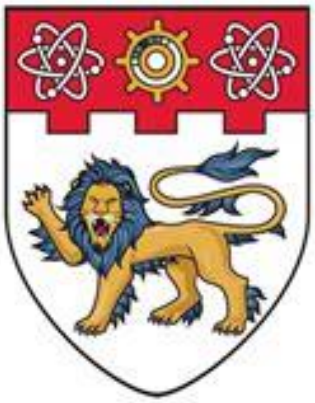


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4003: Computer Vision

Lab 1 Report

Ingale Omkar (U2020724H)

Table of Contents

2.1 Contrast Stretching

2.1 (a) Inputting image ‘mrt-train.jpg’

The input of the picture was done using the imread command. The code was as follows:

```
Pc = imread(['/Users/omkaringale/Desktop/Computer Vision/Assignments' ...
    '/images/mrt-train.jpg']);
whos Pc;
P = rgb2gray(Pc);
whos P;
```

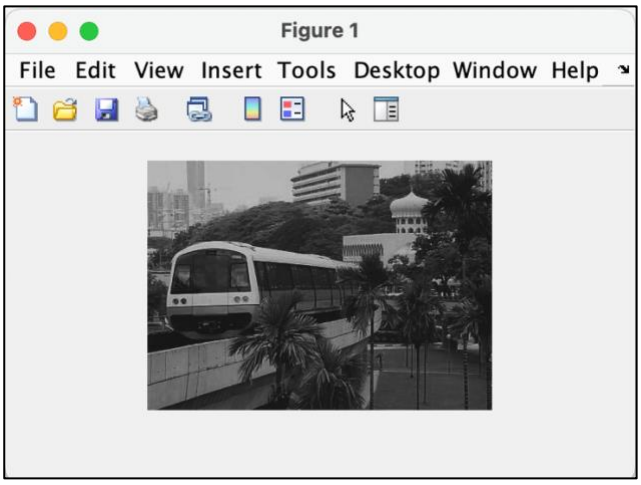
The results of the whos command for P and Pc was as follows:

```
>> whos Pc
  Name        Size                Bytes  Class    Attributes

  Pc        320x443x3            425280  uint8

>> whos P
  Name        Size                Bytes  Class    Attributes

  P         320x443              141760  uint8
```

Observation: There were 3 dimensions in the original image which was reduced to 2 dimensions after the application of the ‘rgb2gray’ command.

2.1 (b) Viewing grayscale image

The image was viewed using the imshow command.



2.1 (c) Minimum and maximum intensities of image

The minimum and maximum intensities of the image are 13 and 204 respectively.

```
ans =

  uint8

    13
```

```
ans =

  uint8

   204
```

2.1 (d) Contrast Stretching

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. In this case, the lowest intensity is 13 and the largest intensity is 204. Contrast stretching was done using the following code:

```
P2(:, :) = imsubtract(P, 13); % r - r_min
P2(:, :) = immultiply(P2, 255/(204-13)); % 255/(r_max-r_min)

min(P2(:)), max(P2(:))
```

The smallest and largest intensities of the image after contrast stretching were as follows:

```
ans =

  uint8

    0
```

```
ans =

  uint8

   255
```

2.1 (e) Image after contrast stretching

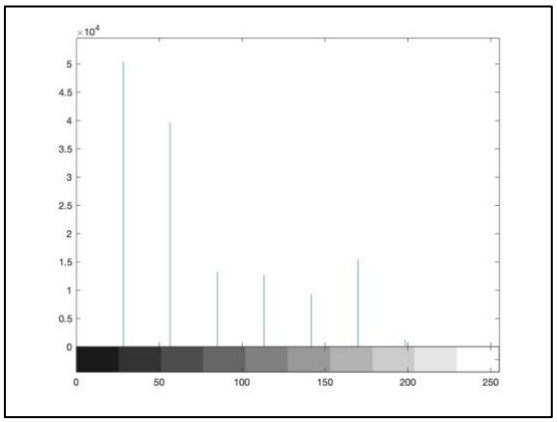
Before

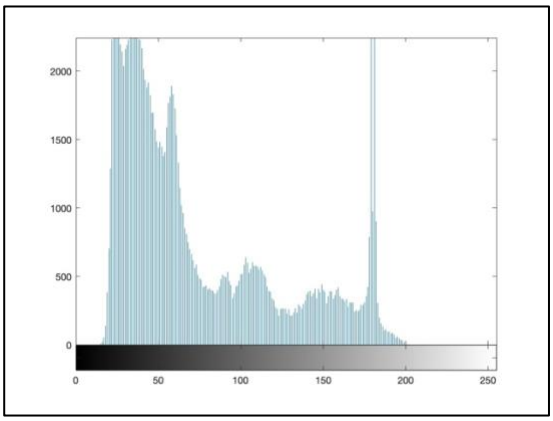
After

The contrast in the new image after contrast stretching seems to be better. This implies that contrast stretching was done correctly.

