CompE565, Fall 2021
Homework 2: JPEG based Image Compression

Prepared by

Elena Pérez-Ródenas Martínez

E-mail: eperezrodenasm3836@sdsu.edu

Electrical and Computer Engineering

San Diego State University

Contents

List of Figures

1 Introduction

Image compression plays a really important role in multimedia communication systems due to the fact that it reduces irrelevance and redundancy of the images and makes it more efficient to transmit data as well as to store it.

The domain of this work is to learn some image compression techniques using the JPEG image of the first assignment such as encoding with 8x8 block DCT and quantization or decode with inverse quantization and reconstruction of the image.

Sometimes JPEG images are too heavy and we need a lot of space to store them. JPEG supports a maximum image size of 65,535×65,535 pixels, hence up to 4 gigapixels. And if we want to store a lot of them it can get very difficult to do it. That's why these techniques are really helpful and it's important to know them.

This report contains a procedural section in where the techniques are explained and with the code to implement them, another one with the results obtained and a final conclusion.

2 Procedural Section

2.1 Encoder

In this section we will need the 4:2:0 YCbCr component image of the first assignment. In order to do that we will repeat the steps we already did back then.

```
clc
clear all
close all
I=imread("C:\Users\Elena Pérez-Ródenas\Desktop\SDSU\MULTIMEDIA COMMUNICATION
    SYSTEMS\Flooded_house.jpg","jpg");
ycbcr=rgb2ycbcr(I);
Y=ycbcr(:,:,1);
Cb=ycbcr(:,:,2);
Cr=ycbcr(:,:,3);
Cb420=Cb(1:2:end, 1:2:end);
Cr420=Cr(1:2:end, 1:2:end);
```

2.1.1 8x8 block DCT transform coefficients

The key of the DCT 8x8 is that any pair of 8x8 blocks can be processed independently. This makes it possible to perform the block-wise transform in parallel. [1]

In order to compute it we will create a dct2 function and use the blockproc command [2] for each component (Y, Cb and Cr after subsampling) specifying the size 8x8 and the DCT function. The resulting images will be shown in the results section.

```
dct_function = @(block_struct) dct2(block_struct.data);
Y_dct = blockproc(Y,[8,8],dct_function)
Cb_dct = blockproc(Cb420,[8,8],dct_function,'PadPartialBlocks',true)
Cr_dct = blockproc(Cr420,[8,8],dct_function,'PadPartialBlocks',true)
figure(1)
subplot(2,2,1), imshow(Y_dct), title("Y after 2D-DCT")
subplot(2,2,2), imshow(Cb_dct), title("Cb420 after 2D-DCT")
subplot(2,2,3), imshow(Cr_dct), title("Cr420 after 2D-DCT")
```

To get the first two blocks in the 6th row from top of the luminance component we will set the indexes (41:48, 1:8) for the first and (41:48, 9:16) for the second one since each block is 8x8 and 6x8=48.

```
fprintf("Coefficient matrix of the first DCT transformed image block\n")
block1R6 = Y_dct(41:48,1:8)
figure(2)
imshow(block1R6), title("First DCT transformed image block")
fprintf("Coefficient matrix of the second DCT transformed image block\ns")
block2R6 = Y_dct(41:48,9:16)
figure(3)
imshow(block2R6), title("Second DCT transformed image block")
```

