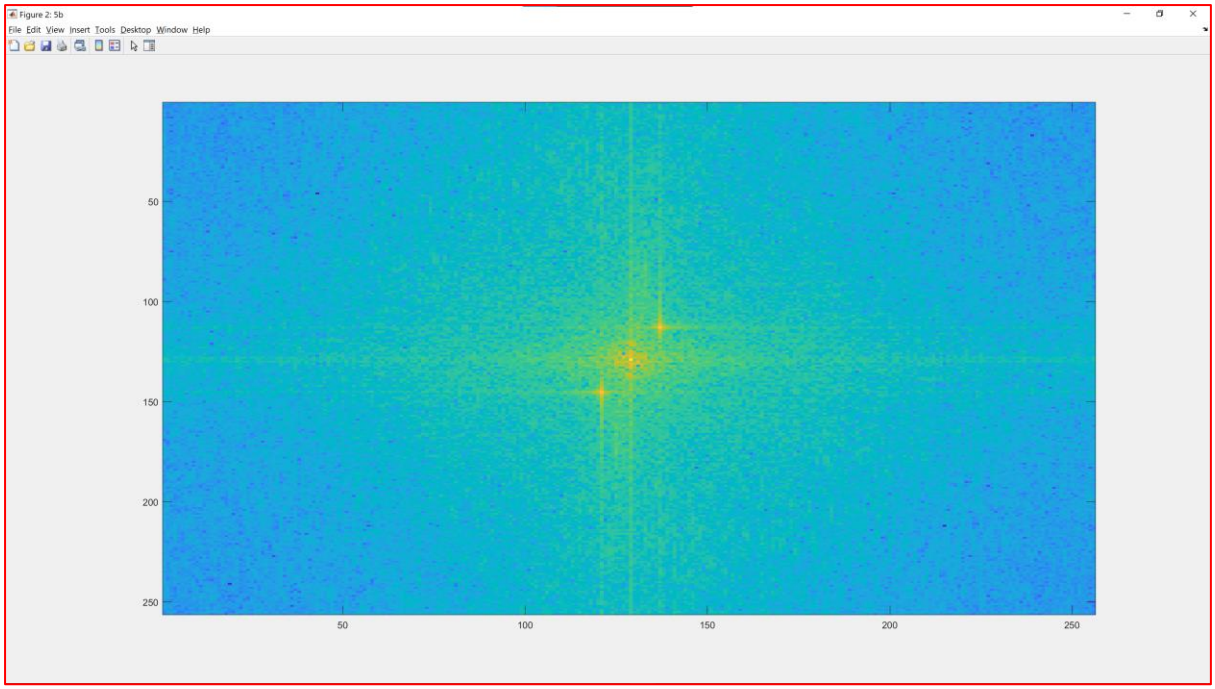
(a) Display pck-int.jpg, an image with dominant diagonal lines

The image seems to be affected by artificially added periodic diagonal noise.

(b) Obtain Fourier Transform F of image and FFT shift

```
%% (b) Obtain Fourier Transform F of image and FFT shift
F = fft2(P); % complex matrix
S = abs(F); % real matrix
figure('Name', '5b'), imagesc(fftshift(log10(S)));
colormap('default');
```