2.2 Histogram Equalization

2.2 (a) Image intensity histogram





10 bin 256 bin

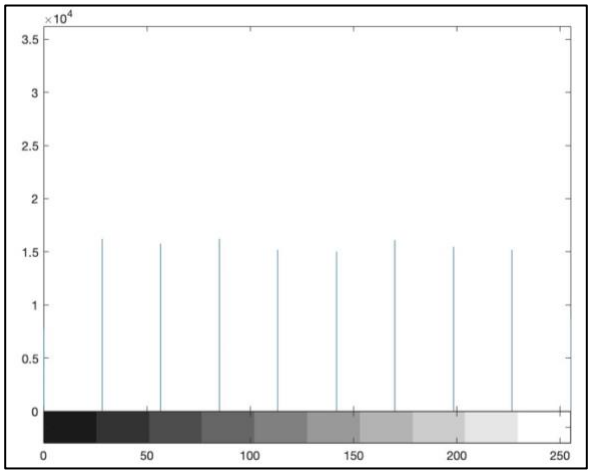
The main difference that is visible in the two histograms is the number of bins used. The first histogram has 10 bins and the second has 256 bins. Due to this, there are more pixels in each bin in the first histogram compared to the second histogram. Furthermore, as there are 256 bins in the second histogram, all pixels are put in the bin that corresponds to their gray levels.

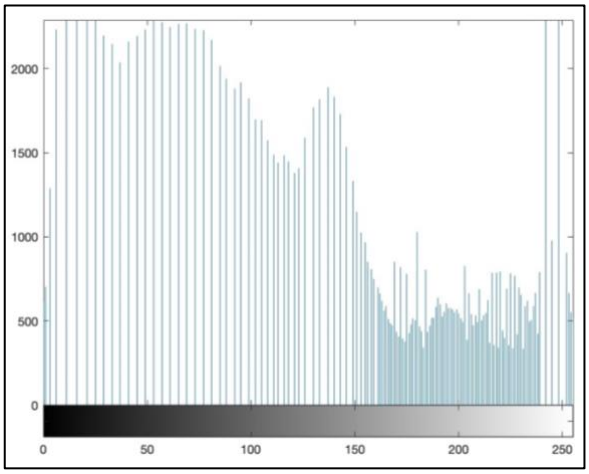
2.2 (b) Histogram Equalization for the image

Histogram equalization for carried out for image P using the following code snippet:

```
% 2.2 (b)
P3 = histeq(P, 255);
imhist(P3, 10);
imhist(P3, 256);
```

The histograms after equalization were as follows:



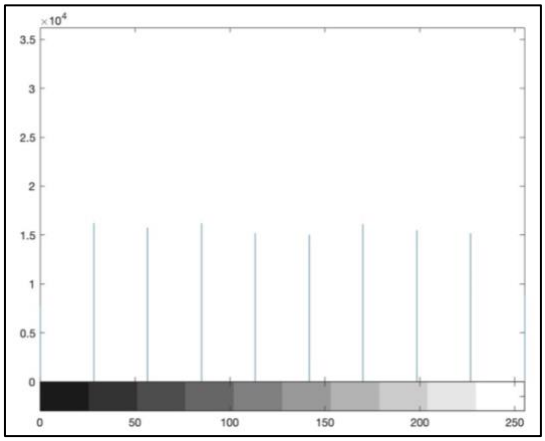


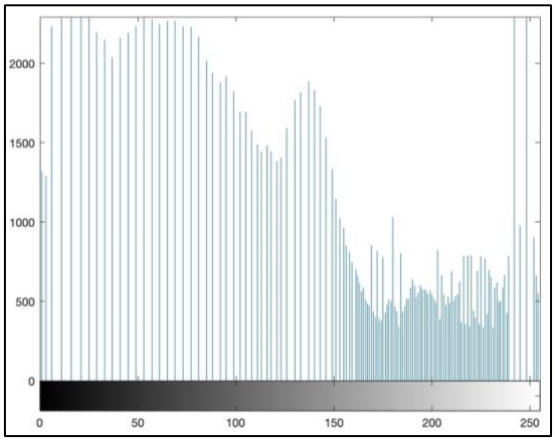
10 bin 256 bin

The 10 bin histogram roughly seems to be equalized. However, the 256 bin histogram is comparatively not. They both show improvement in terms of equalization with histograms shown in section 2.2 which implies that pixels are distributed more equally.

2.2 (c) Redoing Histogram Equalization

The equalization was done in the same way as shown in section 2.2 (b). The histograms after equalization were as follows:





10 bin 256 bin

The histograms are similar to the ones shown in 2.2 (b). This is because the equalization was already done. Repeating the process will not yield any different results. The gray level values are discrete and when moving to another bin, all pixels with a particular value have to move together. Once they have moved to another level, repeating the process will not make find a more suitable bin for these newly placed pixels. As such, the histograms remain unchanged.

