Assignment 1 Report

Part 1

1.1 Matlab Function

```
function editedImage = gammaCorrection(originalImage, gammaValue)
    % Check if input gamma value is positive.
    if (gammaValue <= 0)
         error('Input gamma value cannot be non-positive.');
         editedImage = originalImage;
    else
       % Convert image from uint8 to double.
       originalImage = double(originalImage);
       % Modify input image according to given mapping function.
       editedImage = 255 * (originalImage / 255) .^ gammaValue;
       % Convert the image back to uint8.
       editedImage = uint8(editedImage);
    end
end
```

1.2 Use Different Gamma Values

When gamma is less than 1, the image looks brighter, closer to white. While gamma is greater than 1, the image looks darker, closer to black. If gamma equals 1, nothing changes. Following pictures are presented in ascending gamma values, showing the difference.

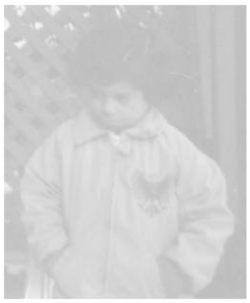
Figure 1.2.1 Gamma = 0.25

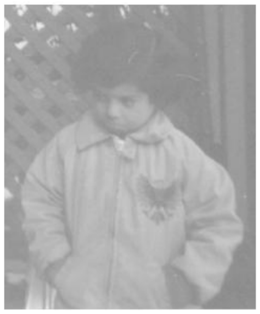
Figure 1.2.2 Gamma = 0.5

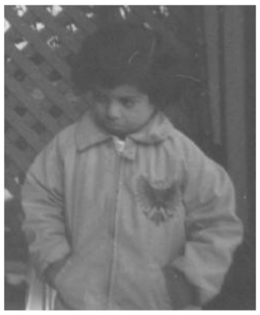
Figure 1.2.3 Gamma = 1

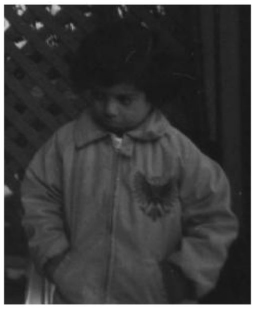
Figure 1.2.4 Gamma = 2

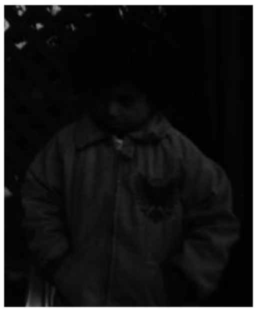
Figure 1.2.5 Gamma = 4

1.3 Image Enhancement

The original image is too dark, so that a gamma value less than 1 should be applied in order to recover the details from image. Different gamma values have been applied on the image, following pictures and histograms are represented in ascending order of gamma values.