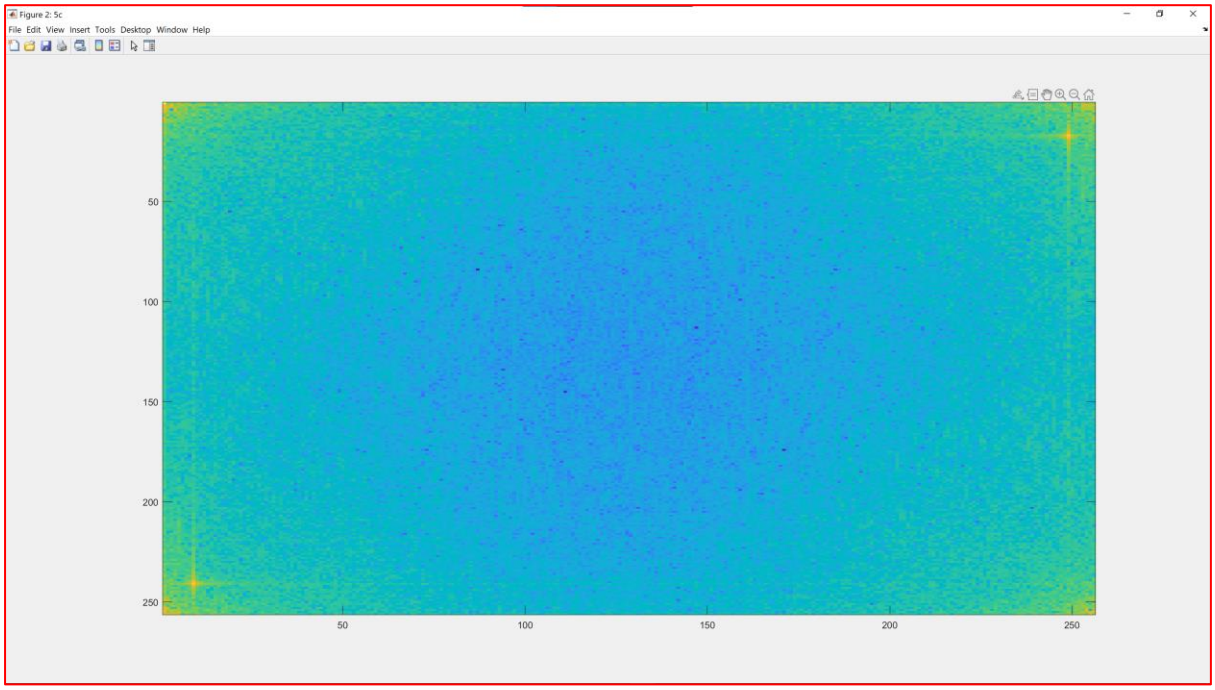
We use log10 instead of root 10 to non-linearly scale our power spectrum and allow more effective visualization.

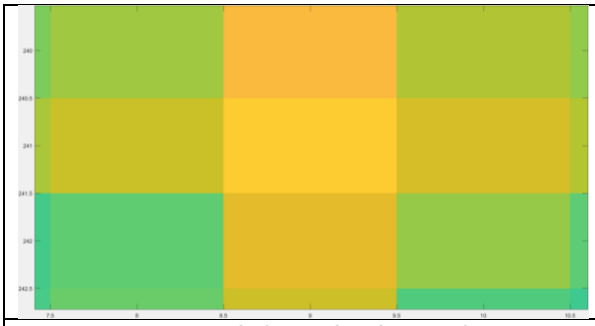
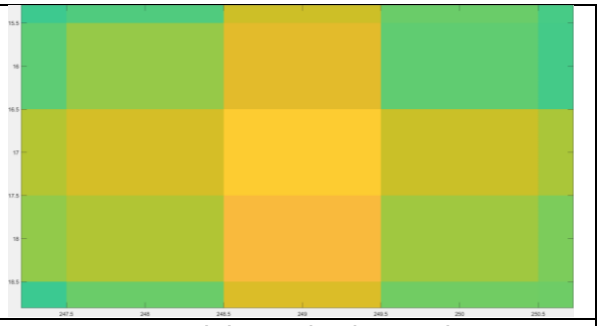


FFT shift is used to shift the origin, which corresponds to the DC (0 frequency) component of the Fourier transform to the centre. The values near the border and the centre of the image represent high frequencies.

From this diagram, 2 symmetrical peaks isolated from the central mass can be observed at relatively low frequency. These two peaks correspond with the periodic pattern observed in the original image.