2.3 Linear Spatial Filtering

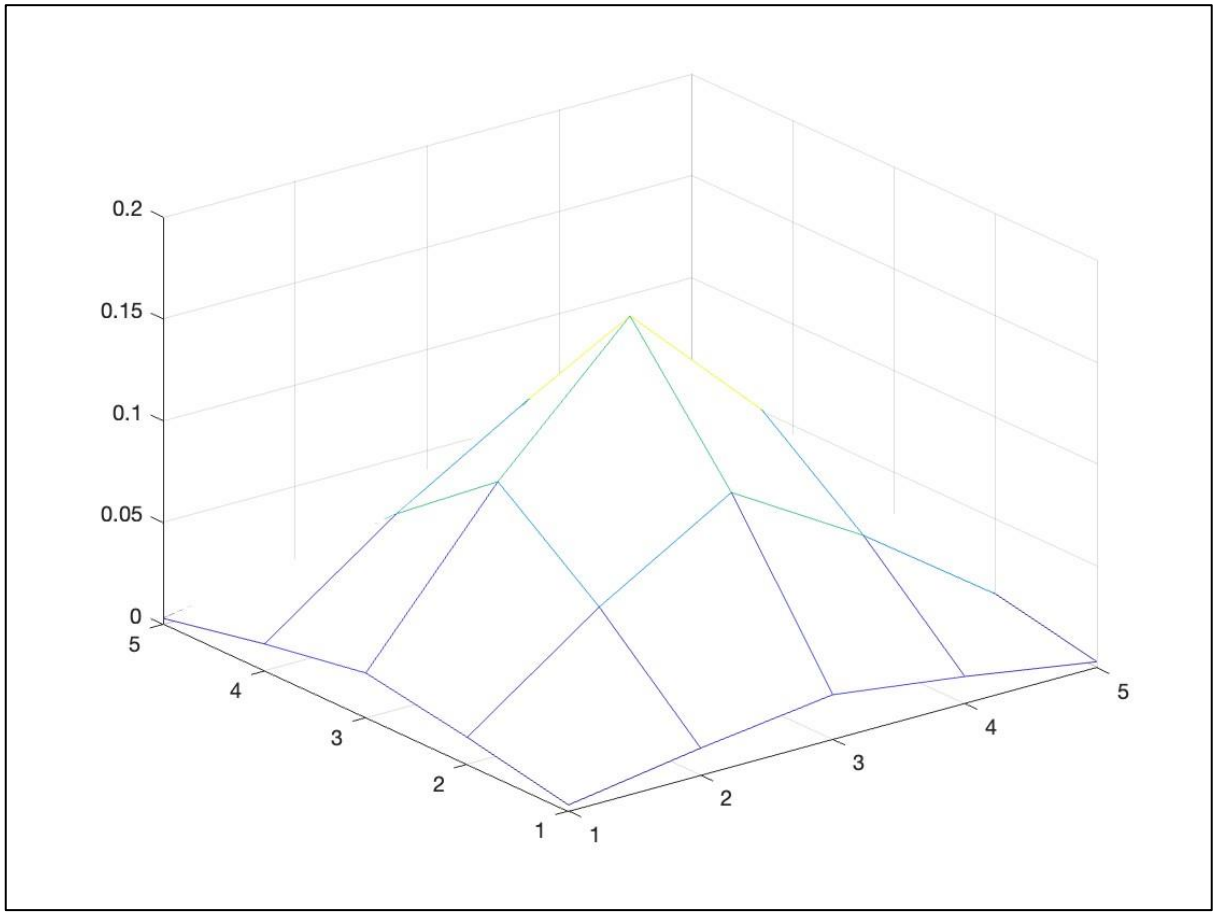
The mesh grid was defined as follows:

```
% 2.3 Linear Spatial Filtering

x = -2:2;
y = -2:2;
[X, Y] = meshgrid(x, y);
```

2.3 (a) Generating filters

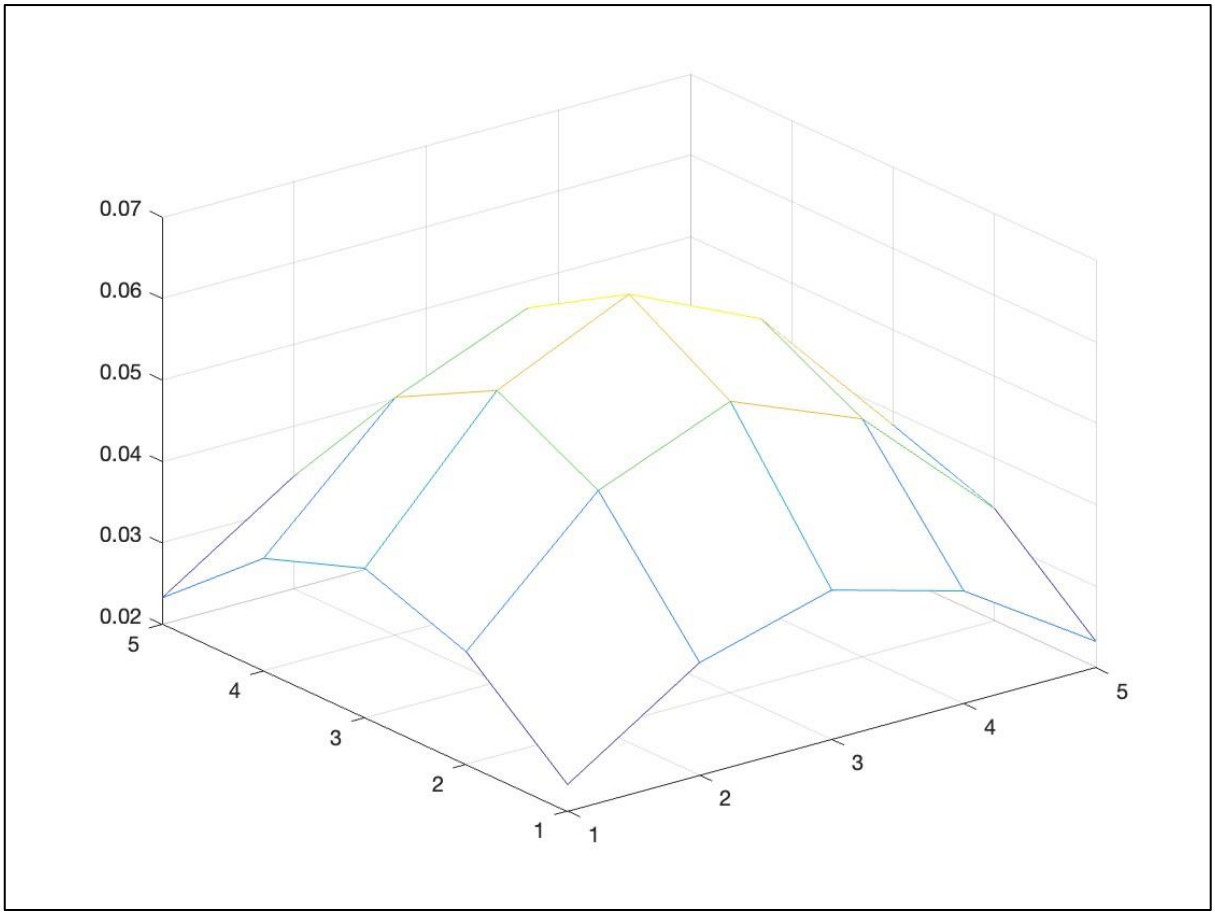
2.3 (a) (i)

```
sigmaA1 = 1.0;
PSF1 = exp(-(X.^2 + Y.^2)/(2 * sigmaA1.^2))/(2 * pi * sigmaA1.^2);
PSF1 = PSF1./sum(PSF1(:));
mesh(PSF1);
```



