Image Enhancement- CS3IA16

Alfie Sargent

BSc Robotics, jy013677@reading.ac.uk

ABSTRACT

The following report details an experiment in image enhancement techniques in MATLAB using methods in the spatial and frequency domain. The aim of the experiment was to develop a program that was capable of removing periodic and random noise from an image. The experiment was successful and the author managed to reduce the mean square error of the image but did result in the image being blurred. The techniques explored in this report detail how this is possible but also what improvements could be made to make the image even more reminiscent of the "perfect" initial image. A median and gaussian filter are used in the spatial domain to remove the random noise and a mask in the Fourier domain applied to remove the periodic noise.

1. INTRODUCTION

The disturbance created by the noisy image used can be split into three different categories: Gaussian noise created by poor illumination when the photo was taken, "salt and pepper noise" resulting from errors in the analogue to digital conversion of the image and periodic noise occurs due to electromechanical interference resulting in a repeatable pattern seen across the entirety of the image (Figure 1). Using the image processing toolbox libraries that MATLAB offers, the noisy image can be significantly enhanced. The following image enhancement techniques have been used to attempt to rectify these errors using filters in both the frequency and the spatial domain.

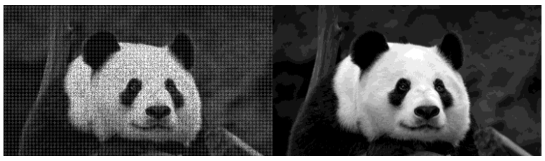


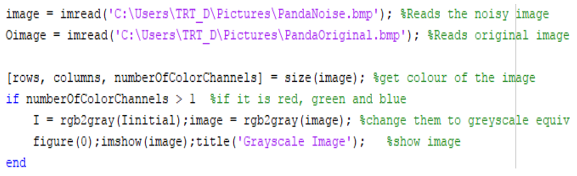
Figure 1: The left hand picture depicts the noisy and imperfect picture of the panda and the right picture depicts the actual "perfect" image that the experiment will be comparing the final results too. From the picture on the left, clear repeatable disturbances, noise on the fur and poor illumination of the general picture can be observed.

2. METHODOLOGY

2.1 Techniques used

Before the code was written, the image enhancement techniques were initially researched and thought about. The primary issue with the noisy picture was the periodic noise. To solve this issue, a band-stop filter was used to essentially remove certain select frequencies from the image. Furthermore, the image also contained a lot of "salt and pepper" noise which made the image fuzzy at certain frequencies. Upon further research, it was determined that the best way to resolve this issue was to use a combination of a median and Gaussian filter. The Gaussian filter does slightly blur the image but removes unwanted spikes and detail in the image. The median filter is very similar but uses a different kernel to the Gaussian filter (essentially the Gaussian filters kernel is shaped like a bell, this is the primary difference).

With the image enhancement techniques decided, the first thing to do was to convert the image into a format in which it could be easily manipulated and analysed. The images were imported into MATLAB and then each pixels location and RGB values were determined and placed into a matrix. The image was initially translated into grey scale just in case any of the pixels contained colour (Figure 2).



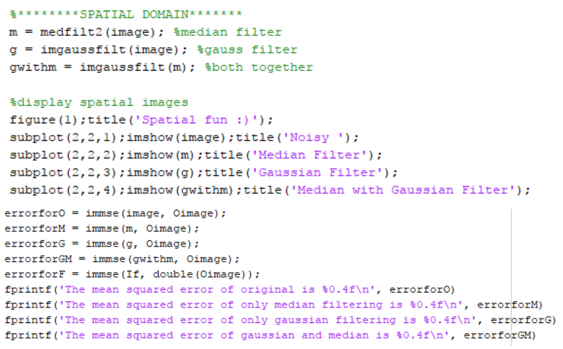
```
image = imread('C:\Users\TRT_D\Pictures\PandaNoise.bmp'); %Reads the noisy image
Oimage = imread('C:\Users\TRT_D\Pictures\PandaOriginal.bmp'); %Reads original image

[rows, columns, numberOfColorChannels] = size(image); %get colour of the image
if numberOfColorChannels > 1  %if it is red, green and blue
    I = rgb2gray(Iinitial);image = rgb2gray(image); %change them to greyscale equiv
    figure(0);imshow(image);title('Grayscale Image');   %show image
end
```

Figure 2: The code detailed above shows how, once the image was initially read in using the imread function, the pixels were sorted into a matrix determining their position in rows and columns and their RGB values. A greyscale conversion was done to the image and then displayed.