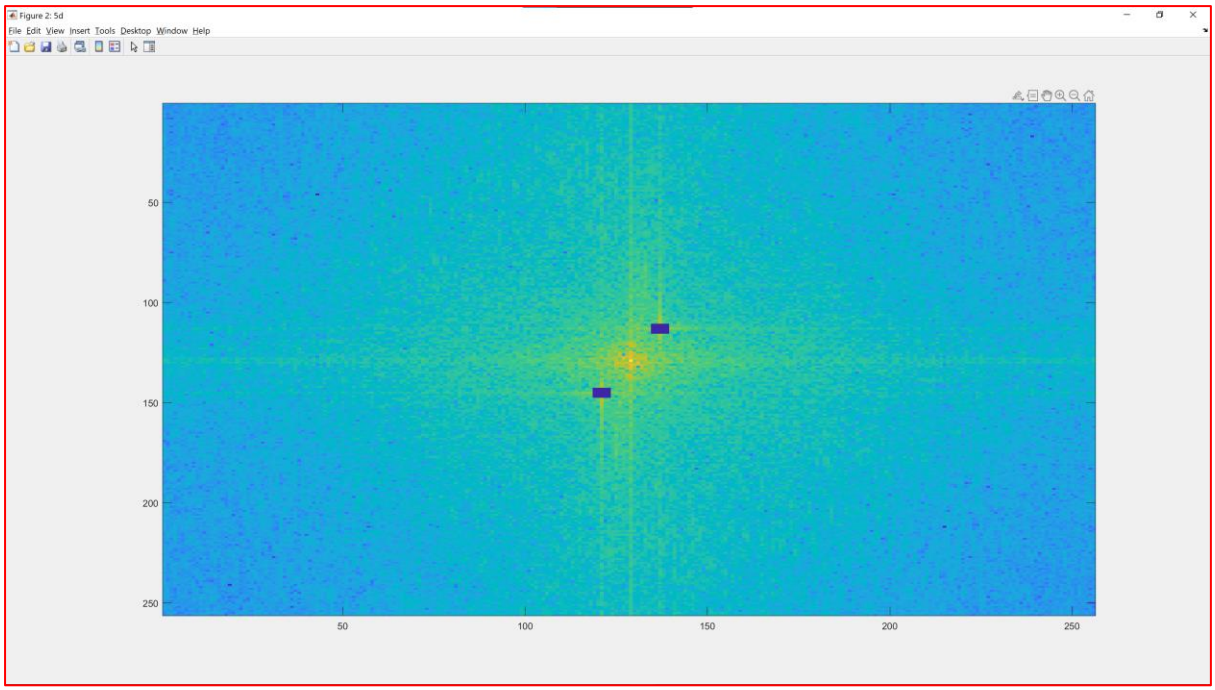
(c) Measure location of peaks on S, w/o FFT shift



	
Bottom-left peak : (241, 9)	Top-right peak : (17,249)

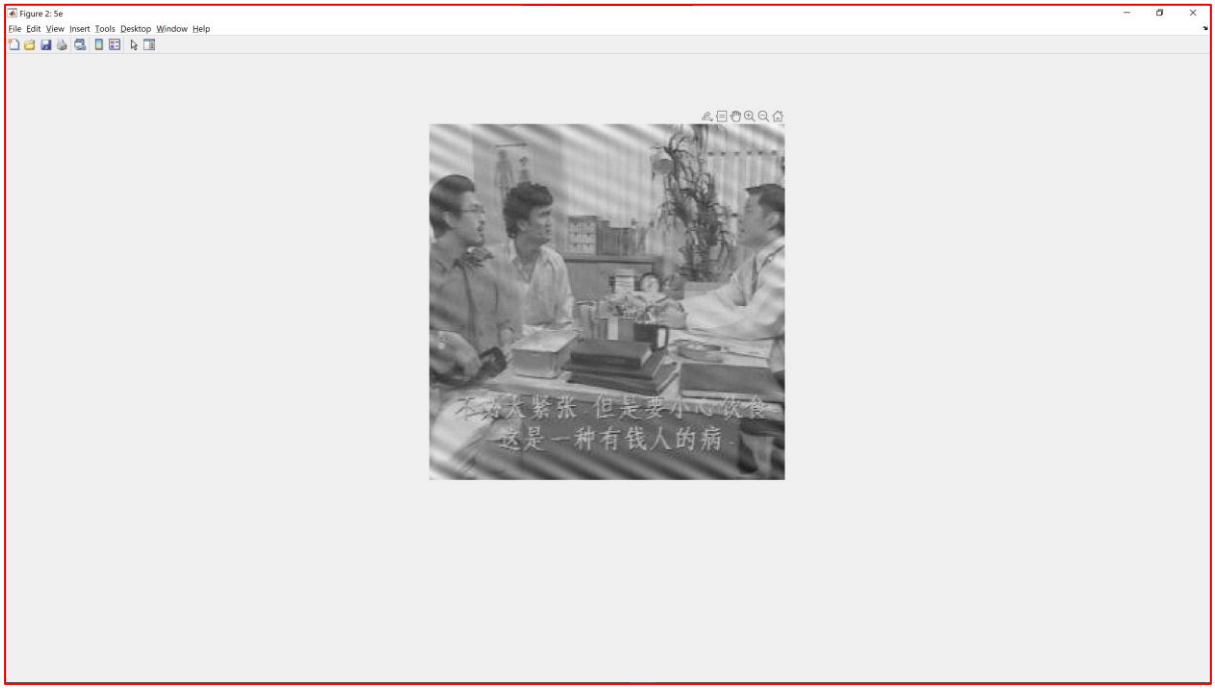
By zooming into the image, we obtain the centre coordinate of both peaks – (241, 9) and (17,249).