2.3 (a) (ii)

```
sigmaA2 = 2.0;
PSF2 = exp(-(X.^2 + Y.^2)/(2 * sigmaA2.^2))/(2 * pi * sigmaA2.^2);
PSF2 = PSF2./sum(PSF2(:));
mesh(PSF2);
```



2.3 (b) Reading image with Gaussian Noise

The image was read using the following piece of code:

```
P4 = imread(['/Users/omkaringale/Desktop/Computer Vision/Assignments' ...
    '/images/ntugn.jpg']);
imshow(P4);
```

The image is as follows:

2.3 (c) Filtering the Image

The image was filtered using the two filters generated in the previous section. This was done as follows:

```
% 2.3 (c)

P4_PSF1 = uint8(conv2(P4, PSF1));
imshow(P4_PSF1);

P4_PSF2 = uint8(conv2(P4, PSF2));
imshow(P4_PSF2);
```

The results are as follows:

$\sigma = 1.0$

$\sigma = 2.0$

The results can be summarised as follows:

a. The second filter removed more noise compared to the first filter.

b. The first image looks more detailed compared to the second image.

Based on these findings, it can be inferred that a higher value of σ results in more noise being removed. However, it also means that some details from the image will also be lost.

2.3 (d) Viewing image with Speckle Noise

The image was viewed using the following commands:

```
P5 = imread(['/Users/omkaringale/Desktop/Computer Vision' ...
    '/Assignments/images/ntusp.jpg']);
imshow(P5);
```

The image is as follows:

2.3 (e) Using Filters with Speckle Noise

```
% 2.3 (e)

P5_PSF1 = uint8(conv2(P5, PSF1));
imshow(P5_PSF1);

P5_PSF2 = uint8(conv2(P5, PSF2));
imshow(P5_PSF2);
```

The resulting image after the application of the filters were as follows:

$\sigma = 1.0$

$\sigma = 2.0$

Just like the previous images, $\sigma = 2.0$ seems to do a better job in removing the noise. However, the noise is not entirely eliminated. Hence, as the name suggests, gaussian filter does a better job in eliminating gaussian noise compared to speckle noise. This may be because of the way gaussian filter works. The filter looks at the weighted value of surrounding pixels. However, in this instance, the speckle noise is in the form of pixels that have a very different value compared to its surrounding pixels. As such, they are also included in the filter's calculations and hence the noise is not completely eliminated.

2.4 Median Filtering

The filtering was done using the following code:

```
P6 = medfilt2(P4, [3, 3]);
imshow(P6);

P7 = medfilt2(P4, [5, 5]);
imshow(P7);

P8 = medfilt2(P5, [3, 3]);
imshow(P8);

P9 = medfilt2(P5, [5, 5]);
imshow(P9);
```

The following set of images tries to showcase both images (ntu-gn.jpg and ntu-sp.jpg) with both (Gaussian and Median) filters.

a. ntu-gn.jpg

Original Image

$\sigma = 1.0$

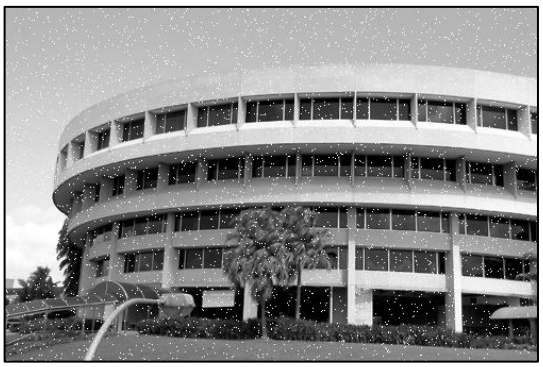
$\sigma = 2.0$

3x3 Median Filter

5x5 Median Filter

As is evident from the above images, median filter is not that efficient in removing gaussian noise, at least with the 3x3 filter. The noise is removed in the second image but much like the gaussian filter with $\sigma = 2.0$, the detail in the image is lost.

b. ntu-sp.jpg



Original Image

$\sigma = 1.0$

$\sigma = 2.0$

3x3 Median Filter

5x5 Median Filter

The median filter handles speckle noise very effectively. The 3x3 filter itself clears out the noise better than the gaussian filter. However, the 5x5 filter makes the image look very unreal. A lot of information from the image appears to be lost.

Conclusion

i. Gaussian filter handles gaussian noise effectively and median filter handles speckle noise effectively.
ii. Gaussian filter considers the weight of all the pixels within the filter and hence the details of the image are never lost as every pixel's weight is considered. This is why gaussian filter works well with gaussian noise as it is distributed over pixels.
iii. As for the median filter, it considers the median of pixels in the filter thus ignoring other pixels. This is why it works with speckle noise as those are spread over certain pixels and get ignored.
iv. In the case of speckle noise, even though a lot of noise is removed by the median filter, the details in the image are also lost. This is not the case with gaussian filter as the noise is not completely removed but the details are also not lost.