2.1.2 Quantization of the DCT image using quantizer matrixes

We will use the Luma and Chroma JPEG quantization matrixes [5] and create one function for each with the correct size to use the blockproc command. After that, we will get the two blocks again as we did in the previous section.

```
LumQMatrix = [16 11 10 16 24 40 51 61;
              12 12 14 19 26 58 60 55;
              14 13 16 24 40 57 69 56;
              14 17 22 29 51 87 80 62;
              18 22 37 56 68 109 103 77;
              24 35 55 64 81 104 113 92;
              49 64 78 87 103 121 120 101;
              72 92 95 98 112 100 103 99];
ChromQMatrix = [17 18 24 47 99 99 99 99;
                18 21 26 66 99 99 99 99;
                24 26 56 99 99 99 99 99;
                47 66 99 99 99 99 99 99;
                99 99 99 99 99 99 99 99;
                99 99 99 99 99 99 99 99;
                99 99 99 99 99 99 99 99;
                99 99 99 99 99 99 99 99];
Quant_Luma = @(block_struct) round(block_struct.data./LumQMatrix);
```

```
Quant_Chroma = @(block_struct) round(block_struct.data./ChromQMatrix);
Y_Q = blockproc(Y_dct, [8 8], Quant_Luma);
Cb_Q = blockproc(Cb_dct, [8 8], Quant_Chroma);
Cr_Q = blockproc(Cr_dct, [8 8], Quant_Chroma);

Block1_Q = Y_Q(41:48,1:8);
figure(4)
imshow(Block1_Q), title("First block of the 6th row after quantization")
Block2_Q = Y_Q(41:48,9:16);
figure(5)
imshow(Block2_Q), title("Second block of the 6th row after quantization")
```

We are asked to show the DC DCT coefficient of the two blocks so we will just print them like this:

```
fprintf("The DC coefficient for the first 8x8 block of Y is %d\n", Block1_Q(1:1))
fprintf("The DC coefficient for the second 8x8 block of Y is %d\n", Block2_Q(1:1))
```

We are also asked to show the Zigzag scanned AC DCT coefficients so we have to create a zigzag function and then obtain the AC DCT coefficients.

We will go through each position in the matrix following the zigzag order. r will be the current row and c the current column while v will be our new matrix with the zigzag order.

```
function v = zigzag2d(M)
    N = length(M(:,1));
    fmax = [ (1:N-1) (N*ones(1,N)) ];
    fmin = [ ones(1,N) (2:N) ];
    k = 0;
    v = [];
    for u = 2:N+N
        for r = fmin(u-1):fmax(u-1)
            c = u-r;
            k = k+1;
            if rem(u,2) == 0,
                v(k) = M(r,c);
            else
                v(k) = M(c,r);
            end
        end
    end
v = v';
```

[3]

Finally, we will print the AC DCT coefficients after applying the zigzag function.

```
Block1_zz = zigzag2d(Block1_Q);
Block2_zz = zigzag2d(Block2_Q);
AC_block1 = Block1_zz(2:end)
AC_block2 = Block2_zz(2:end)
```

2.2 Decoder

2.2.1 Inversed quantized images

In this section we will use the images obtained in section 2.1.2 and compute the inverse quantization. We will create a block structure and now instead of dividing by the Luma and Chroma quantization matrixes we will multiply them to the data of the block structure and then use the blockproc function as before.

```
InvQ_Luma = @(block_struct) round(block_struct.data*LumQMatrix);
InvQ_Chroma = @(block_struct) round(block_struct.data*ChromQMatrix);
Y_invQ = blockproc(Y_Q, [8 8], InvQ_Luma);
Cb_invQ = blockproc(Cb_Q, [8 8], InvQ_Chroma);
Cr_invQ = blockproc(Cr_Q, [8 8], InvQ_Chroma);
figure(6)
subplot(2,2,1), imshow(Y_invQ), title("Y after inversed quantization")
subplot(2,2,2), imshow(Cb_invQ), title("Cb after inversed quantization")
subplot(2,2,3), imshow(Cr_invQ), title("Cr after inversed quantization")
```

2.2.2 Reconstruction with DCT coefficients

First we will construct a IDCT function and use blockproc to get the 3 components.

```
IDCT_function = @(block_struct) idct2(block_struct.data);
Y_IDCT = blockproc(Y_invQ, [8 8], IDCT_function);
Cb_IDCT = blockproc(Cb_invQ, [8 8], IDCT_function);
Cr_IDCT = blockproc(Cr_invQ, [8 8], IDCT_function);
```

Then, since we started with a subsampled image we will recover the original one upsampling it.

```
Cb_new = Cb;
Cb_new(2:2:end, 2:2:end) = Cb_new(1:2:end,1:2:end);
Cr_new = Cr;
Cr_new(2:2:end, 2:2:end) = Cr_new(1:2:end,1:2:end);
```