(d) Remove peaks from F



The peaks are removed by replacing the peak centres and the surrounding neighbours (5x5) with 0.

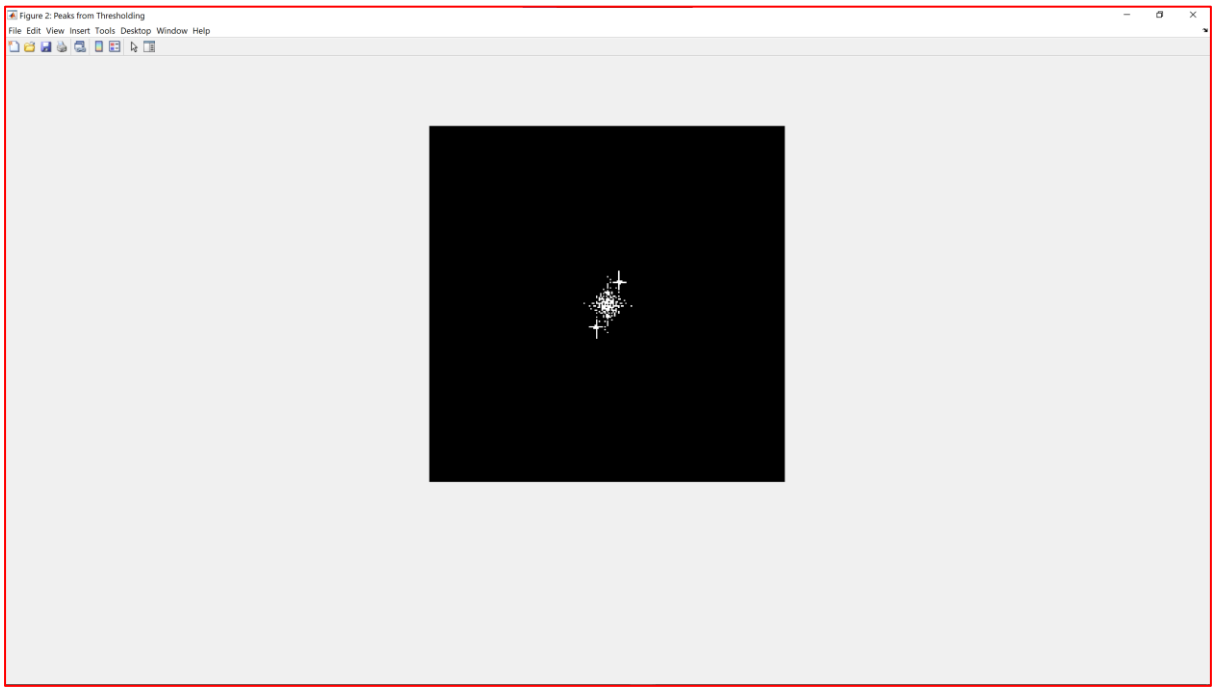
(e)(i) Inverse Fourier Transform on F and display result



The result has some of the signals removed by some still of the diagonal signals remain, especially near the borders of the image. A way to improve this would be to use a band-reject filter that we can use to filter out the specific frequencies that are causing the interference.