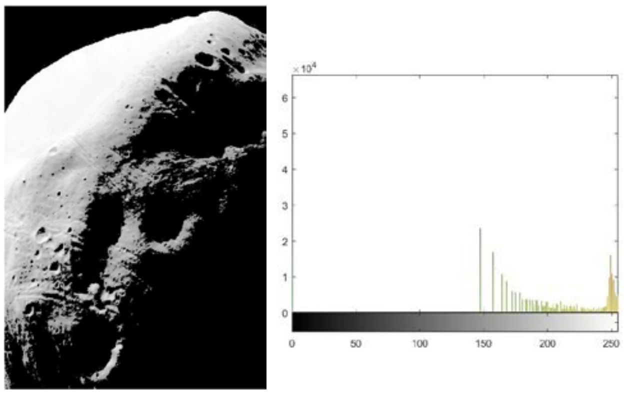

Figure 1.3.1 Gamma = 0.1

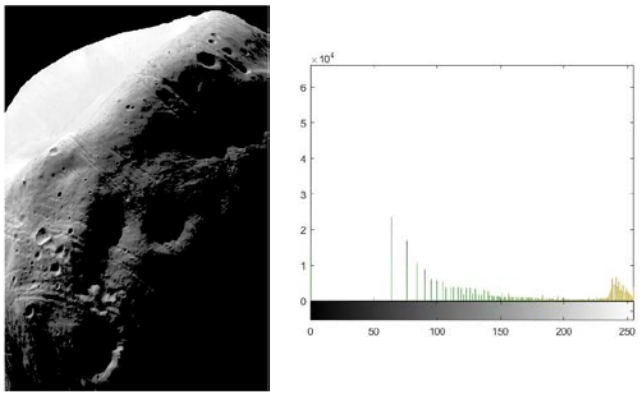

Figure 1.3.2 Gamma = 0.25

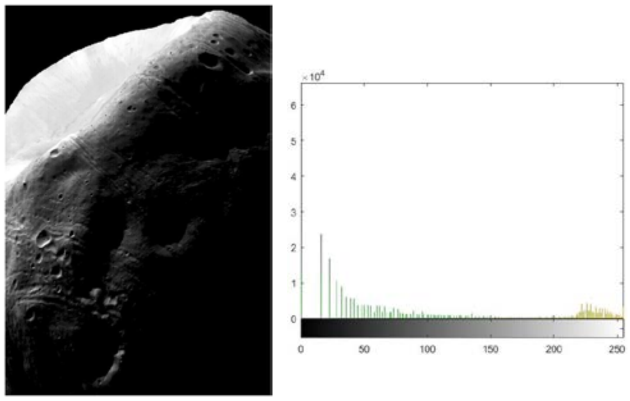

Figure 1.3.3 Gamma = 0.5

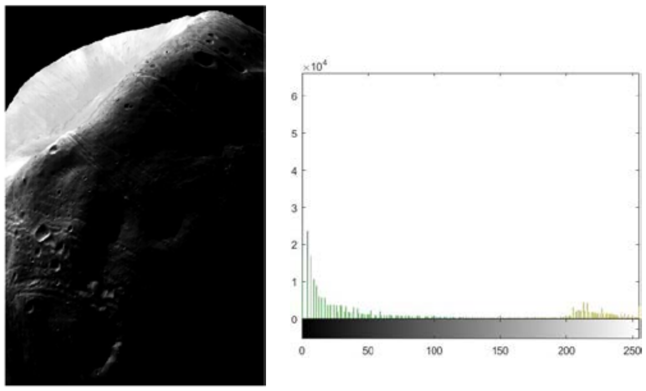

Figure 1.3.4 Gamma = 0.75

From the 4 images above, when gamma equals 0.25, the image contains the most details in both dark and bright parts.

The picture after applying histogram equalization function is attached below.

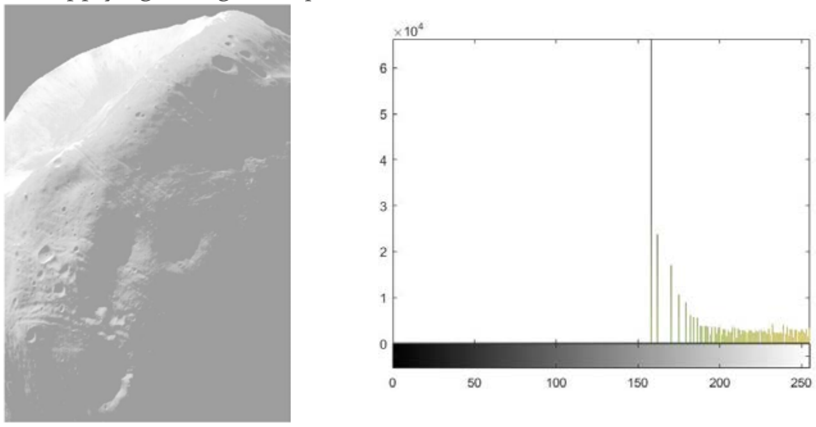

Figure 1.3.5 Results by function: histeq

Part 2

2.1 Matlab Function

```
function editedImage = sharpenImage(image, alpha)
    % Initialize Laplacian filter parameter
    laplacianFilter = [ 0 -0.25 0; -0.25 1 -0.25; 0 -0.25 0 ];
    image = double(image);
    % Get high frequency information by filtering the image
    highFreq = imfilter(image, laplacianFilter);
    % Enhance high frequency information
    editedImage = image + alpha * highFreq;
    % Convert image back to uint8
    editedImage = uint8(editedImage);
end
```

2.2 Sharpening a Normal Image

Several different alpha are tried, from -1 to 50, following pictures are represented in ascending order of alpha.