Finally, we will create a new image with the 3 YCbCr components and use the function ycbcr2rgb to recover the RGB image.

```
NEW = cat(3,Y,Cb_new,Cr_new);
RGB_NEW = ycbcr2rgb(NEW);
figure(7)
imshow(RGB_NEW), title("Reconstructed RGB image")
```

2.2.3 Error Image

To calculate the error image we just have to substract the Luminance component and the one after IDCT. Then, we can show it setting the second parameter as [0 max(Image_Error(:))]. This command like that displays the grayscale image, specifying the display range as a two-element vector, [low high]. [4]

```
Image_Error = abs(double(Y) - Y_IDCT);
figure(8)
imshow(Image_Error,[0 max(Image_Error(:))]), title("Error Image")
```

2.2.4 PSNR for the luminance component of the decoded image

First we calculate the MSE taking the mean of the squared errors and then we obtain the PSNR with the formula $PSNR = 10 \cdot log_{10}(\frac{(2^n-1)^2}{MSE})$

```
MSE = mean(Image_Error(:).^2);
PSNR = 10*log10(255.^2/MSE)
```

3 Results

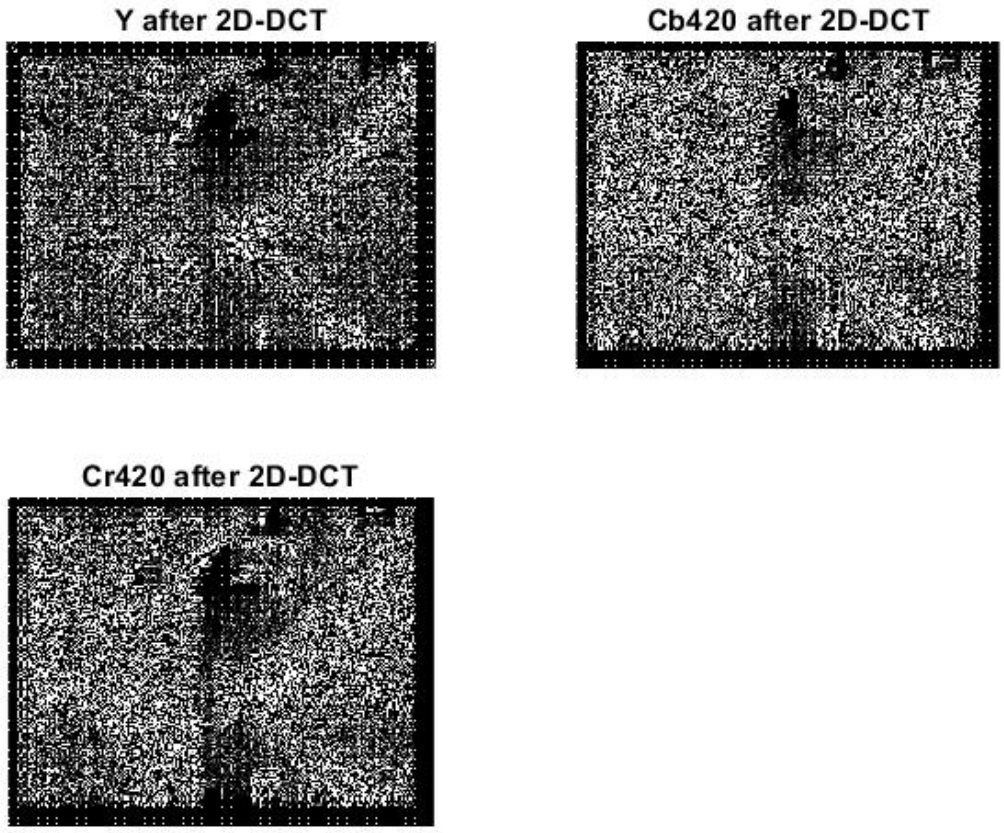
In this section the results of the commands in the procedural section will be shown and discussed.