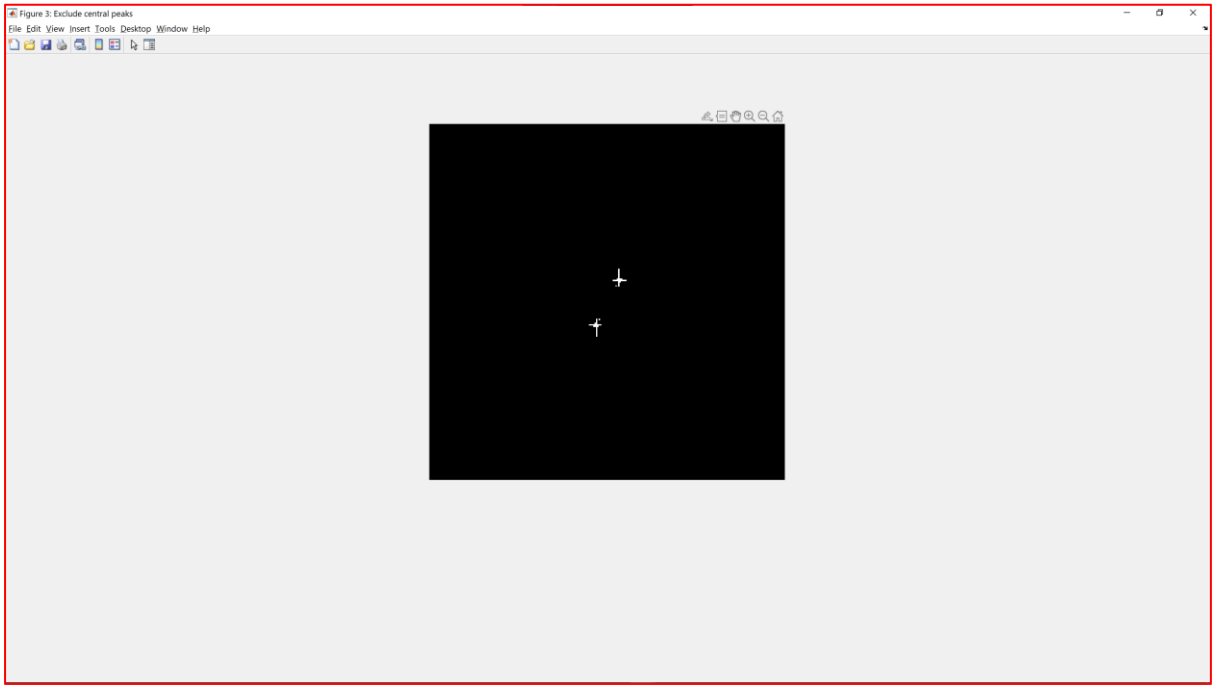
(e)(ii) Improve upon results from (e)(i)