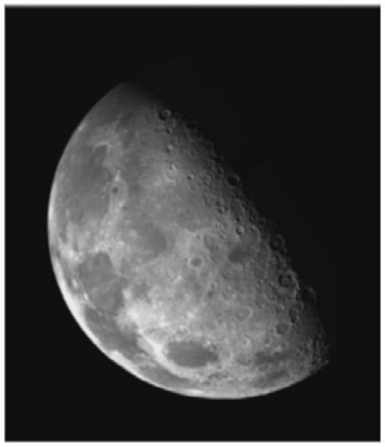
Figure 2.2.1 Alpha = -1

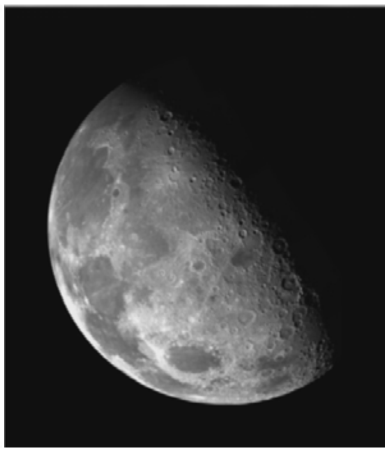
Figure 2.2.2 Original Image

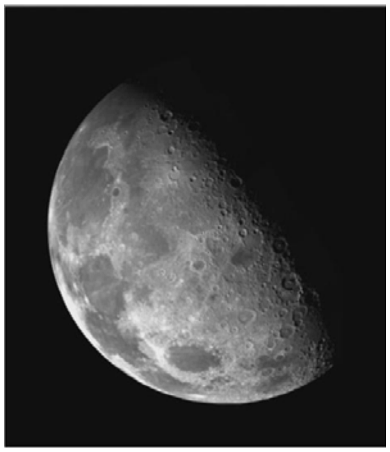
Figure 2.2.3 Alpha = 1

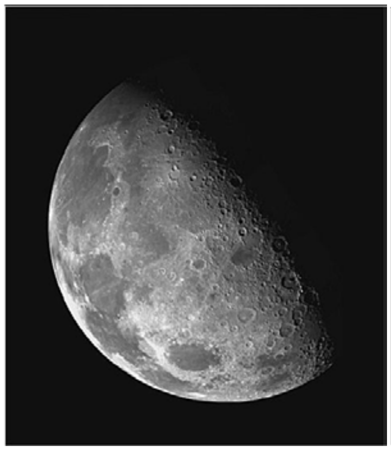
Figure 2.2.4 Alpha = 5

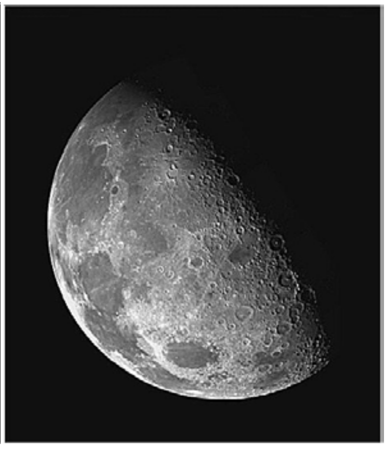
Figure 2.2.5 Alpha = 10

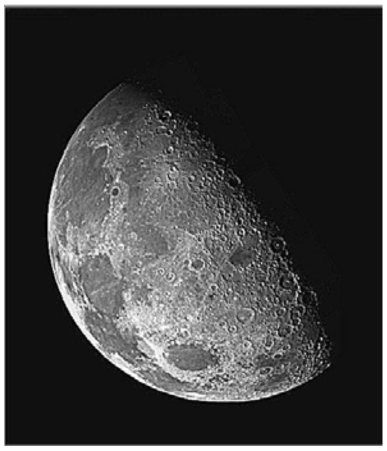
Figure 2.2.6 Alpha = 20

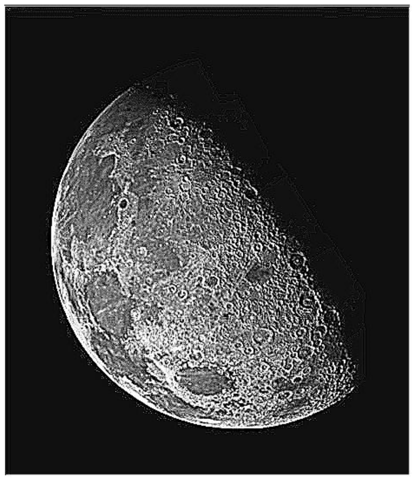
Figure 2.2.7 Alpha = 50

From the images above, we can find that a negative alpha may make an image blurrier, while a positive alpha may make an image sharper.

Among all the images, it can be concluded that an alpha around 5 may work best for this image. A lower value may not enhance edges of shapes enough, while a higher value may enhance edges so much that other details of the picture, like texture, color, etc., are affected. An unreasonably high alpha value may amplify unnoticeable high frequency noise so that it would become noticeable.

2.3 Sharpening a Blurry Image

In order to sharpen an unfocused image, several positive and even high value alpha should be introduced. Following 6 images are presented from alpha of 1 to 100.

