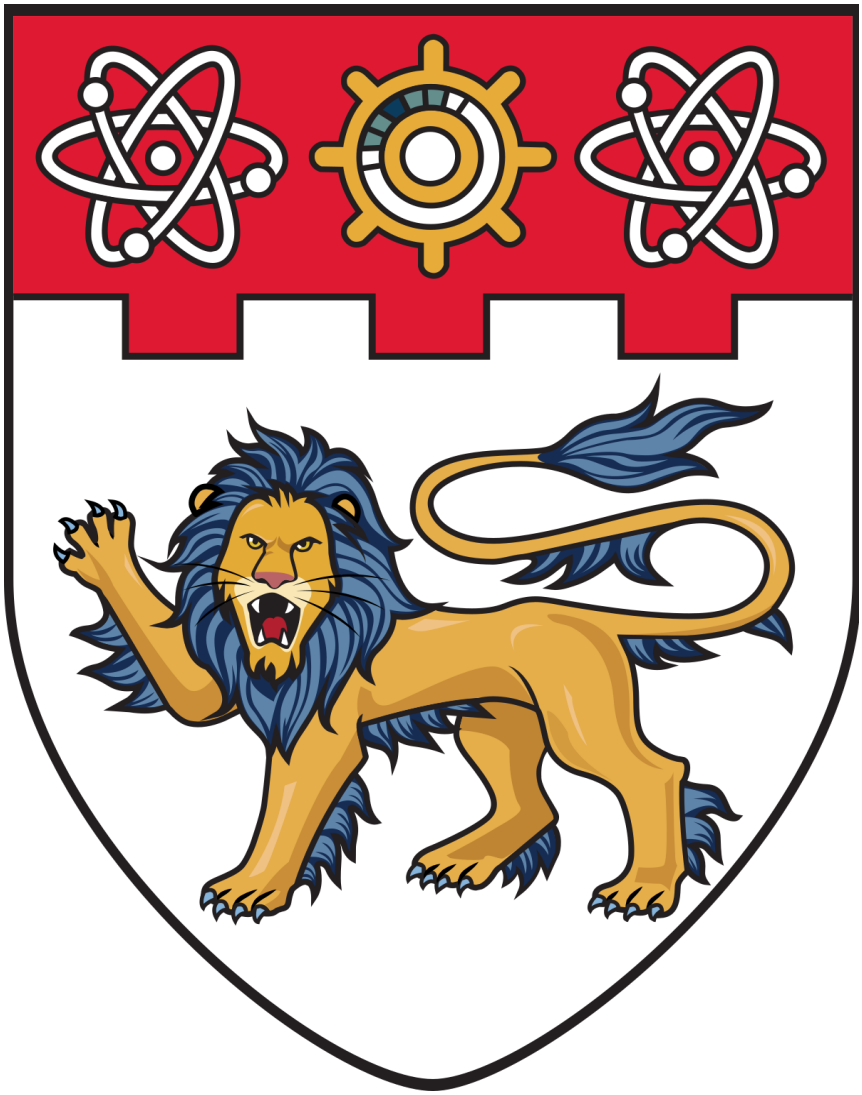

Name: Tha Toe Oo @ Jodi
MatricID: U1621448G
Subject Code: CE4003

Contrast Stretching

a) Question:

Input the image into a MATLAB matrix variable by executing:

>> Pc = imread('mrt-train.jpg');
>> whos Pc

The whos command is to show whether the image is read as an RGB or gray-scale image. Notice that this is a 320x443x3 uint8 matrix indicating a colour image with byte values. You will need to convert this into a grayscale image by:

>> P = rgb2gray(Pc);

Answer:

The following Matlab code can be used to read the image and run the whos command:

```
Pc = imread('../images/mrt-train.jpg');
whos Pc;
```

Using whos command returns the following output.

```
  Name        Size                Bytes  Class    Attributes

  Pc        320x443x3             425280  uint8
```

As expected, the output shows that the image can be read as a 320x443x3 uint8 matrix indicating a color image with byte values.

The following Matlab code can be used to convert the image to gray using the rgb2gray command and to run the whos command again:

```
P = rgb2gray(Pc);
whos P;
```

After converting the image to gray using the rgb2gray command, the dimension of the matrix changes from 320x443x3 to 320x443 as shown by the output below.

```
  Name        Size                Bytes  Class    Attributes

  P         320x443               141760  uint8
```

This shows that the original color image has been converted to a grey-scale image.

b) Question:

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

Answer:

To view the image, the following Matlab code can be used:

```
imshow(P);
```

Figure 1: Image of the original MRT Train

c) Question:

Check the minimum and maximum intensities present in the image:

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

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

Answer:

To find the minimum and maximum intensities present in the image, the following Matlab code can be used:

```
min(P(:));
max(P(:));
```

Using min(P(:)) returns 13 and using max(P(:)) returns 204 as seen from the output below.

```
ans =

  uint8

   13


ans =

  uint8

   204
```

d) Question:

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

Answer:

First, the minimum and the maximum intensity of the image will be stored into the variables shown below after being converted into double:

```
P_minVal = double(min(P(:)));
P_maxVal = double(max(P(:)));
```

The code used to do contrast stretching is the following:

```
subOperation = imsubtract(P, P_minVal);
P2 = immultiply(subOperation, 255 / (P_maxVal - P_minVal));
```

Using min(P2(:)) returns 0 and max(P2(:)) returns 255 as shown by the output below. This shows the the final image P2 has the correct minimum and maximum intensities of 0 and 255.

```
>> min(P2(:))

ans =

  uint8

    0

>> max(P2(:))

ans =

  uint8

    255
```

e) Question:

Finally, redisplay the image by

>> imshow(P2);

Note again that if you choose to convert your byte images to doubles first, you will need to use the uint8 command is necessary to convert the P2 double values back to byte values. Otherwise, imshow detects P2 to be double and assumes that the intensity range is between 0.0 and 1.0. An alternative is to use the imshow as such:

>> imshow(P2,[]);

This does automatic contrast stretching of the display but does not affect the input matrix. This allows you to ignore whether you are displaying byte or double-valued images.

Answer:

The following Matlab code can be used to show the image:

```
imshow(P2);
```

The image below shows the image after contrast stretching. As we can see the contrast of the image has improved.

Figure 2: Image of MRT after contrast stretching has been conducted

f) Source Code:

```
Pc = imread('../images/mrt-train.jpg');
whos Pc;
P = rgb2gray(Pc);
whos P;
imshow(P);
P_minVal = double(min(P(:)));
P_maxVal = double(max(P(:)));
subOperation = imsubtract(P, P_minVal);
P2 = immultiply(subOperation, 255 / (P_maxVal - P_minVal));
imshow(P2);
```

Histogram Equalization

a) Question:

Display the image intensity histogram of P using 10 bins by

>> imhist(P,10);

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

Answer:

Image intensity of P using 10 bins can be displayed by using the following code:

```
imhist(P, 10);
```

Image intensity of P using 256 bins can be displayed by using the following code:

```
imhist(P, 256);
```

Figure 3 shows the image intensity histogram of P using 10 bins while Figure 4 shows the image intensity histogram of P using 256 bins. The differences between the two figures are that the image intensity histogram of P using 256 bins show a better distribution of grey levels than when using 10 bins. The histogram with 10 bins holds more pixels per bin than the histogram with 256 bins so it has much less details about the distribution of grey levels. Also, since the greyscale images have 256 grey levels, the histogram with 256 bins can show us the details of every grey level.