2.2 Spatial Domain

With the image now in a format capable of analysing, the image was enhanced firstly in the spatial domain. The image was passed through a median filter, a gaussian filter and then both; determining the mean square error for each filter applied using the imnse function in MATLAB (Figure 3). The image processing toolbox that MATLAB provides made this process relatively simple. With the filters applied, they were simply output as subplots with their mean square error also being displayed.



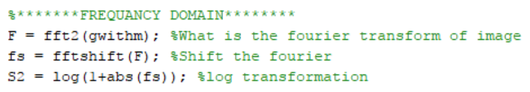
```
%********SPATIAL DOMAIN*******
m = medfilt2(image); %median filter
g = imgaussfilt(image); %gauss filter
gwithm = imgaussfilt(m); %both together

%display spatial images
figure(1);title('Spatial fun :)');
subplot(2,2,1);imshow(image);title('Noisy ');
subplot(2,2,2);imshow(m);title('Median Filter');
subplot(2,2,3);imshow(g);title('Gaussian Filter');
subplot(2,2,4);imshow(gwithm);title('Median with Gaussian Filter');
errorforO = immse(image, Oimage);
errorforM = immse(m, Oimage);
errorforG = immse(g, Oimage);
errorforGM = immse(gwithm, Oimage);
errorforF = immse(If, double(Oimage));
fprintf('The mean squared error of original is %0.4f\n', errorforO)
fprintf('The mean squared error of only median filtering is %0.4f\n', errorforM)
fprintf('The mean squared error of only gaussian filtering is %0.4f\n', errorforG)
fprintf('The mean squared error of gaussian and median is %0.4f\n', errorforGM)
```

Figure 3: This code illustrates how each of the spatial filters were applied and then displayed as sub plots in MATLAB. Furthermore, each filtered image has a mean square error calculated using MATLABS inbuilt image processing functions.

2.3 Frequency Domain

The Spatial domain filters now applied, the frequency domain was then analysed. The functions fft2 and ftshift are used to discover the Fourier transform of the image and then to shift the image (Figure 4). This made the image a little easier to analyse but then the absolute values were logged so that it could be evaluated even better.



```
%*******FREQUANCY DOMAIN********
F = fft2(gwithm); %What is the fourier transform of image
fs = fftshift(F); %Shift the fourier
S2 = log(1+abs(fs)); %log transformation
```

Figure 4: The code shown above illustrates how the initial noisy image was passed through a Fourier transform and then the absolute values logged so that peak recurring frequencies can be easily seen and analysed.

With the image now, a lot easier to analyse and consequently manipulate, a basic sketch was created to map the coordinates of each unwanted patterned spike in the image relative to the peaks shown in the logged image (Figure 5). Each spikes coordinate was noted and then used in the frequency enhancement function. The mapping of the coordinates was not the best method for enhancing the image; placing the points into a matrix and reflecting the pattern throughout the image would have been much more elegant and neater (refer to section 3).

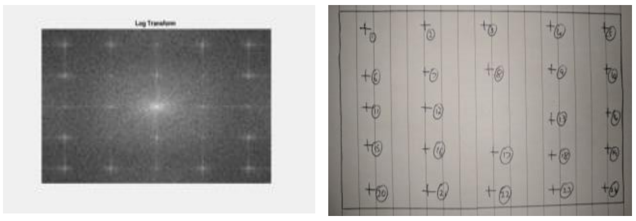
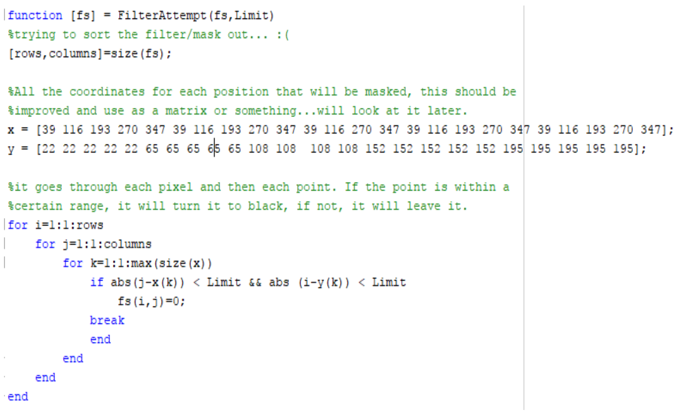


Figure 5: The images show how the log transformation makes it much easier to analyse where the periodic spikes in noise occur, the white crosses displayed. A crude sketch was made to map each point on paper and then each coordinate was mapped and written down so that a mask could be applied to them.