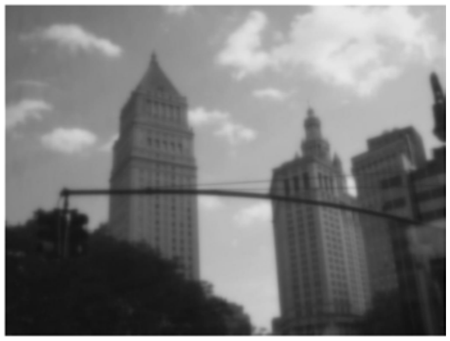
Figure 2.3.1 Alpha = 1

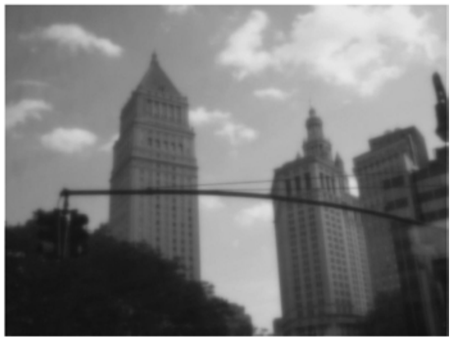
Figure 2.3.2 Alpha = 5

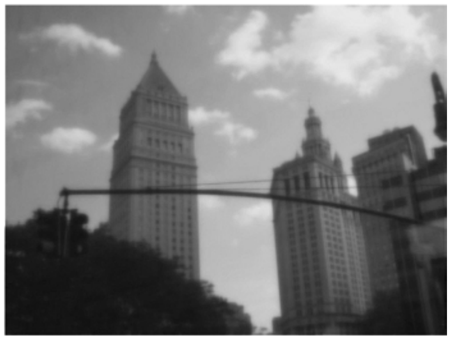
Figure 2.3.3 Alpha = 10

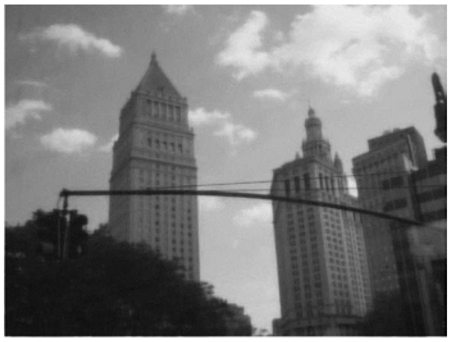
Figure 2.3.4 Alpha = 30

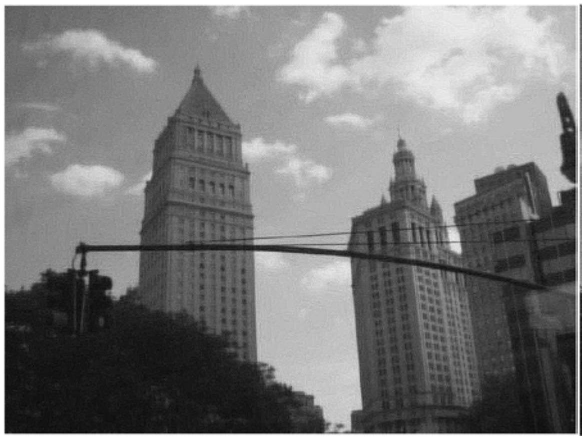
Figure 2.3.5 Alpha = 50

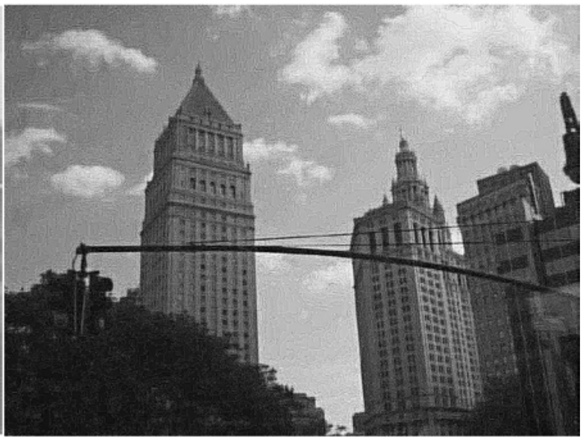
Figure 2.3.6 Alpha = 100

Although an extremely high alpha may help image looks sharper, it introduced high frequency noise. So that it is not possible to recover the scene from an unfocused image. An unfocused image cannot contain enough information, and existing information may also contain noise, if certain parts of information is amplified, noise included may also be amplified to affect image quality.

Part 3

3.1 Average & Median Filters

3.1.1 Matlab Script

```
image = imread('peppersNoise1.tiff');

% Average filter of 3 * 3
avgFilter3 = fspecial('average', 3);
avgFilteredImage3 = imfilter(image, avgFilter3);
figure, imshow(avgFilteredImage3);

% Average filter of 5 * 5
avgFilter5 = fspecial('average', 5);
avgFilteredImage5 = imfilter(image, avgFilter5);
figure, imshow(avgFilteredImage5);

% Median filter of 3 * 3
midFilteredImage3 = medfilt2(image, [3 3]);
figure, imshow(midFilteredImage3);

% Median filter of 5 * 5
midFilteredImage5 = medfilt2(image, [5 5]);
figure, imshow(midFilteredImage5);
```

3.1.2 Comparison of 2 Filters with Different Sizes

4 different filters are applied to the pepper image, average and median filters, with different sizes of 3 * 3 and 5 * 5 respectively. Output images are provided below.