Tradeoffs

i. Gaussian Filter: Speckle noise not completely removed but details in the image are preserved.
ii. Median Filter: Speckle noise removed but details are lost.

2.5 Suppressing Noise Interference Patterns

2.5 (a) Reading the image

```
P10 = imread(['/Users/omkaringale/Desktop/Computer Vision/Assignments' ...
    '/images/pckint.jpg']);
imshow(P10);
```

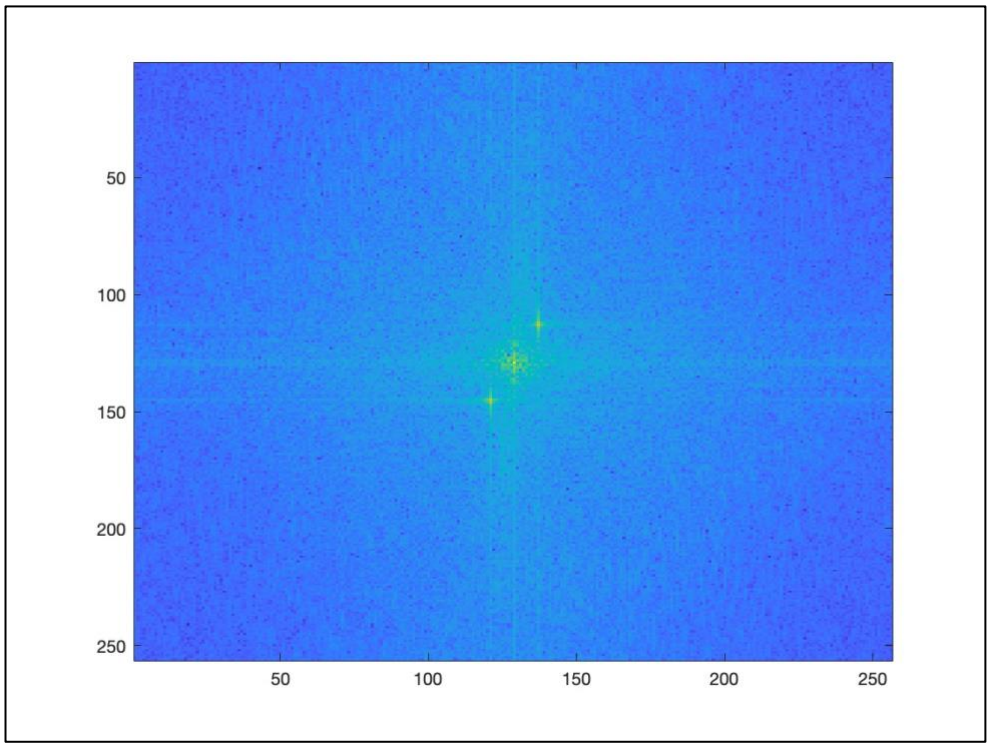
The image is as follows:

2.5 (b) Fourier Transform and Power Spectrum

The Fourier transform and the power spectrum were calculated using the following code:

```
F = fft2(P10);
S = abs(F);
imagesc(fftshift(S.^0.1));
colormap('default');
```

The power spectrum that was obtained was:



2.5 (c) Power Spectrum without fftshift

The coordinates of the peaks were taken using ginput using the following pieces of code.

```
imagesc(S.^0.1);
colormap('default');
[x, y] = ginput(2);
x;
y;
```

The coordinates that were taken using ginput were:

```
>> x

x =

   16.3832
  239.8693
```

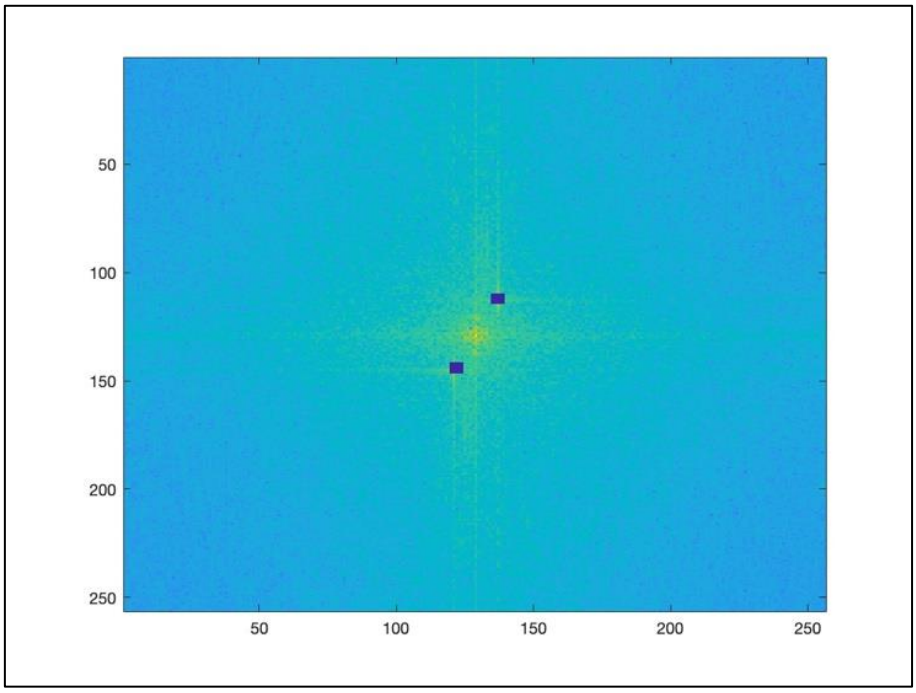
```
>> y

y =

  249.8641
    9.2005
```