A new function was then created to create a mask for each of these spikes in frequency. Each spikes coordinate was placed into an array and then a nested for loop was used to cycle through each pixel and if the x and y coordinates match those that are stored in the array, a mask of a pre-determined radius was applied (Figure 6). The ideal radius was determined through trial and error to get the quality picture.



```
function [fs] = FilterAttempt(fs,Limit)
%trying to sort the filter/mask out... :(
[rows,columns]=size(fs);

%All the coordinates for each position that will be masked, this should be
%improved and use as a matrix or something...will look at it later.
x = [39 116 193 270 347 39 116 193 270 347 39 116 270 347 39 116 193 270 347 39 116 193 270 347];
y = [22 22 22 22 22 65 65 65 65 65 108 108  108 108 152 152 152 152 152 195 195 195 195 195];

%it goes through each pixel and then each point. If the point is within a
%certain range, it will turn it to black, if not, it will leave it.
for i=1:1:rows
    for j=1:1:columns
        for k=1:1:max(size(x))
            if abs(j-x(k)) < Limit && abs (i-y(k)) < Limit
                fs(i,j)=0;
            break
            end
        end
    end
end
```

Figure 6: The code shown sorts all of the coordinates of the unwanted spikes in frequency into two arrays, one for x and one for y. A nested for loop then cycles through each point in the picture and if both the x AND the y coordinates fall within the band, the mask is applied.

The result essentially blocks out these unwanted frequencies and therefore it removes the periodic noise from the image (Figure 7).

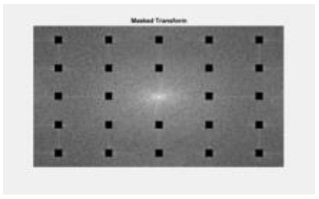


Figure 7: The image clearly shows how the mask has been applied. Each previously seen spike in frequency has been essentially blocked out.

Squares were initially used and their mean square error recorded. However, circles where also used to compare how they removed periodic noise (Figure 8).

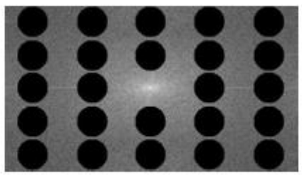
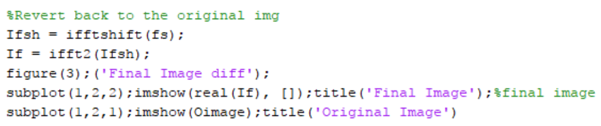


Figure 8: A circle mask used instead.

With the noise now removed, the mask filter was applied and the image inversely transformed and shifted to display the panda again (Figure 9) and the same function was used to determine the mean square error of the transformation.



```
%Revert back to the original img
Ifsh = ifftshift(fs);
If = ifft2(Ifsh);
figure(3);('Final Image diff');
subplot(1,2,2);imshow(real(If), []);title('Final Image');%final image
subplot(1,2,1);imshow(Oimage);title('Original Image')
```

Figure 9: This final part of the code simply reverts the image back into a legible format via an inverse Fourier transform. It is then displayed next to the original image for easier comparison.

With both the spatial and frequency domain image enhancement techniques applied to remove the random and periodic noise, the mean square error for each filter can be analysed.

3. RESULTS & DISCUSSION

All methods discussed in the previous section were used in an attempt to remove the periodic and random noise from the image. The noisy image and "perfect" image was compared and a mean square error of 875.2103 was found.

3.1 Spatial domain enhancement

To remove the random noise from the image both a median and gaussian filter was applied (Figure 10). The mean square error for each technique was calculated and the results recorded (Figure 11).

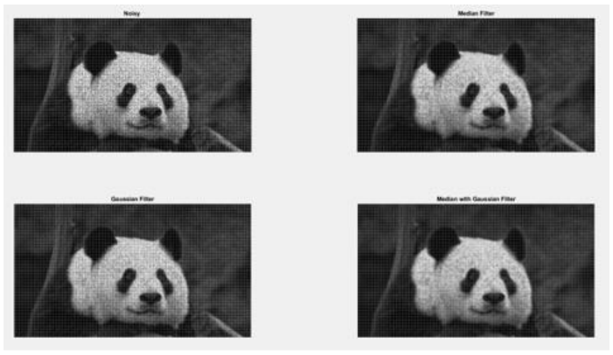
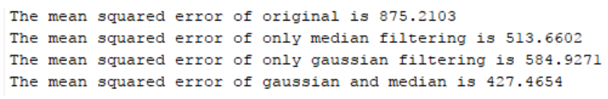