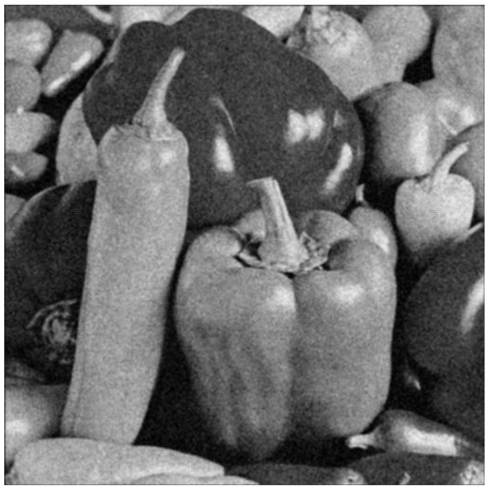
*Figure 3.1.2.1 Average Filter 3 * 3*

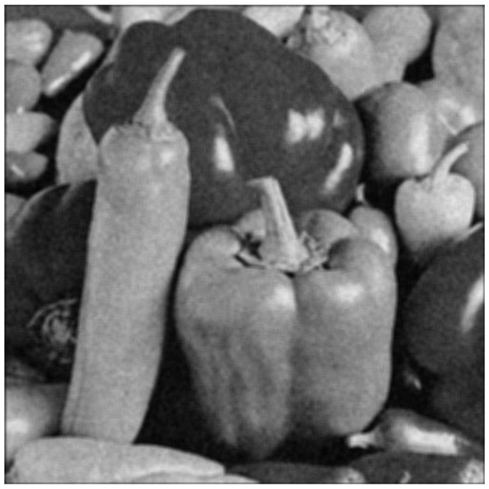
*Figure 3.1.2.2 Average Filter 5 * 5*

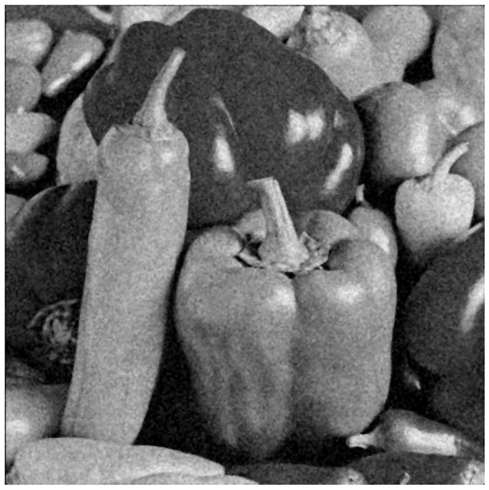
*Figure 3.1.2.3 Median Filter 3 * 3*

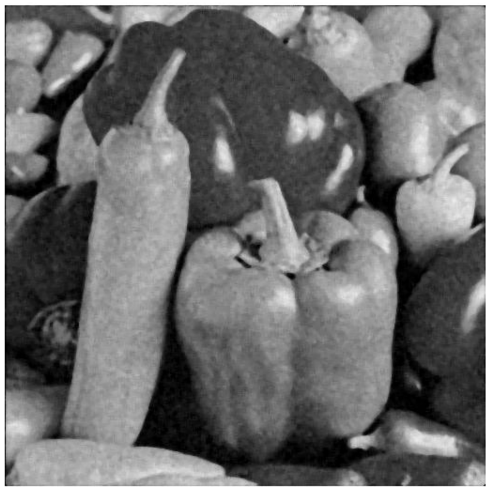
*Figure 3.1.2.4 Median Filter 5 * 5*

From 4 images above, it can be concluded that under the same filter size, average filter may generate output with less noise than that of median filter, while median filter preserves edges better so that output images look sharper than average filter outputs.

Under the same type of filter, a larger filter may remove noise better, however at the same time it may make the image blurrier.

3.2 Generating Edge Map

3.2.1 Matlab Script

```
% Create filters
Sx = fspecial('sobel');
Sy = Sx';
% Applying filter in both directions
```

```
sobelImageX = imfilter(double(image), Sx);
sobelImageY = imfilter(double(image), Sy);
% Calculate gradients
sobelImage = uint8(sqrt(sobelImageX .^2 + sobelImageY .^2));
for i = 1 : size(sobelImage, 1)
    for j = 1 : size(sobelImage, 2)
        if sobelImage(i, j) > 200
            sobelImage(i, j) = 255
        else
            sobelImage(i, j) = 0;
        end
    end
end
figure, imshow(sobelImage);
```

3.2.2 Versions of Edge Maps

Using the script above, 2 filtered images filtered by 3 * 3 average and median filters are processed to generate edge maps. The gradient values are converted to uint8 image, and I chose threshold of 200 to report edge components.