3.1 Encoder

3.1.1 8x8 block DCT transform coefficients

After computing the transformation these are the images obtained:

Figure 1: YCbCr components after 8x8 block DCT transform



And the image of the block two blocks are the following:

Figure 2: First DCT transformed image block

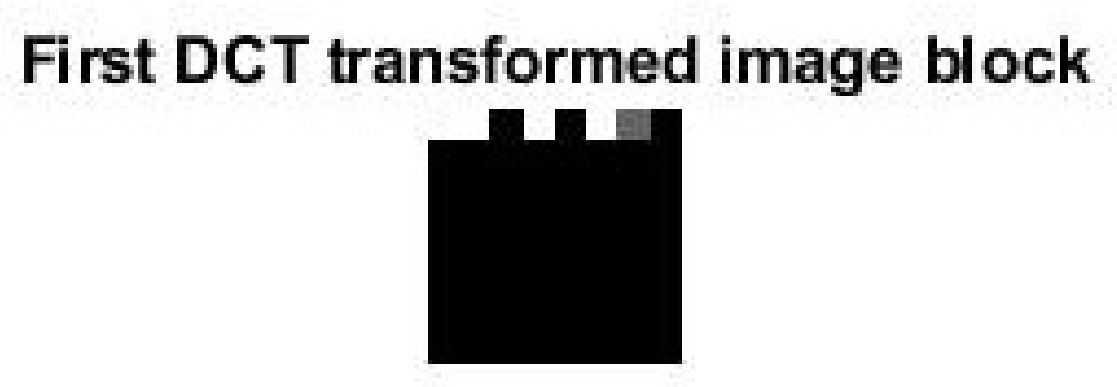
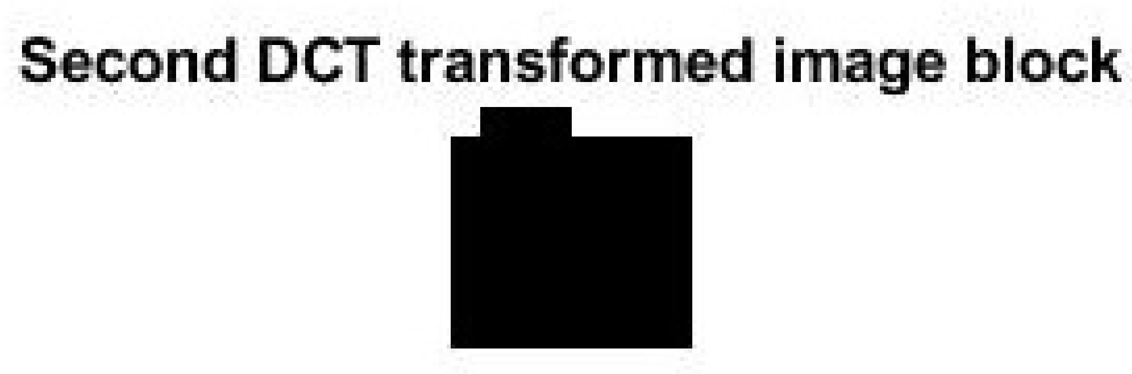


Figure 3: Second DCT transformed image block



We can't really check in the picture why they are almost completely black but if we made zoom and check the 2 first blocks of the 6th row they should match.

Also, these are the coefficient matrixes obtained:

Coefficient matrix of the first DCT transformed image block
block1R6 =
1.0e+03 *
1.6880 0.1472 -0.0323 0.0142 -0.0060 0.0043 0.0004 -0.0010 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Coefficient matrix of the second DCT transformed image block s block2R6 =
1.0e+03 *
1.7120 -0.2099 -0.1457 -0.0611 0.0180 0.0669 0.0820 0.0525 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3.1.2 Quantization of the DCT image using quantizer matrixes

After applying quantization to the different components we get the first and second blocks of the 6th row and print them:

Figure 4: First block of the 6th row after quantization

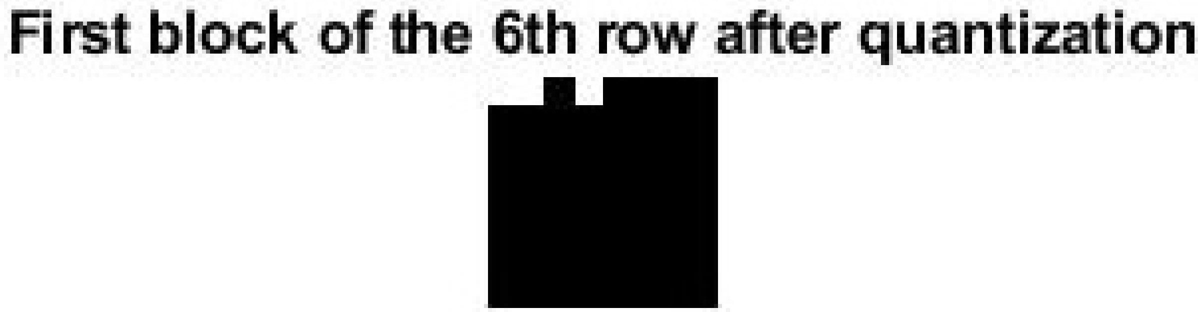
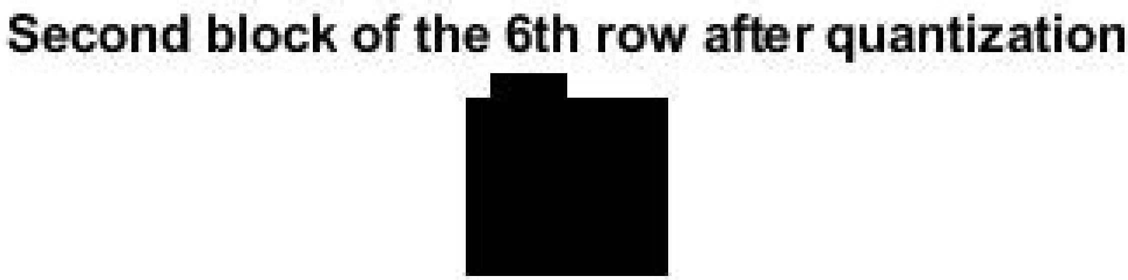


Figure 5: Second block of the 6th row after quantization



When we print the DC DCT coefficients we get:

The DC coefficient for the first 8x8 block of Y is 105
The DC coefficient for the second 8x8 block of Y is 107

And after applying the zigzag function and getting the AC DCT coefficients we get:

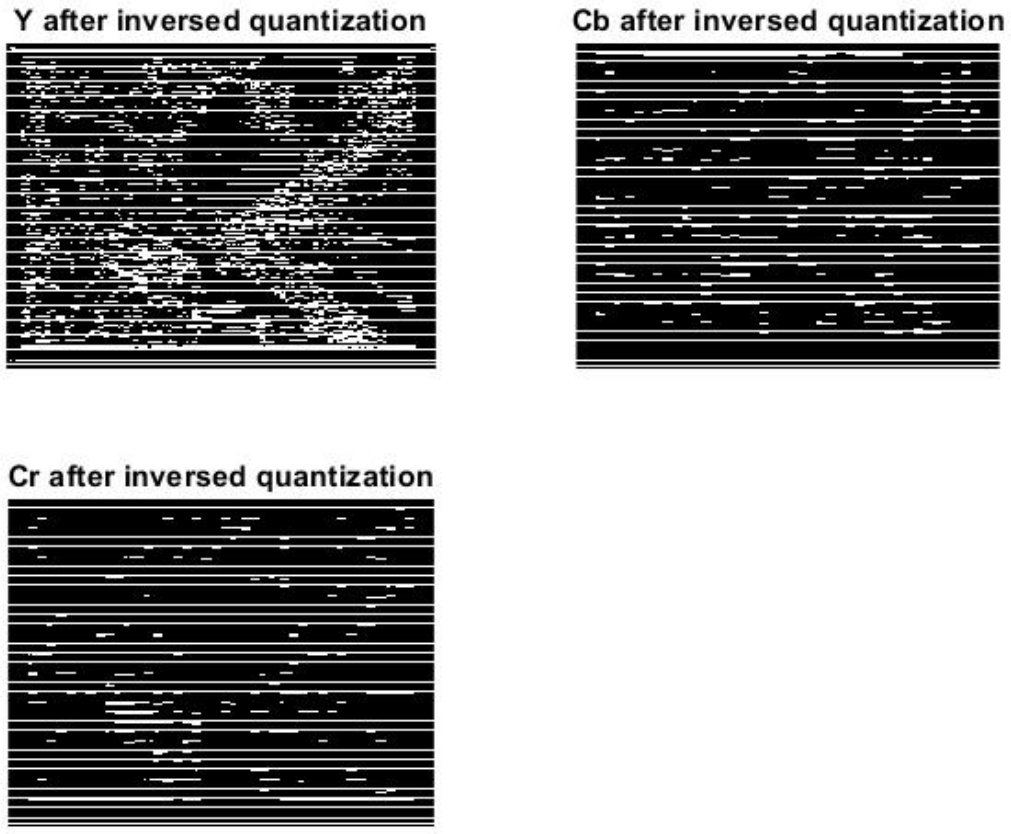
AC_block1 =
0 13 -3 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
AC_block2 =
0 -19 -15 0 0 0 0 0 -4 1 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Which makes sense because most of the pixels are black and it's represented by 0.