Part A



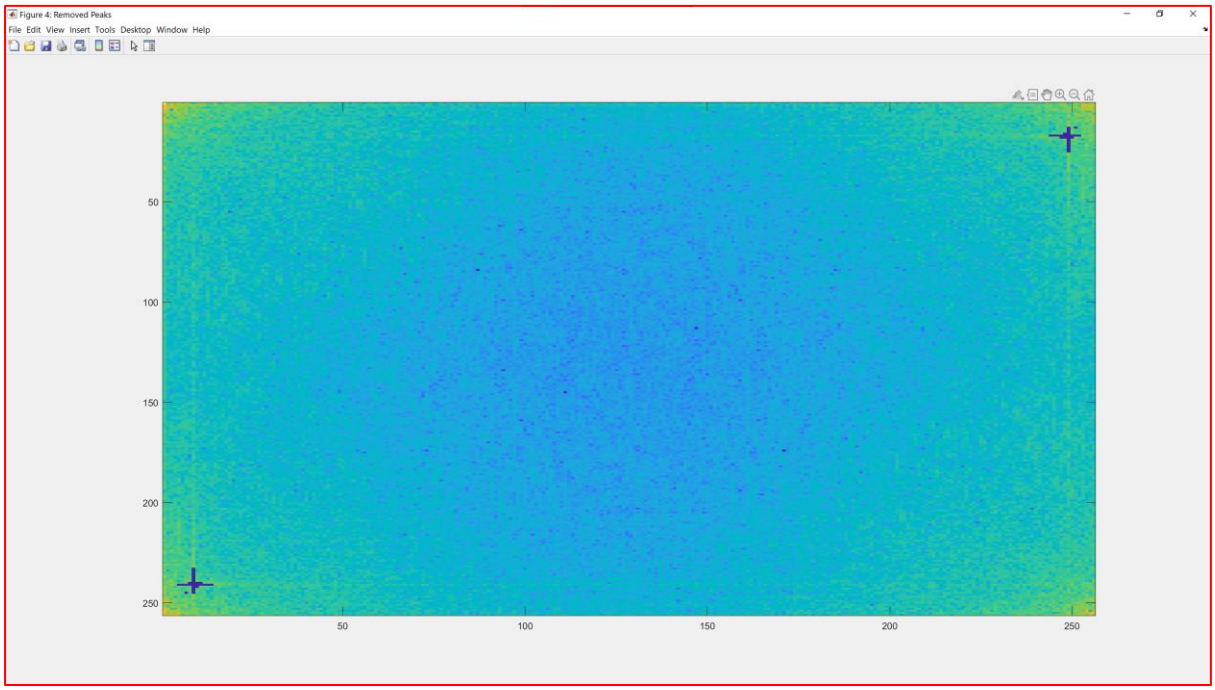
We threshold the shifted power spectrum to obtain this binary image.

Part B



We exclude the DC frequency components from the threshold as these low frequencies correspond to the information in the image.

Part C



We replace the threshold frequencies with 0 and obtain this power spectrum.

Part D

The resultant image has more of the diagonal signal removed but has interlaced dark pixels. To remedy this, we apply Gaussian filter to obtain the final image with red borders in Part E.

Part E

The image with the blue border shows the result from (e)(i) while the image with the red border shows the result from (e)(ii). While the image obtained from (e)(ii) has more of its diagonal signal removed, it is blurrier from the usage of the gaussian filter to average the interlaced image obtained from part D.

(f) "Free" the Primate !