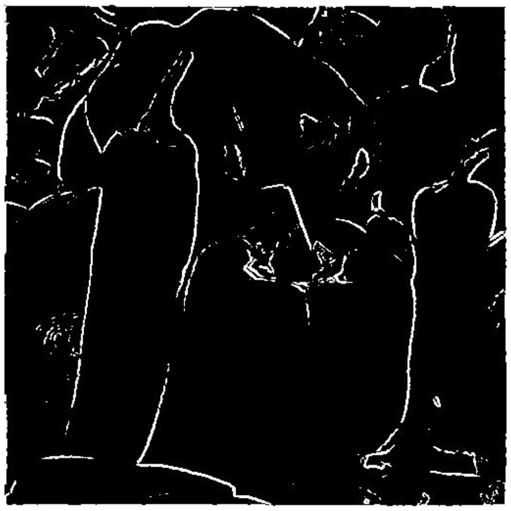

Figure 3.2.2.1 Average Filtered Edge Map

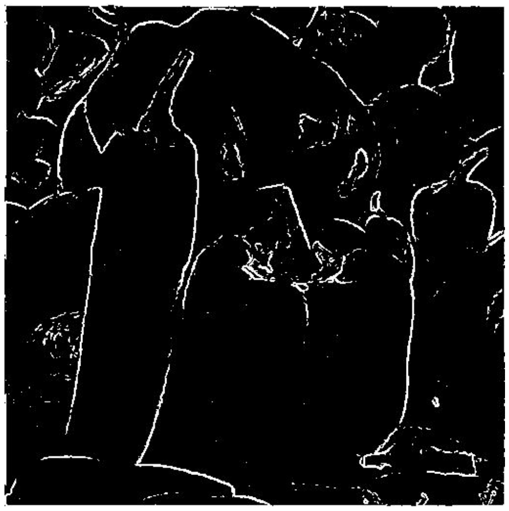

Figure 3.2.2.2 Median Filtered Edge Map

Median filtered images preserves edge information better, although it contains more noise than average filtered image. Since this section concentrates on edge preserving, median filter is better.

Part 4

4.1 Saving JPEG Files with Different Quality Factors

4.1.1 Matlab Function

```
function [psnr1, psnr2] = SaveJPGs()
    % Read 2 images.
    pepperImage = imread('peppers.tif');
    baboonImage = imread('baboon.tif');
    psnrPepper = ones(1, 5);
    % Saving image 1 with different quality factors
    imwrite(pepperImage, 'peppers90.jpg', 'jpg', 'Quality', 90);
    % Reading saved image and calculate PSNR between original image
    pepperImage90 = imread('peppers90.jpg');
    psnrPepper(1) = PSNRCal(pepperImage, pepperImage90);
    imwrite(pepperImage, 'peppers70.jpg', 'jpg', 'Quality', 70);
    pepperImage70 = imread('peppers70.jpg');
    psnrPepper(2) = PSNRCal(pepperImage, pepperImage70);
    imwrite(pepperImage, 'peppers50.jpg', 'jpg', 'Quality', 50);
    pepperImage50 = imread('peppers50.jpg');
    psnrPepper(3) = PSNRCal(pepperImage, pepperImage50);
    imwrite(pepperImage, 'peppers30.jpg', 'jpg', 'Quality', 30);
    pepperImage30 = imread('peppers30.jpg');
    psnrPepper(4) = PSNRCal(pepperImage, pepperImage30);
    imwrite(pepperImage, 'peppers10.jpg', 'jpg', 'Quality', 10);
    pepperImage(5) = imread('peppers10.jpg');
    psnrPepper10 = PSNRCal(pepperImage, pepperImage10);

    psnrBaboon = ones(1, 5);