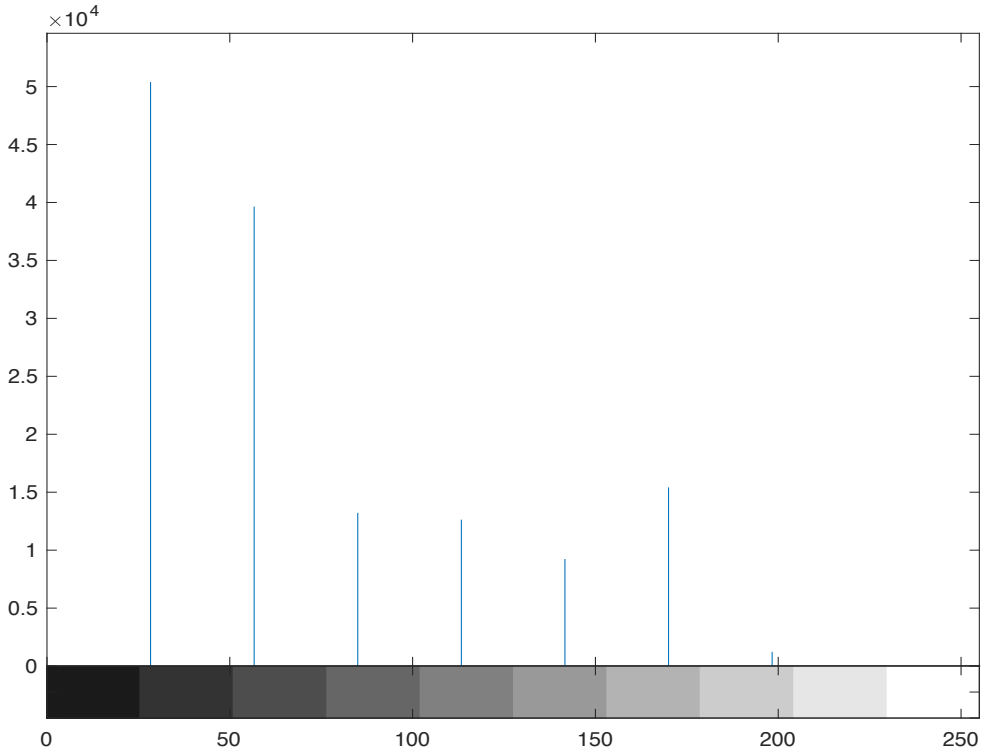

Figure 3 Image intensity histogram of P using 10 bins

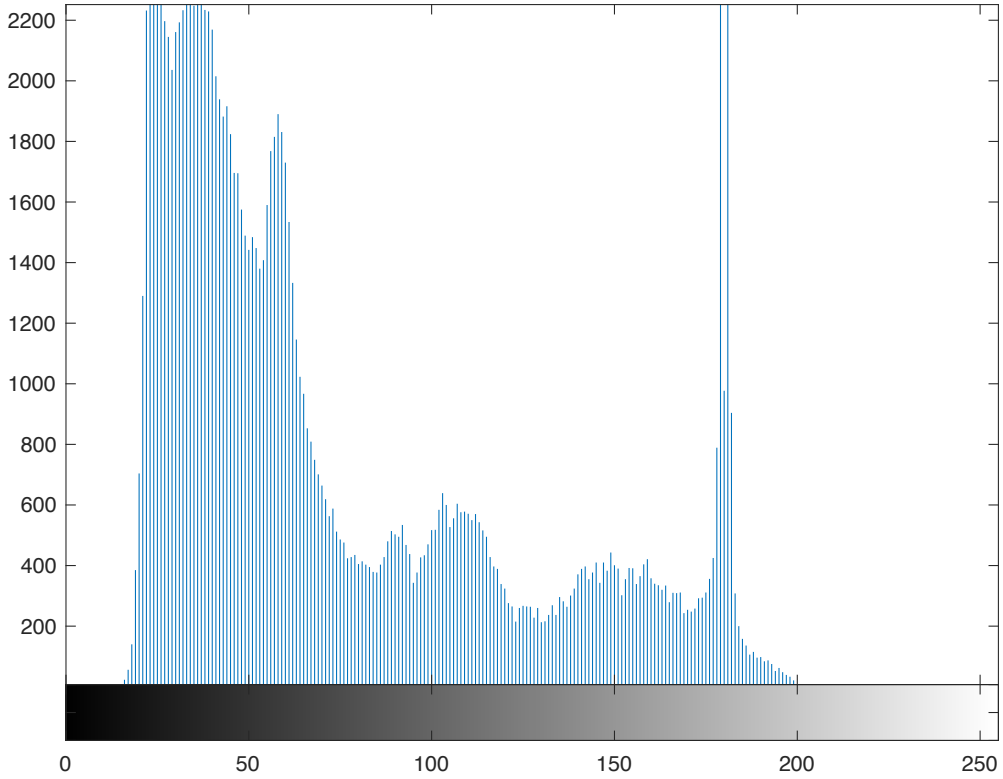

Figure 4: Image intensity histogram of P using 256 bins

b) Question:

Next, carry out histogram equalization as follows:

>> P3 = histeq(P,255);

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

Answer:

The following code can be used to carry out histogram equalization:

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

Figure 5 shows the image intensity histogram of P3 using 10 bins while Figure 6 shows the image intensity histogram of P3 using 256 bins. Both images display behaviour of the histogram becoming more uniform. However, this is more noticeable for the histogram of P3 using 10 bins as compared to the histogram using 256 bins as in the former, the histogram looks almost flat. This shows that at a more detailed level of 256 bins, the histogram won't be totally flat after histogram equalization but at a less detailed level of 10 bins, the histogram looks fairly equalized.