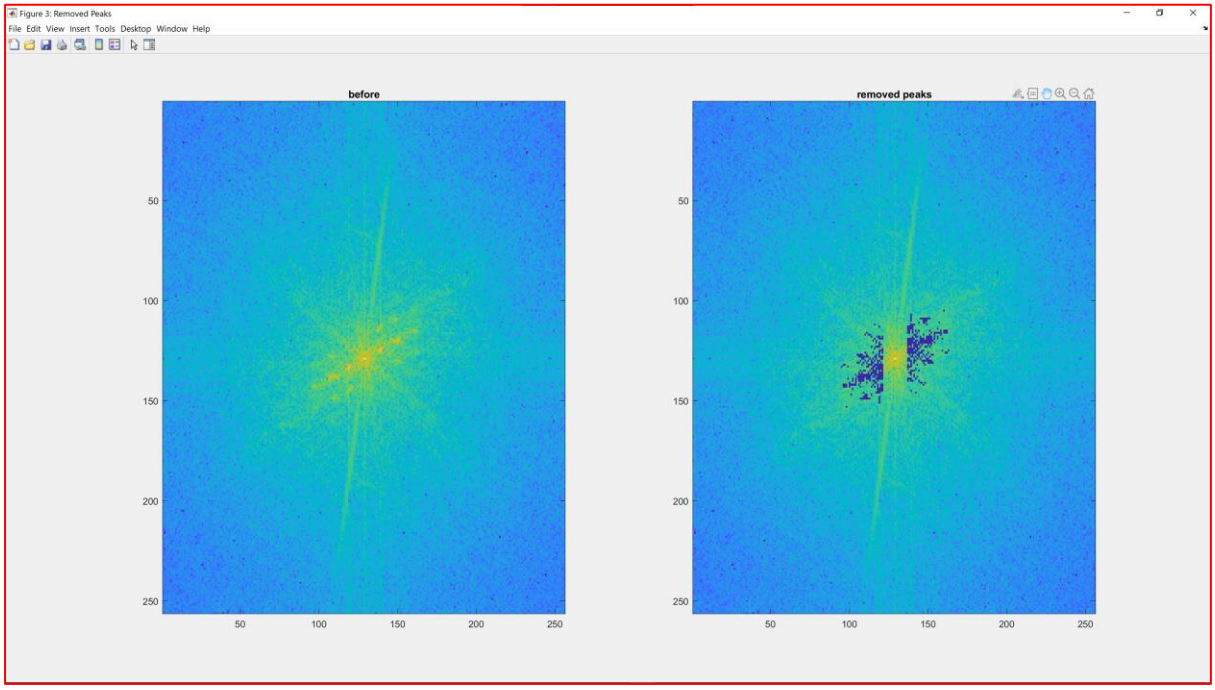
Part A

We first use Canny edge detection to detect the edges. After which, we remove edges of objects which enclose less than 110 pixels (8-connected). We then calculate the average pixel value of the non-fence parts of the image (95.8605) to be used in Part D.

Part B

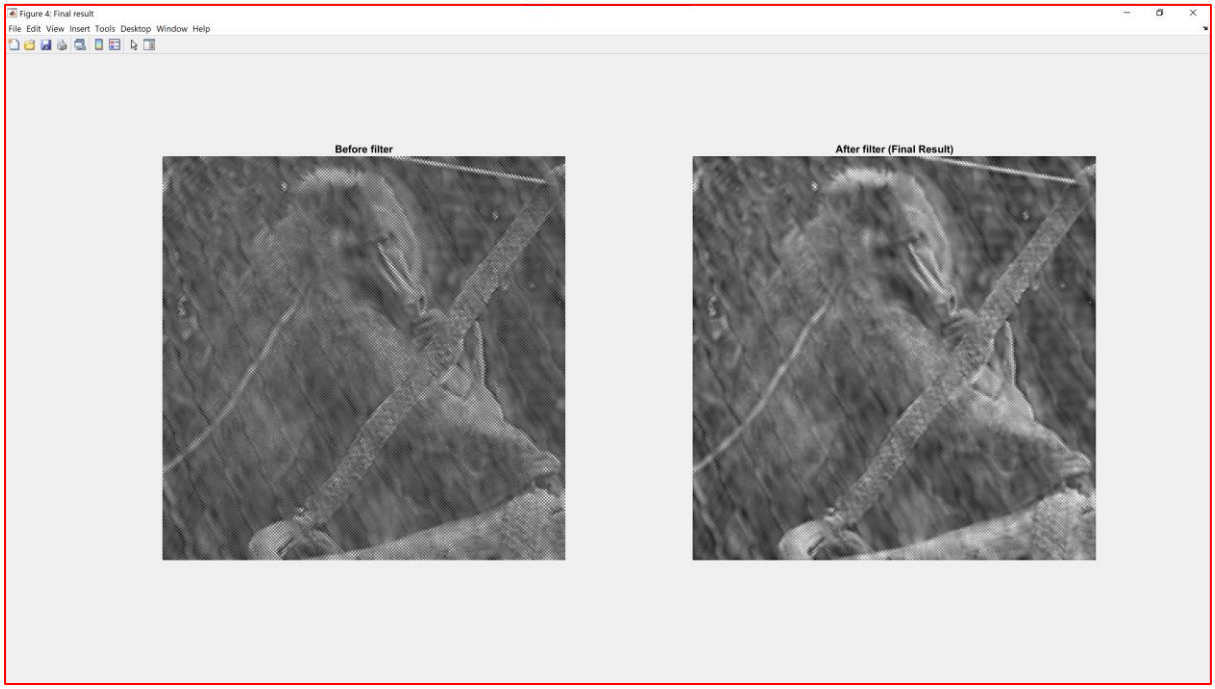
We threshold the power spectrum of the original image to obtain the image on the left. We then exclude the DC frequency spectrum to obtain the image on the right.