2.5 (d) Setting Peaks as 0

This was done by using the following code:

```
x1 = x(1);
y1 = y(1);
x2 = x(2);
y2 = y(2);

F(x1-2:x1+2, y1-2:y1+2) = 0; % 5x5 neighbourhood
F(x2-2:x2+2, y2-2:y2+2) = 0;

S = abs(F);
imagesc(fftshift(S.^0.1));
colormap('default');
```

The output was the following:



2.5 (e) Inverse Fourier Transform

The inverse Fourier transform was calculated as follows:

```
P11 = uint8(ifft2(F));
imshow(P11);
```

The resultant image was as follows:

As we can see that the noise is reduced but not completely removed. The noise corresponds to the peaks that we saw in step c. When we set those peaks to 0 in the Fourier domain, we set the high frequency of the noise pattern in the frequency domain to 0. This resulted in the noise pattern to fade in the image. However, as the noise pattern is not completely gone, we can infer that there is another peak that we did not set to 0. If we found the peak and set it to 0, the noise pattern may disappear.

2.5 (f) Free the Primate

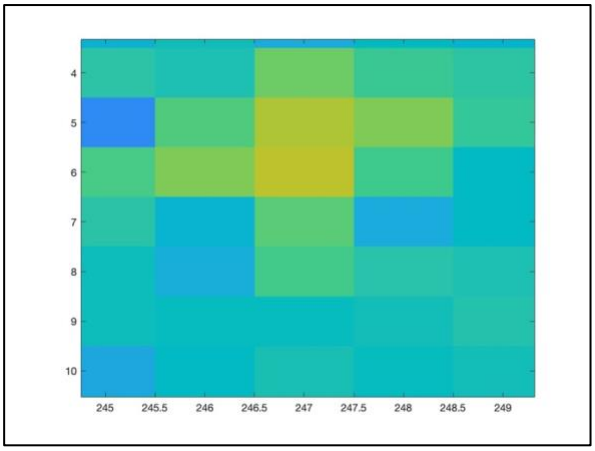
Inputting the caged primate image:

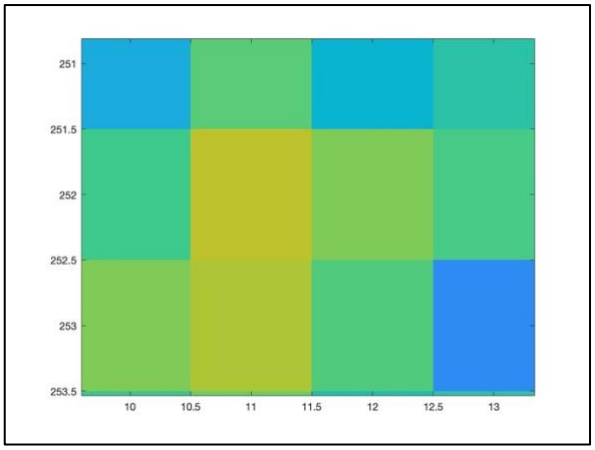
```
P12 = imread(['/Users/omkaringale/Desktop/Computer Vision/Assignments' ...
    '/images/primatecaged.jpg']);
imshow(P12);
```

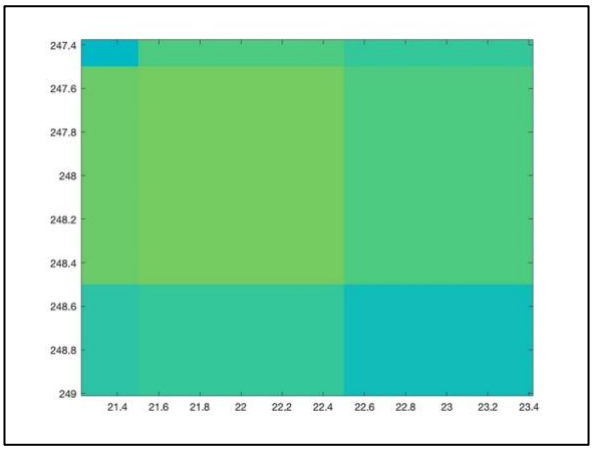
The picture is as follows:

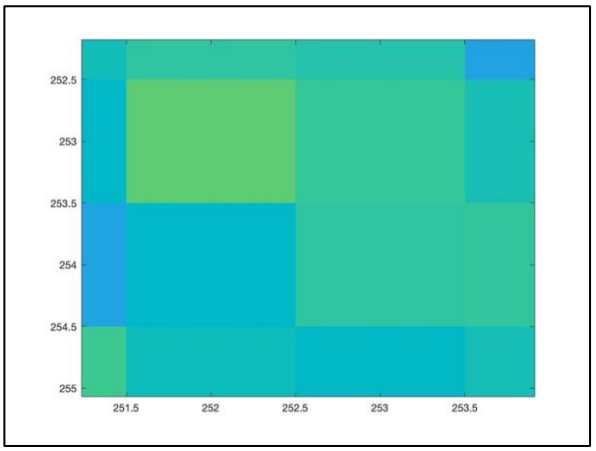
```
>> whos P12
  Name        Size                    Bytes  Class    Attributes
  P12         256x256x3              196608  uint8
```