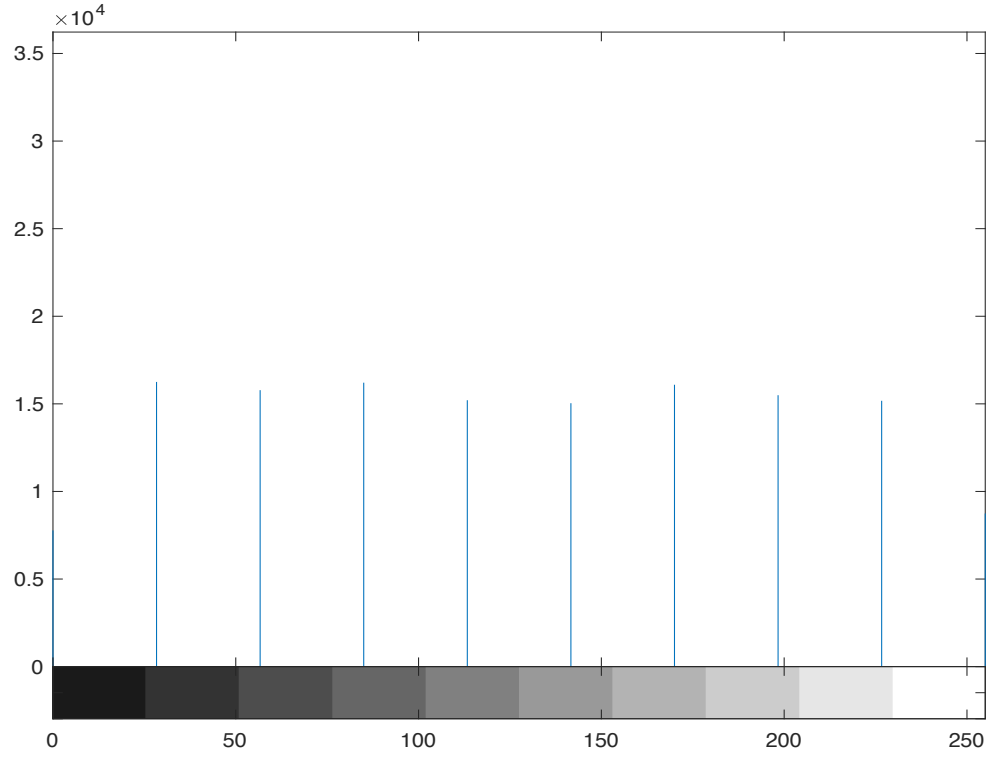

Figure 5: Image intensity histogram of P3 after histogram equalization with 10 bins

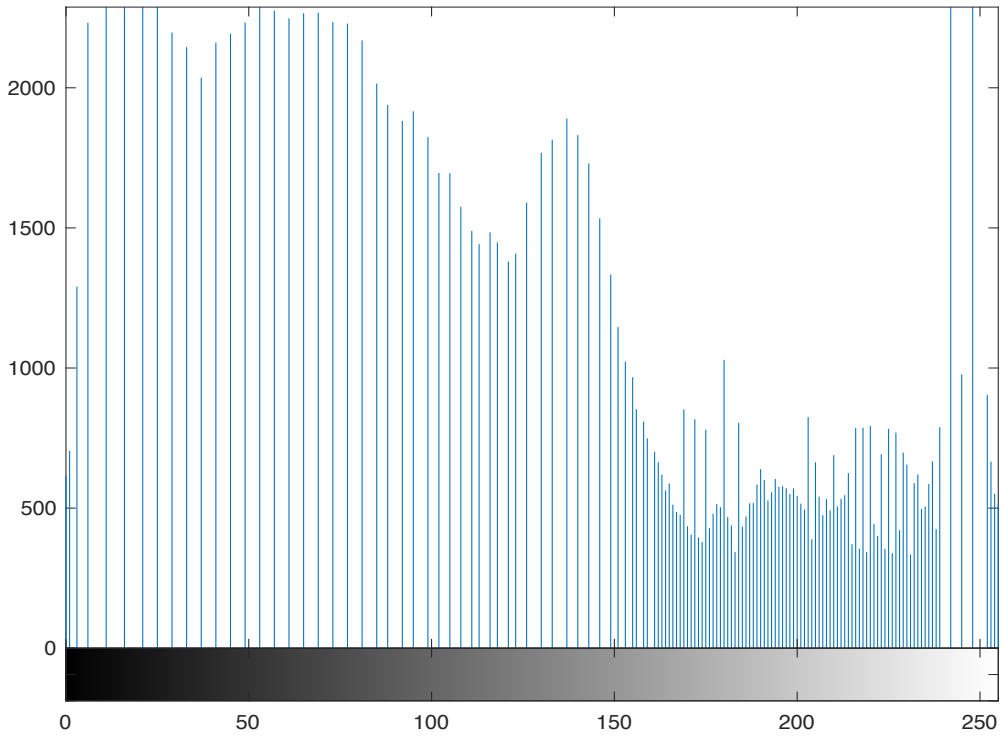

Figure 6: Image intensity histogram of P3 after histogram equalization with 256 bins

c) Question:

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

Answer:

The following code can be used to rerun histogram equalization on P3:

```
P3_re = histeq(P3, 255);
```

No. The histogram does not become more uniform. Figure 7 shows the image intensity histogram of P3 after rerunning histogram equalization using 10 bins while Figure 8 shows the image intensity histogram of P3 after rerunning histogram equalization using 256 bins. The histograms remain roughly the same. This is because after the first run of histogram equalization, the histogram is already mapped out based on the average number of pixels in each bin such that the distribution of the grey level intensity will be as uniform as possible. As a result, most of the bins would already be in the right place and this will cause the histogram to remain roughly the same.

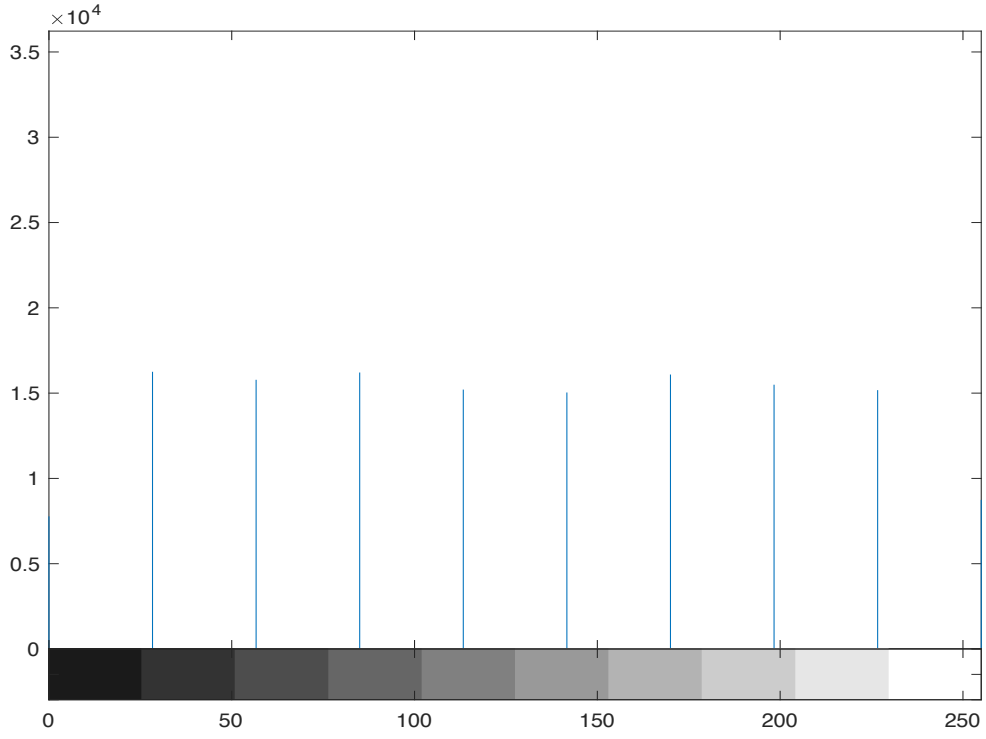

Figure 7: Image intensity histogram after rerunning histogram equalization with 10 bins