    % Saving image 2 with different quality factors
    imwrite(baboonImage, 'baboon90.jpg', 'jpg', 'Quality', 90);
    baboonImage90 = imread('baboon90.jpg');
    psnrBaboon(1) = PSNRCal(baboonImage, baboonImage90);
    imwrite(baboonImage, 'baboon70.jpg', 'jpg', 'Quality', 70);
    baboonImage70 = imread('baboon70.jpg');
    psnrBaboon(2) = PSNRCal(baboonImage, baboonImage70);
    imwrite(baboonImage, 'baboon50.jpg', 'jpg', 'Quality', 50);
    baboonImage50 = imread('baboon50.jpg');
    psnrBaboon(3) = PSNRCal(baboonImage, baboonImage50);
    imwrite(baboonImage, 'baboon30.jpg', 'jpg', 'Quality', 30);
    baboonImage30 = imread('baboon30.jpg');
    psnrBaboon(4) = PSNRCal(baboonImage, baboonImage30);
    imwrite(baboonImage, 'baboon10.jpg', 'jpg', 'Quality', 10);
    baboonImage10 = imread('baboon10.jpg');
    psnrBaboon(5) = PSNRCal(baboonImage, baboonImage10);

    psnr1 = psnrPepper;
    psnr2 = psnrBaboon;

end

function psnr = PSNRCal(image1, image2)

    % Calculate PSNR
    if image1 == image2
        error('Input images must not be the same.');
    end

    diff = image1 - image2;
    psnr = 20 * log10(255/(sqrt(mean(mean(diff.^2)))));

end
```

4.1.2 Analysis of Images

PSNR table (in dB, larger is better):

Quality	90%	70%	50%	30%	10%
Pepper	53.6885	49.7581	39.5757	38.4988	34.8933
Baboon	40.3616	34.6076	33.1935	32.2288	30.7696

File size table (in KB):

Quality	90%	70%	50%	30%	10%
Pepper	48	33	27	16	9
Baboon	104	56	41	29	13

From the above chart, it can be concluded that the better quality of the image, the larger file size that the image has.