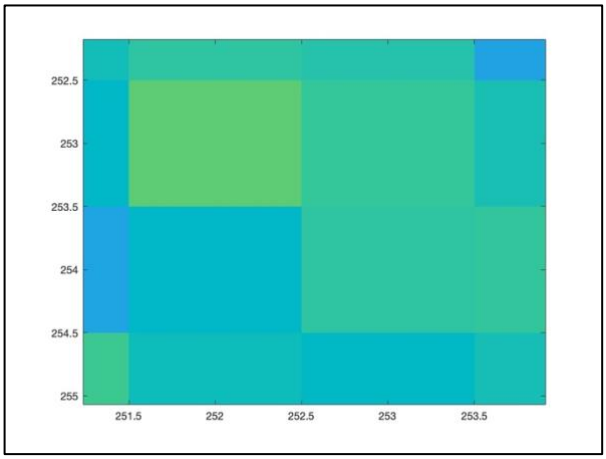
The whos command reveals that the picture has 3 dimensions. As such the grb2gray command was used to reduce it to a one dimension gray pixel image. The fourier transform and power spectrum were calculated to find the coordinates of the peaks. Five peaks were found and their pixel images are as follows:







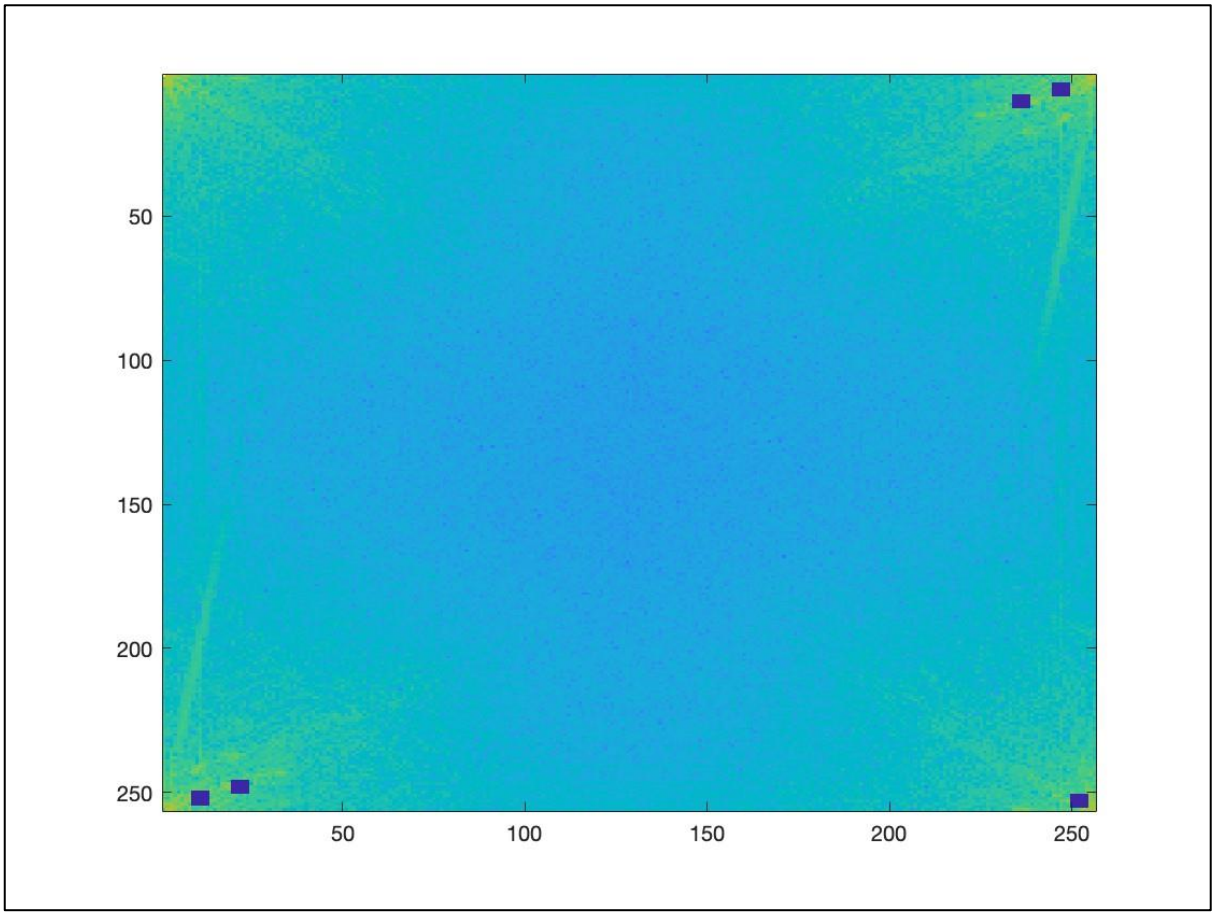




The above mentioned coordinates were saved and the corresponding coordinates were set to 0 in the fourier transform. The code is as follows:

```
y1 = 247; x1 = 6;
y2 = 236; x2 = 10;
y3 = 11; x3 = 252;
y4 = 22; x4 = 248;
y5 = 252; x5 = 253;

F(x1-2:x1+2, y1-2:y1+2) = 0;
F(x2-2:x2+2, y2-2:y2+2) = 0;
F(x3-2:x3+2, y3-2:y3+2) = 0;
F(x4-2:x4+2, y4-2:y4+2) = 0;
F(x5-2:x5+2, y5-2:y5+2) = 0;

S = abs(F);
imagesc(S.^0.1);
colormap('default');

P13 = uint8(ifft2(F));
imshow(P13);
```

The new power spectrum is as follows:



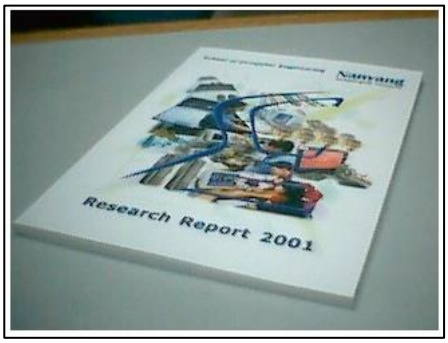
The final image is shown below alongside the original image for comparison:

The primate is not completely free. An explanation can be that the cage is not really a noise pattern. It is a real part of the image and hence does not adhere to the typical properties of a noise pattern.