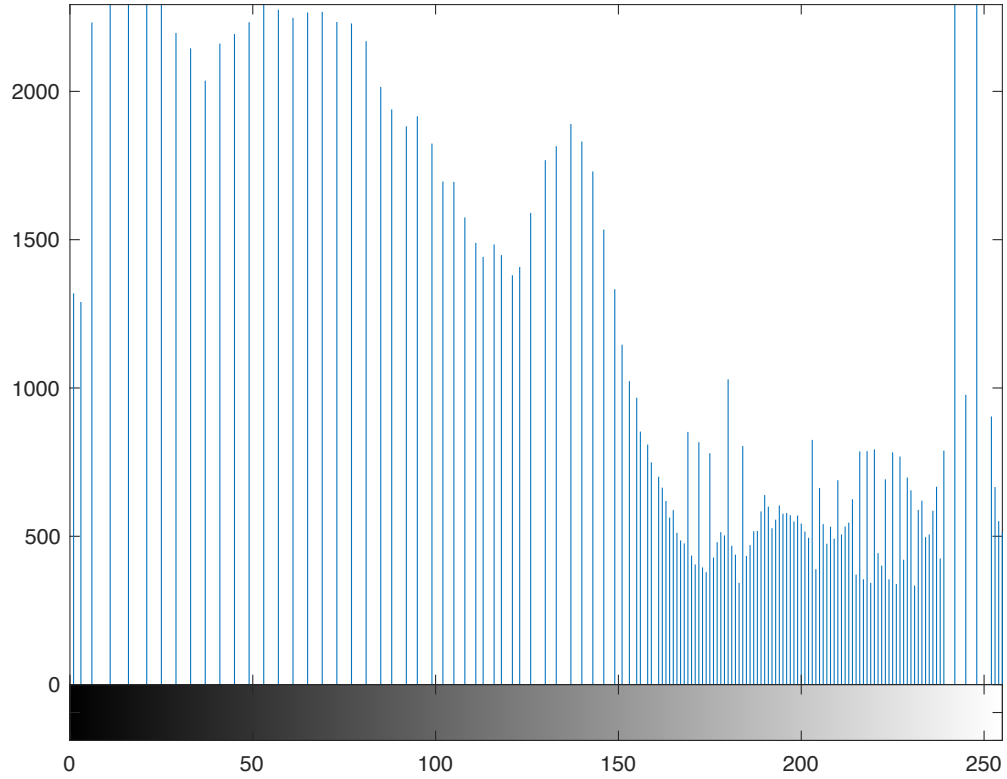

Figure 8: Image intensity histogram after rerunning histogram equalization with 256 bins

d) Source Code:

```
Pc = imread('../images/mrt-train.jpg');
P = rgb2gray(Pc);
imhist(P, 10);
imhist(P, 256);
P3 = histeq(P, 255);
imhist(P3, 10);
imhist(P3, 256);
P3_re = histeq(P3, 255);
imhist(P3_re, 10);
imhist(P3_re, 256);
```

Linear Spatial Filtering

a) Question:

Generate the following filters:

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

(i) Y and X-dimensions are 5 and σ = 1.0
(ii) Y and X-dimensions are 5 and σ = 2.0

Normalize the filters such that the sum of all elements equals to 1.0. View the filters as 3D graphs by using the mesh function. These filters are Gaussian averaging filters.

Answer:

The gaussian filters can be generated using the fspecial function as shown below.

Gaussian filter with sigma = 1.0:

```
sigma = 1.0;
hsize = 5;
filter = fspecial('gaussian', hsize, sigma);
```

Gaussian filter with sigma = 2.0:

```
sigma = 2.0;
hsize = 5;
filter = fspecial('gaussian', hsize, sigma);
```

The following code can be used to create image of the filters using the mesh function:

```
mesh(filter);
```

Viewing the the gaussian filter with the mesh function will result in the following images:

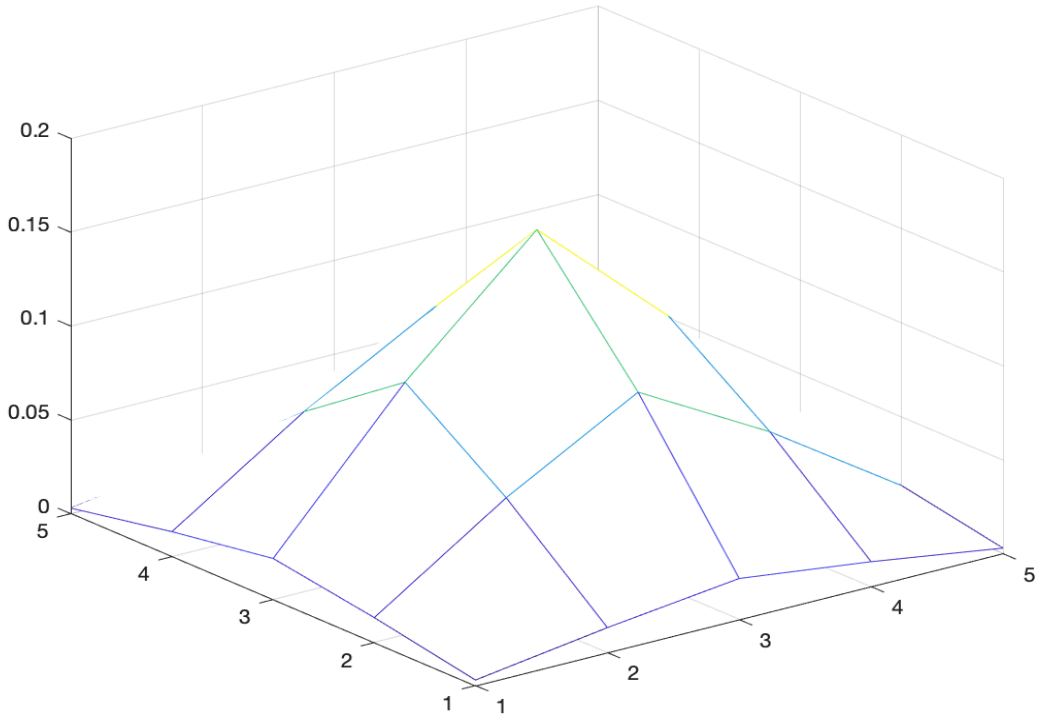

Figure 9: Image of filter viewed with mesh function for sigma = 1.0

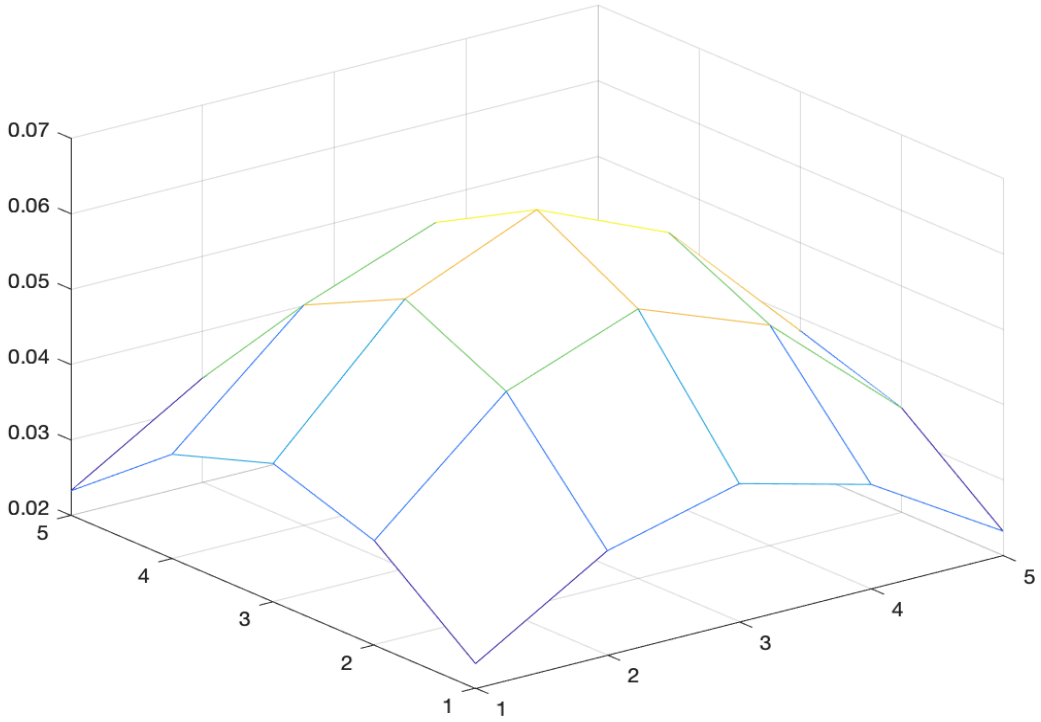

Figure 10: Image of filter viewed with mesh function for sigma = 2.0

b) Question:

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

Answer:

The following code can be used to view the image:

```
Pc = imread('../images/ntu-gn.jpg');
imshow(Pc);
```

It will result in the following image being shown on Matlab:

Figure 11: Original image with additive Gaussian noise

c) Question:

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

Answer:

The following code can be used to filter the image with the linear filter and show the resulting image:

```
P4 = uint8(conv2(Pc, filter));
imshow(P4);
```

The following images will be generated after applying the linear filters:

Figure 12: Image with additive Gaussian noise after application of filter with sigma = 1.0

Figure 13: Image with additive Gaussian noise after application of filter with sigma = 2.0

Using a filter with a higher sigma value will remove more noise. However it will also make the image appear more blur. We can see that a filter with a higher coefficient is more effective at filtering noise but will also make the image more blur.

d) Download the image 'ntu-sp.jpg' from edveNTUre and view it. Notice that this image has additive speckle noise.

The following code can be used to view the image:

```
Pc = imread('../images/ntu-sp.jpg');
imshow(Pc);
```

This will result in the following image being shown on Matlab:

Figure 14: Original image with additive speckle noise

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

The following code can be used to filter the image with the linear filter and show the resulting image:

```
P4 = uint8(conv2(Pc, filter));
Imshow(P4);
```

The following images will be generated after applying the linear filters:

Figure 15: Image with additive speckle noise after application of filter with sigma = 1.0

Figure 16: Image with additive speckle noise after application of filter with sigma = 2.0

Similar to the image with additive Gaussian noise, using a filter with a higher sigma value will remove more noise. However it will also make the image appear more blur. We can see that a filter with a higher coefficient are more effective at filtering noise but will also make the image more blur.

However, the filters do not remove speckle noise as well as they remove gaussian noise as we can still see the speckle noise distinctly even with the filter with the higher sigma. This shows that the filters are better at removing gaussian noise than speckle noise.

f) Source Code:

```
sigma = 1.0;
sigma = 2.0;
hsize = 5;
filter = fspecial('gaussian', hsize, sigma);
mesh(filter);
Pc = imread('../images/ntu-gn.jpg');
imshow(Pc);
P4 = uint8(conv2(Pc, filter));
imshow(P4);
Pc = imread('../images/ntu-sp.jpg');
imshow(Pc);
P4 = uint8(conv2(Pc, filter));
imshow(P4);
```

Median Filtering

Question:

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

Answer:

To perform median filtering with size of 3x3 filter and to show the resulting image, the following code can be used:

```
median_filter_size = [3 3];
P5 = uint8(medfilt2(Pc, median_filter_size));
imshow(P5);
```

To perform median filtering with size of 5x5 filter and to show the resulting image, the following code can be used:

```
median_filter_size = [5 5];
P5 = uint8(medfilt2(Pc, median_filter_size));
imshow(P5);
```

The following images are generated after performing filtering with the median filter generated above:

Figure 17: Image with additive Gaussian noise that has undergo median filtering with filter of size 3x3

Figure 18: Image with additive Gaussian noise that has undergo median filtering with filter of size 5x5

Figure 19: Image with additive speckle noise that has undergo median filtering with filter of size 3x3

Figure 20: Image with additive speckle noise that has undergo median filtering with filter of size 5x5

From the images above, it can be seen that median filtering is better at filtering speckle noise than gaussian noise as even a median filter of size 3x3 is able to remove speckle noise better than the linear filters used in Gaussian filtering. Gaussian filtering is better at filtering gaussian noise.

The tradeoffs of using median filters are that they result in a greater information loss compared to using Gaussian filter. So if information details are important, Gaussian filter should be used instead.

Source Code:

```
median_filter_size = [3 3];
median_filter_size = [5 5];
Pc = imread('../images/ntu-gn.jpg');
P5 = uint8(medfilt2(Pc, median_filter_size));
Pc = imread('../images/ntu-sp.jpg');
P5 = uint8(medfilt2(Pc, median_filter_size));
imshow(P5);
```

Suppressing Noise Interference Patterns

a) Question:

Download the image ‘pck-int.jpg’ from edveNTUre and display it from MATLAB. Notice the dominant diagonal lines destroying the quality of the image.

Answer:

The following code can be used to view the image:

```
Pc = imread('../images/pck-int.jpg');
imshow(Pc);
```

This will result in the following image being shown on Matlab:

Figure 21: Original image with dominant diagonal lines

b) Question:

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

>> imagesc(fftshift(S.^0.1));
>> colormap(‘default’);

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

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

Answer:

The following code can be used to obtain the Fourier transform F of the image using fft2, the power spectrum S and to display the power spectrum:

```
F = fft2(Pc);
S = abs(F).^2 / length(Pc);
imagesc(fftshift(S.^0.1));
colormap('default');
```

It will result in the following image:

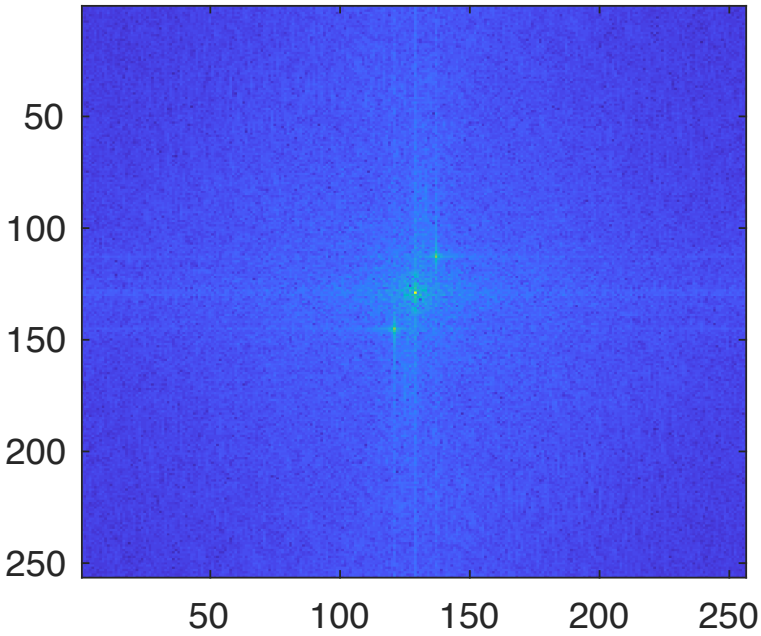

Figure 22: Power Spectrum after fftshift

c) Question:

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

Answer:

The following code can be used to redisplay the power spectrum without fftshift.

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

It will result in the following image:

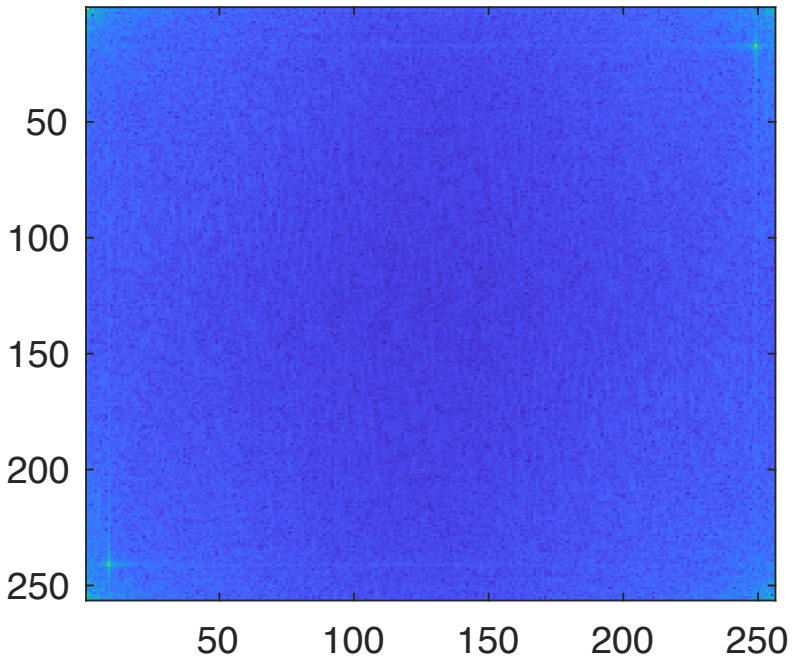

Figure 23: Power Spectrum without fftshift

By referring to the image generated, we can find out that the peak can be found at the following points:

(y, x) = (17, 249) and (241, 9).

d) Question:

Set to zero the 5x5 neighbourhood elements at locations corresponding to the above peaks in the Fourier transform F, not the power spectrum. Recompute the power spectrum and display it as in step (b).

Answer:

The following code can be used to set to zero the 5x5 neighbourhood elements at locations corresponding to the peaks in the Fourier transform F, compute the power spectrum and display it.

```
y1 = 17;
x1 = 249;
y2 = 241;
x2 = 9;
F(y1 - 2:y1 + 2, x1 - 2:x1 + 2) = 0;
F(y2 - 2:y2 + 2, x2 - 2:x2 + 2) = 0;
S = abs(F).^2 / length(Pc);
imagesc(S.^0.1);
colormap('default');
```

It will result in the following image:

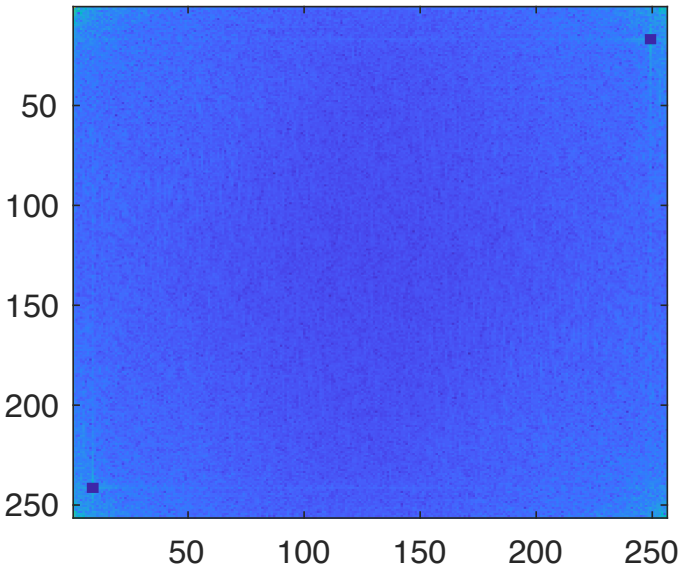

Figure 24: Power spectrum with the 5x5 neighbourhood elements at locations corresponding to the peaks set to zero

e) Question:

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

Answer:

The following code can be used to compute the inverse fourier transform using ifft2 and display the resultant image.

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

Figure 25: Resulting image after inverse fourier transform

As you can see the image looks better and the intensity of the original diagonal lines have been reduced especially around the central region. However, there are still some highly obvious diagonal lines remaining. To try and improve this result, the following can be done:

The horizontal and vertical lines extending from the peaks identified can be set to zero. The following code can be used to do so:

```
F(y1,:) = 0;
F(y2,:) = 0;
F(:,x1) = 0;
F(:,x2) = 0;
```

To observe the new power spectrum the following code can be used:

```
S = abs(F).^2 / length(Pc);
imagesc(S.^0.1);
colormap('default');
```