Part C



This diagram shows the FFT shift of the power spectrum before and after removing the threshold-ed frequencies.

Part D



After removing the threshold frequencies, we obtain an interlaced image, we replace the dark pixels with the average non-fence pixel values we obtain in Part A and obtain the image on the left. We then apply 3x3 Gaussian filter with a low value of 0.5 standard deviation to minimize the blurry effect of gaussian filtering.

Part E

This diagram shows a comparison of the original image with our obtained result. The fence has been mostly removed and the details of the non-fenced parts of the primate have mostly remained – we can still observe the primate's fingers and fur patterns on the its back.

This result can be attributed to the use of the canny edge detector to detect the fence and obtain the average pixel values of the non-fenced parts of the image to improve the gaussian filtering of in Part D.

2.6 Undoing Perspective Distortion of Planar Surface

(a) Display book.jpg

(b) Obtain the corner of the book in the image, as well as the desired 4 corners

```
%% (b)Obtain the corner of the book in the image, as well as the desired 4 corners
% [X, Y] = ginput(4);
x1 = 142; y1 = 28; % Top-left
x2 = 308; y2 = 48; % Top-right
x3 = 3; y3 = 159; % Btm-Left
x4 = 257; y4 = 216; % Btm-right


Xw = [x1; x2; x3; x4]; % x-coord (Top-left; Top-right; Btm-Left, Btm-right)
Yw = [y1; y2; y3; y4]; % y-coord (Top-left; Top-right; Btm-Left, Btm-right)

Xim = [0; 210; 0; 210]; % x-coord (Top-left; Top-right; Btm-Left, Btm-right)
Yim = [0; 0; 297; 297]; % y-coord (Top-left; Top-right; Btm-Left, Btm-right)
```

Description	Coord in book.jpg	Coord in desired image
Top-left	(142,28)	(0,0)
Top-right	(308,48)	(210,0)
Bottom-left	(3,159)	(0,297)
Bottom-right	(257,216)	(210,297)

The 4 corners from book.jpg was obtained from zooming into the image and manually recording the coordinates, like how the peaks were obtained in 2.5(c).

(c) Setup matrix to estimate the projective transformation

```
%% (c) Setup matrix to estimate the projective transformation
% Generate matrix A
A = zeros(8,8);
for i = 1:4
    A(i*2-1, :) = [Xw(i); Yw(i); 1; 0; 0; 0; (-1 * Xim(i) * Xw(i)); (-1 *  Xim(i) * Yw(i))];
    A(i*2, :) = [0; 0; 0; Xw(i); Yw(i); 1; (-1 * Yim(i) * Xw(i)); (-1 *  Yim(i) * Yw(i))];
end

v = [Xim(1); Yim(1); Xim(2); Yim(2); Xim(3); Yim(3); Xim(4); Yim(4)];
u = A \ v;
U = reshape([u;1], 3, 3)';

% Verify U is correct by transforming into original coordinates.
w = U*[Xw'; Yw'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:))
```

```
w =

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

U is correct as W corresponds to the coordinates we have designated for our desired image (vectors Xim and Yim).

(d) Warp image

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

The image P2 is obtained by transforming P.

(e) Display and comment on results

Dimensions	Total number of pixels
240x320	76800
297x210	62370

This result from the affine transformation is as expected. Due to the originally poor resolution (76800 pixels), and the requirement to produce an image with roughly 82.1% (62370/76800*100%) of the original pixel count, the resultant image is expected to be blurry. However, there are some parts of the image that are clearer as compared to the others. The lower half of the book is the sharpest while the upper half of the book is distorted. This could be due to the angle at which the picture is taken which gives limited information. Hence, the upper half could not accurately undo this angle without having a higher resolution image.

Overall, the quality of the result is good considering the sharp angle the picture is taken at.