Distortion of JPEG images mainly lies in error when performing quantization. The less quality factor is introduced, a larger quantization step is applied, so that compressed image data would deviate farther from original values. Since larger quantization step is applied, compressed image would contain less details of luminance and color, so that distortion is introduced.

As my observation, when the quality factor drops to 30%, the distortion can be exactly noticeable, while it comes to 10%, it is strongly unacceptable.

4.2 My Own JPEG Encoder

4.2.1 Matlab Encoder Function

```
function fileLen = myOwnJPGEncoder(image, YTable)

    blockSize = 8;
    [row, col] = size(image);
    % Performing DCT by 8 * 8 block
    image = double(image) - 128;
    image = blkproc(image, [blockSize blockSize], 'dct2(x)');
    quaMatrix = YTable;
    % Performing quantization
    image = blkproc(image, [blockSize blockSize],
                                              'round(x./P1)', quaMatrix);

    newRow = row * col / (blockSize * blockSize);
    Y = ones(newRow, blockSize * blockSize);
    k = 1;
    % Zigzag image
    for i = 1 : blockSize : row
        for j = 1 : blockSize : col
            Y(k, :) = ZigzagMtx2Vector(image(i : i+blockSize-1,
                                                    j : j+blockSize-1));
            k = k + 1;
        end
    end

    % Entropy encoding
    fileLen = JPEG_entropy_encode(row, col, blockSize,
                                            YTable, Y, '', 1);

end
```

4.2.2 Matlab Decoder Function

```
function image = myOwnJPGDecoder()

    % Recover zigzag image from entropy decoder
    [row,col,blockSize,quaMatrix,zImage] = JPEG_entropy_decode('');
    image = ones(row, col);
    rowIndex = 1;
    colIndex = 1;
    % Recover image from zigzaged image
    for i = 1 : size(zImage, 1)
        image(rowIndex : rowIndex+blockSize-1,
                  colIndex : colIndex+blockSize-1) =
                                    Vector2ZigzagMtx(zImage(i, :));
        colIndex = colIndex + blockSize;
        if colIndex > col
            colIndex = 1;
            rowIndex = rowIndex + blockSize;
        end
    end

    % Performing invert quantization
    image = blkproc(image, [blockSize blockSize],
                                          'x .* P1', quaMatrix);
    % Performing invert DCT
    image = blkproc(image, [blockSize blockSize], 'idct2(x)');
    image = uint8(image + 128);

end
```

4.3 Applying My Own Encoder

4.3.1 Default Test on Standard JPEG Table

Using quantization factor of 1, and standard JPEG luminance quantization table, image file size is 24.8 KB, and PSNR is 39.6280 dB.

It is possible to achieve lower size and higher PSNR by modifying the JPEG quantization table. Since standard table is based on past experience, if quantization table can be thoroughly customized according to current image, the compression can be achieved more efficiently. Please refer to next section for more information on how to alternating values of the JPEG quantization table.

4.3.2 Customized JPEG Tables

By using an optimized luminance table in the following:

```
optiTable = [ 11  11  12  15  20  27   36   47;
              11  12  15  20  27  36   47   93;
              12  15  20  27  36  47   93   185;
              15  20  27  36  47  93   185  369;
              20  27  36  47  93  185  369  737;
              27  36  47  93  185 369  737  1473;
              36  47  93  185 369 737  1473 2945;
              47  93  185 369 737 1473 2945 5889 ];
```

The program achieved a PSNR with 40.7521 dB, with a file size of 21.8 KB.

Comparing to standard JPEG quantization table, with the same quantization factor, file size has decreased 3 KB, while the PSNR of compressed image increased by around 1 dB.