Figure 10: The image above shows the separate filters that were applied to the noisy image. The top left image is the initial noisy image, the top right the median filter, bottom right the gaussian filter and bottom left a combination of the two.



```
The mean squared error of original is 875.2103
The mean squared error of only median filtering is 513.6602
The mean squared error of only gaussian filtering is 584.9271
The mean squared error of gaussian and median is 427.4654
```

Figure 11: The results of the mean square error for each filter applied.

From observation, we can see that for all cases in which a filter has been used, the noise has been reduced but unfortunately the image has been blurred and has lost some of its edge. Possible edge enhancement could have been used to refine the image further and would be an improvement that would be made if the experiment were to be conducted again. From a visual analysis the median filter does seem to remove more noise than the gaussian filter but also blurs the image the most. The combination of the two removes the most noise but again, blurs the image the most.

From the results gathered, the observational analysis does match the statistical data. The mean square error for the median filter is higher than that of the gaussian filter but the combination of the two does reduce it to the smallest error. However, although the smallest mean square error is discovered by using a combination of the two filters, it does not necessarily mean that this is the best filter to use due to the blurring seen and could be improved via edge enhancement of the image.

3.2 Frequency Domain enhancement

Using the techniques previously discussed, a Fourier transformation was used on the noisy image and a mask used to overlay the spikes in unwanted frequency. Two different shapes were used, squares and circles to give different mean square values for the final image (Figures 12 and 13).

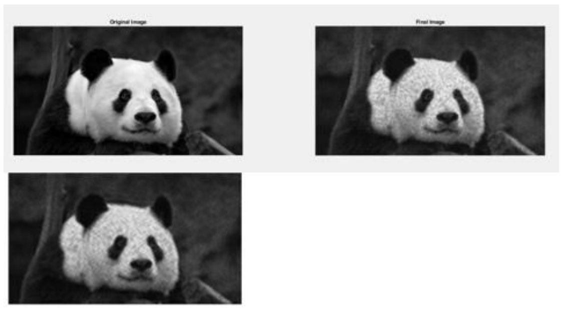
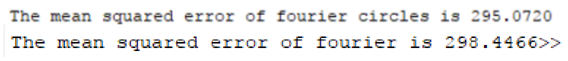


Figure 12: The top left image is the initial noisy image, the top right using a square mask and the bottom left using a circle mask.

From observation, the periodic noise has clearly been removed in both cases and their does not seem to be much difference just by looking at them. However, although the periodic noise has been removed, the image has once again become fairly blurry as a result. Two techniques that could be used to rectify this are: applying a deblurring process to the image or by changing and tweaking the radius of the mask used.



```
The mean squared error of fourier circles is 295.0720
The mean squared error of fourier is 298.4466>>
```

Figure 13: The mean square error for the circle mask (top) and the square mask (bottom).

With the mean square error results gathered, the difference in error is not substantial. As stated previously, although the mean square error has been reduced in comparison to the original image, that is not representative of the quality of the image due to the blurry nature of the filter. More image enhancement techniques need to be used to make the quality even closer to that of the original "perfect" image.

4. CONCLUSION

In conclusion, the image enhancement techniques that have been used do clearly show an increase in quality of the image and it does look closer to the original "perfect" image shown.

The spatial filtering that has been applied does remove a lot of the random noise from the image shown on the fur of the panda. However, it does blur the image significantly and consequently, a lot of the details on the edge of the image have been lost. A possible technique for improving this would be to use some form of edge enhancement to refine the image further. Furthermore, it would also be beneficial to experiment with some different low pass filters in an attempt to get the best possible image.

The frequency filtering used was very successful and did get rid of the periodic noise in the initial image. However, again, the image did become blurred and some deblurring techniques such as subspace analysis. Moreover, exploring different shapes to discover the best mask to use that produced the smallest mean square error is also something that should be explored further. Lastly, the code itself for the frequency domain function could be made more elegant due to the nature of the peaks shown. Only a few coordinates were needed due to them sharing a lot of the same x and y positions, they could have been reflected across the mask and therefore make the code much more efficient, adaptable and neater.

Overall, the experiment conducted was a success as the image enhancement techniques did reduce the mean square error of the image.