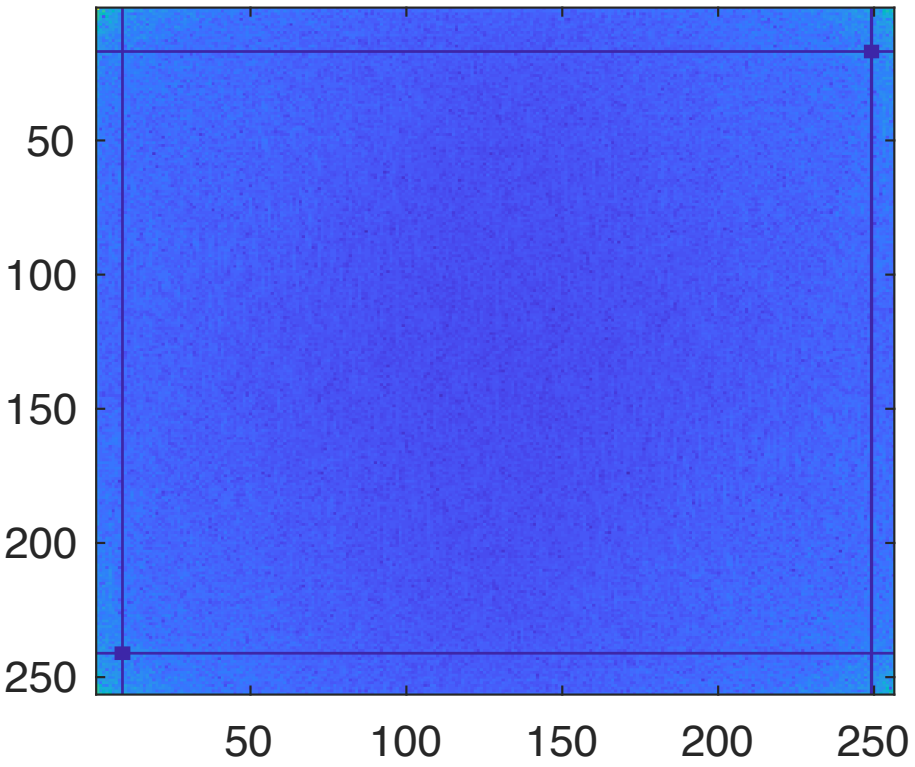

Figure 26: Power spectrum of image after the horizontal and vertical lines extending from the peaks identified is set to zero.

To display the resulting image the following code can be used:

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

The Matlab code will result in the following image:

Figure 27: Resulting image after horizontal and vertical lines extending from the peaks identified is set to zero

As you can see from the resulting image, the intensity of the diagonal lines has been reduced and has become less obvious. However, the trade off is that the contrast of the image has been reduced and the image has become more blurred. In order to improve the contrast of the image, contrast stretching can be carried out in the following manner:

```
P_minVal = double(min(result(:)));
P_maxVal = double(max(result(:)));
subOperation = imsubtract(result, P_minVal);
P2 = immultiply(subOperation, 255 / (P_maxVal - P_minVal));
```

It will result in the following image:

f) Question:

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

Answer:

The following code can be used to show the primate photo in Matlab:

```
Pc = imread('../images/primate-caged.jpg');
Pc = rgb2gray(Pc);
imshow(Pc);
```

The code will result in the image shown below:

Figure 28: Original Caged Primate Image

The power spectrum that has been fftshift can be shown using the code below:

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

It will result in the following power spectrum image:

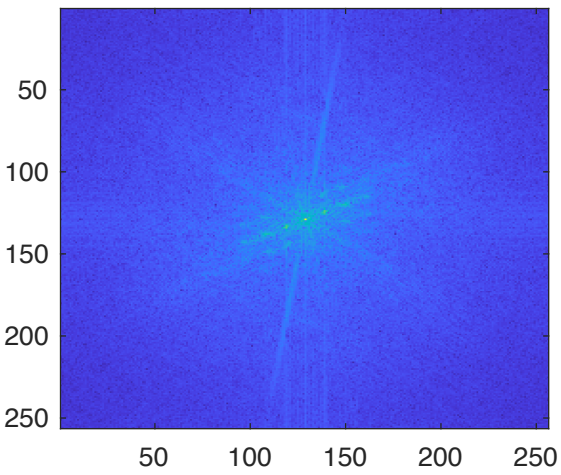

Figure 29: Power spectrum of image after fftshift

The power spectrum of the image without fftshift can be shown using the following code:

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

It will result in the image shown below:

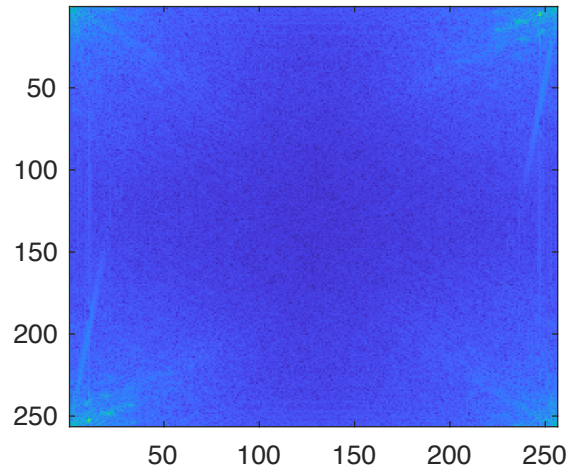

Figure 30: Power spectrum of image without fftshift

From the power spectrum, the following peaks can be found:

(y, x) = (252, 11), (6, 247), (248, 21), (10, 237), (2, 255), (254, 3)

The following code can be using to set to zero the 5x5 neighbourhood elements at locations corresponding to the peaks in the Fourier transform F, compute the power spectrum and display it.

```
y1 = 252;
x1 = 11;
y2 = 6;
x2 = 247;
y3 = 248;
x3 = 21;
y4 = 10;
x4 = 237;
y5 = 2;
x5 = 255;
y6 = 254;
x6 = 3;
F(y1-2 : y1+2, x1-2 : x1+2) = 0;
F(y2-2 : y2+2, x2-2 : x2+2) = 0;
F(y3-2 : y3+2, x3-2 : x3+2) = 0;
F(y4-2 : y4+2, x4-2 : x4+2) = 0;
F(y5-1 : y5+2, x5-2 : x5+1) = 0;
F(y6-2 : y6+2, x6-2 : x6+2) = 0;
S = abs(F).^2 / length(Pc);
imagesc(S.^0.1);
colormap('default');
```

The code will result in the image shown below:

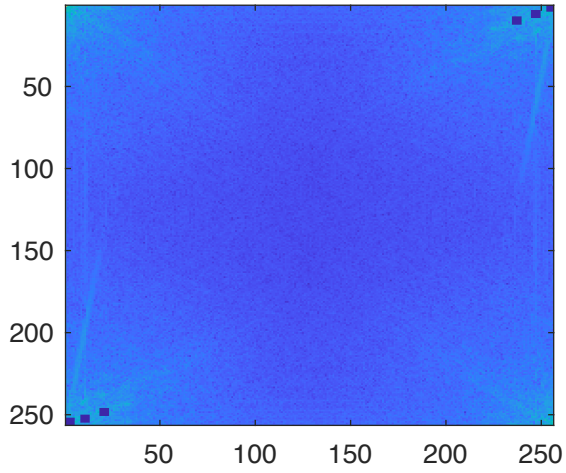

Figure 31: Power spectrum with the 5x5 neighbourhood elements at locations corresponding to the peaks set to zero

In order to show the resulting image, the following code can be used:

```
result = uint8(ifft2(F));
result = real(result);
imshow(result);
```

Figure 32: Resulting image of caged primate with 5x5 neighbourhood elements at locations corresponding to the peaks set to zero

The cage is not removed cleanly even though the cage has been blurred out. This might be because the cage is not actually noise but actually a natural component of the picture.