2.6 Undoing Perspective Distortion of Planar Surface

2.6 (a) Inputting the image

```
P14 = imread(['/Users/omkaringale/Desktop/Computer Vision/' ...
    'Assignments/images/book.jpg']);

imshow(P14);
```

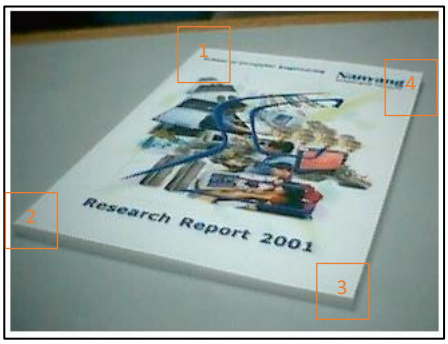


2.6 (b) Taking coordinates of input image

The following code was used to get the 4 coordinates of the image:

```
[X, Y] = ginput(4);
```

The order in which the 4 coordinates were selected was as follows:



The variables X and Y are as follows:

```
>> X

X =

  146.0000
    2.0000
  256.0000
  310.0000
```

```
>> Y

Y =

   26.0000
  158.0000
  214.0000
   46.0000
```

The vectors of the 4 corners of the desired image shown in the image below. They are labelled 0, 210, 297 in accordance with the dimensions of an A4 paper. This is done to as 1 pixel I required to be equal to 1mm on paper.

```
% A4 dimension 210mm x 297mm
imageX = [0, 0, 210, 210];
imageY = [0, 297, 297, 0];
```

2.6 (c) Setting up matrices

The matrices were set up as follows:

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

v = [imageX(1); imageY(1); imageX(2); imageY(2); imageX(3); imageY(3);
    imageX(4); imageY(4)];
```

The matrices U and w are as follows:

```
U =

    1.4194    1.5485 -247.4969
   -0.4418    3.6231  -29.6917
    0.0001    0.0052    1.0000

>> w

w =

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

The matrix w does represent the original values in matrix imageX and image Y. The coordinates being: (0, 0), (0, 297), (210, 297), (210, 0).

2.6 (d) Warp the Image

```
T = maketform('projective', U');
P15 = imtransform(P14, T, 'XData', [0 210], 'YData', [0 297]);

imshow(P15);
```

2.6 (e) Final Image

The image has achieved the final goal: to shift the book in proportion to an A4 image where 1 pixel = 1mm. The bottom of the image is very readable as the words “Research Report 2001” can be clearly seen. The top of the report, however, is not that great. The only words readable are “Nanyang”. However, considering that the quality of the original image is not that great (words that cannot be seen in the current image cannot be seen in the original image as well), it lives up to the expectations.

2.6 (f) Computer Screen

Note: “To identify” is assumed to mean crop and enhance the given computer image. It is done using the same techniques employed in previous sections. The code was as follows:

```
% 2.6 (f)
imshow(P15);
[X, Y] = ginput(4);
imageX = [0, 50, 50, 0];
imageY = [0, 0, 50, 50];

A = [
        [X(1), Y(1), 1, 0, 0, 0, -imageX(1)*X(1), -imageX(1)*Y(1)];
        [0, 0, 0, X(1), Y(1), 1, -imageY(1)*X(1), -imageY(1)*Y(1)];
        [X(2), Y(2), 1, 0, 0, 0, -imageX(2)*X(2), -imageX(2)*Y(2)];
        [0, 0, 0, X(2), Y(2), 1, -imageY(2)*X(2), -imageY(2)*Y(2)];
        [X(3), Y(3), 1, 0, 0, 0, -imageX(3)*X(3), -imageX(3)*Y(3)];
        [0, 0, 0, X(3), Y(3), 1, -imageY(3)*X(3), -imageY(3)*Y(3)];
        [X(4), Y(4), 1, 0, 0, 0, -imageX(4)*X(4), -imageX(4)*Y(4)];
        [0, 0, 0, X(4), Y(4), 1, -imageY(4)*X(4), -imageY(4)*Y(4)];
];

v = [imageX(1); imageY(1); imageX(2); imageY(2); imageX(3); imageY(3);
    imageX(4); imageY(4)];

u = A \ v;

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

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

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

T = maketform('projective', U');
P16 = imtransform(P15, T, 'XData', [0 50], 'YData', [0 50]);

imshow(P16);

P17 = histeq(rgb2gray(P16), 255);
imshow(P17);

P18 = uint8(conv2(P17, PSF1));
imshow(P18);
```

The output was as follows:

As can be inferred from the picture, the content on the screen of the computer cannot be decoded effectively. This may be because the original image from which the image of the notebook was derived was itself shot at a very low resolution. The above image, however, does retain some of the features from the original image. For example, the black curve at the top left of the image is from the exhaust pipes that can be seen over the computer screen.