5. APPENDICES

Appendix 1; Main code

```
clear all; close all; clc
image =
imread('C:\Users\ASargent\Pictures
\PandaNoise.bmp'); %Reads the
noisy image
Oimage =
imread('C:\Users\ASargent\Pictures
\PandaOriginal.bmp'); %Reads
original image

[rows, columns,
numberOfColorChannels] =
size(image); %get colour of the
image
if numberOfColorChannels > 1  %if
it is red, green and blue
    I = rgb2gray(Iinitial);image =
rgb2gray(image); %change them to
greyscale equiv

figure(0);imshow(image);title('Gra
yscale Image');   %show image
end
m = medfilt2(image); %median
filter
```

```
g = imgaussfilt(image);
gwithm = imgaussfilt(m);

%display spatial images
figure(1);title('Spatial fun :)');
subplot(2,2,1);imshow(image);title
('Noisy ');
subplot(2,2,2);imshow(m);title('Me
dian Filter');
subplot(2,2,3);imshow(g);title('Ga
ussian Filter');
subplot(2,2,4);imshow(gwithm);titl
e('Median with Gaussian Filter');
F = fft2(gwithm); %What is the
fourier transform of image
fs = fftshift(F); %Shift the
fourier
S2 = log(1+abs(fs)); %log
transformation

fs=FilterAttempt(fs,6);
%Fourier Image
figure(2);
subplot(2,2,1);imshow(S2,
[]);title('Log Transform ');
subplot
(2,2,2);imshow(log(1+abs(fs)),[]);
title('Masked Transform');

%Revert back to the original img
Ifsh = ifftshift(fs);
If = ifft2(Ifsh);
figure(3);('Final Image diff');
subplot(1,2,2);imshow(real(If),
[]);title('Final Image');%final
image
subplot(1,2,1);imshow(Oimage);titl
e('Original Image')

%display all of the error values
of the images
errorforO = immse(image, Oimage);
errorforM = immse(m, Oimage);
errorforG = immse(g, Oimage);
errorforGM = immse(gwithm,
Oimage);
errorforF = immse(If,
double(Oimage));
fprintf('The mean squared error of
original is %0.4f\n', errorforO)
fprintf('The mean squared error of
only median filtering is %0.4f\n',
errorforM)
fprintf('The mean squared error of
only gaussian filtering is
%0.4f\n', errorforG)
fprintf('The mean squared error of
gaussian and median is %0.4f\n',
errorforGM)
fprintf('The mean squared error of
fourier is %0.4f', errorforF)
```

Appendix 2; Masked transformation (squares)

```
function [fs] =
FilterAttempt(fs,Limit)
%trying to sort the filter/mask
out... :(
[rows,columns]=size(fs);

%All the coordinates for each
position that will be masked, this
should be
%improved and use as a matrix or
something...will look at it later.
x = [39 116 193 270 347 39 116 193
270 347 39 116 270 347 39 116 193
270 347 39 116 193 270 347];
y = [22 22 22 22 22 65 65 65 65 65
108 108  108 108 152 152 152 152
152 195 195 195 195 195];

%it goes through each pixel and
then each point. If the point is
within a
%certain range, it will turn it to
black, if not, it will leave it.
for i=1:1:rows
    for j=1:1:columns
        for k=1:1:max(size(x))
            if abs(j-x(k)) < Limit
&& abs (i-y(k)) < Limit
                fs(i,j)=0;
            break
            end
        end
    end
end
```

Appendix 3; Masked transformation (circles)

```
function [fs] =
FilterAttempt(fs,Limit)
%trying to sort the filter/mask
out... :(
[rows,columns]=size(fs);

%All the coordinates for each
position that will be masked, this
should be
%improved and use as a matrix or
something...will look at it later.
x = [39 116 193 270 347 39 116 193
270 347 39 116 270 347 39 116 193
270 347 39 116 193 270 347];
```

```
y = [22 22 22 22 22 65 65 65 65 65
108 108  108 108 152 152 152 152
152 195 195 195 195 195];

%it goes through each pixel and
then each point. If the point is
within a
%certain range, it will turn it to
black, if not, it will leave it.
for i = 1:1:max(size(x)) %for each
spike
        t = 0:pi/20:2*pi; %for
plotting points of circle
             xcc =
Limit*cos(t)+x(i);
                ycc =
Limit*sin(t)+y(i);
 mask = poly2mask(double(xcc),
double(ycc), rows, columns);
 fs(mask) = 0; %sets mask to 0
end
end
```