g) Source Code for removing diagonal lines from pck-int.png:

```
Pc = imread('../images/pck-int.jpg');
imshow(Pc);

F = fft2(Pc);
```

```
S = abs(F).^2 / length(Pc);
imagesc(fftshift(S.^0.1));
imagesc(S.^0.1);
colormap('default');

% Two peaks are at (17, 249) and (241, 9)
y1 = 17;
x1 = 249;
y2 = 241;
x2 = 9;
F(y1 - 2:y1 + 2, x1 - 2:x1 + 2) = 0;
F(y2 - 2:y2 + 2, x2 - 2:x2 + 2) = 0;

S = abs(F).^2 / length(Pc);
imagesc(fftshift(S.^0.1));
imagesc(S.^0.1);
colormap('default');

F(y1,:) = 0;
F(y2,:) = 0;
F(:,x1) = 0;
F(:,x2) = 0;

S = abs(F).^2 / length(Pc);
imagesc(S.^0.1);
imagesc(fftshift(S.^0.1));
colormap('default');

result = uint8(ifft2(F));
P_minVal = double(min(result(:)));
P_maxVal = double(max(result(:)));
subOperation = imsubtract(result, P_minVal);
result = immultiply(subOperation, 255 / (P_maxVal - P_minVal));
imshow(result);
```

Source code for removing the cage around the primate:

```
Pc = imread('../images/primate-caged.jpg');
Pc = rgb2gray(Pc);
imshow(Pc);

F = fft2(Pc);
S = abs(F).^2 / length(Pc);
imagesc(fftshift(S.^0.1));
imagesc(S.^0.1);
colormap('default');
y1 = 252;
x1 = 11;
y2 = 6;
x2 = 247;
y3 = 248;
x3 = 21;
y4 = 10;
```

```
x4 = 237;
y5 = 2;
x5 = 255;
y6 = 254;
x6 = 3;
F(y1-2 : y1+2, x1-2 : x1+2) = 0;
F(y2-2 : y2+2, x2-2 : x2+2) = 0;
F(y3-2 : y3+2, x3-2 : x3+2) = 0;
F(y4-2 : y4+2, x4-2 : x4+2) = 0;
F(y5-1 : y5+2, x5-2 : x5+1) = 0;
F(y6-2 : y6+2, x6-2 : x6+2) = 0;
S = abs(F).^2 / length(Pc);
imagesc(S.^0.1);
colormap('default');
result = uint8(ifft2(F));
result = real(result);
imshow(result);
```

Undoing Perspective Distortion of Planar Surface

a) Question:

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

Answer:

The image of the book can be shown using the following code:

```
Pc = imread('../images/book.jpg');
imshow(Pc);
```

It will result in the following image being shown:

b) Question:

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

>> [X Y] = ginput(4)

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

To use the ginput function to image coordinates and to specify the vectors x and y to indicate the four corners of the desired image the following code can be used:

```
[X, Y] = ginput(4);
xim = [0 210 210 0];
yim = [0 0 297 297];
```

The order in which the image will be captured will be clockwise starting from the top left corner. Using the ginput function will approximately return the following:

X = [1.425326678765880e+02; 3.084854809437386e+02; 2.566524500907441e+02; 2.714156079854831]
Y = [27.178765880217780; 46.343920145190566; 2.157813067150635e+02; 1.608992740471869e+02]

c) Question:

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

>> u = A \ v;

The above computes $u = A^{-1} v$, and you can convert the projective transformation parameters to the normal matrix form by

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

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

>> w = U*[X'; Y'; ones(1,4)];
>> w = w ./ (ones(3,1) * w(3,:))

Does the transformation give you back the 4 corners of the desired image?

Answer:

The following code can be used to find the matrices required to estimate the projective transformation parameters:

```
v = [xim(1); yim(1); xim(2); yim(2); xim(3); yim(3); xim(4); yim(4)];
A = [
[X(1), Y(1), 1, 0,    0,    0, -xim(1) * X(1), -xim(1) * Y(1)];
[0,    0,    0, X(1), Y(1), 1, -yim(1) * X(1), -yim(1) * Y(1)];
[X(2), Y(2), 1, 0,    0,    0, -xim(2) * X(2), -xim(2) * Y(2)];
[0,    0,    0, X(2), Y(2), 1, -yim(2) * X(2), -yim(2) * Y(2)];
[X(3), Y(3), 1, 0,    0,    0, -xim(3) * X(3), -xim(3) * Y(3)];
[0,    0,    0, X(3), Y(3), 1, -yim(3) * X(3), -yim(3) * Y(3)];
[X(4), Y(4), 1, 0,    0,    0, -xim(4) * X(4), -xim(4) * Y(4)];
[0,    0,    0, X(4), Y(4), 1, -yim(4) * X(4), -yim(4) * Y(4)];
];
```

Calculating the matrix u using the given matrix code

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

will result in the following matrix:

```
u =

    1.4380
    1.5035
 -245.8193
   -0.4168
    3.6092
  -38.6840
    0.0001
    0.0051
```

You can reshape the projective transformation parameters to the normal matrix form by using the following matrix code:

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

And the following output can be seen in Matlab:

```
U =

    1.4380    1.5035 -245.8193
   -0.4168    3.6092  -38.6840
    0.0001    0.0051    1.0000
```

The following matrix code can be used to verify that the Matrix U is correct:

```
w = U * [X'; Y'; ones(1,4)];
w = w ./ (ones(3,1) * w(3,:));
```

And the following output can be seen in Matlab:

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

The transformation returns the 4 corners of the desired image.

d) Question:

Warp the image via

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

Answer:

In order to follow the instruction above, the same code is entered in Matlab.

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

e) Question:

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

Answer:

The image can be shown using the following code:

```
imshow(P2);
```

It will result in the following image:

Figure 33: Image of the book that has been transformed

This image is exactly as expected. The quality is better for the parts that's closer to the camera. As a result the top part of the book is especially blurry, especially the top left part which is the furthest from the camera. The words on the top part of the book is especially hard to read and this is as expected as it is also hard to read from the original image.

f) Source code:

```
Pc = imread('../images/book.jpg');
imshow(Pc);
[X, Y] = ginput(4);
xim = [0 210 210 0];
yim = [0 0 297 297];
v = [xim(1); yim(1); xim(2); yim(2); xim(3); yim(3); xim(4); yim(4)];
A = [
    [X(1), Y(1), 1, 0,    0,    0, -xim(1) * X(1), -xim(1) * Y(1)];
    [0,    0,    0, X(1), Y(1), 1, -yim(1) * X(1), -yim(1) * Y(1)];
    [X(2), Y(2), 1, 0,    0,    0, -xim(2) * X(2), -xim(2) * Y(2)];
    [0,    0,    0, X(2), Y(2), 1, -yim(2) * X(2), -yim(2) * Y(2)];
    [X(3), Y(3), 1, 0,    0,    0, -xim(3) * X(3), -xim(3) * Y(3)];
    [0,    0,    0, X(3), Y(3), 1, -yim(3) * X(3), -yim(3) * Y(3)];
    [X(4), Y(4), 1, 0,    0,    0, -xim(4) * X(4), -xim(4) * Y(4)];
    [0,    0,    0, X(4), Y(4), 1, -yim(4) * X(4), -yim(4) * Y(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');
P2 = imtransform(Pc, T, 'XData', [0 210], 'YData', [0 297]);
imshow(P2);
```