3.2 Decoder

3.2.1 Inversed quantized images

These are the images shown after inversed quantization:

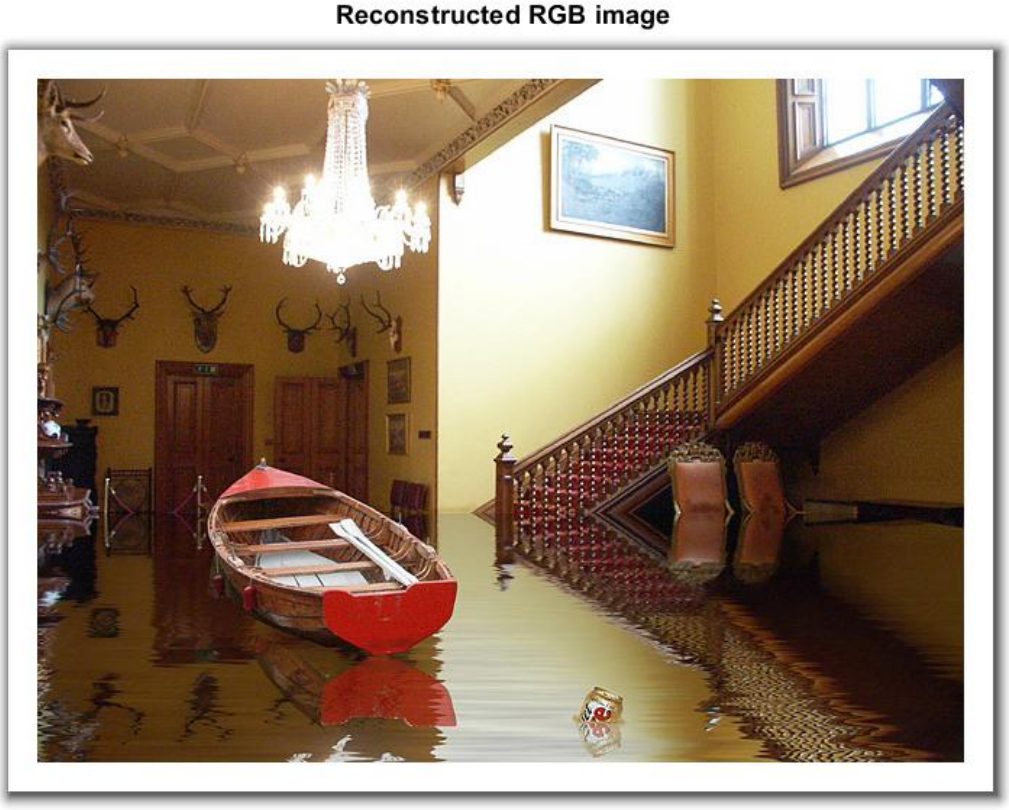
Figure 6: Original image



3.2.2 Reconstruction with DCT coefficients

This is the reconstructed image after computing inverse DCT coefficients.

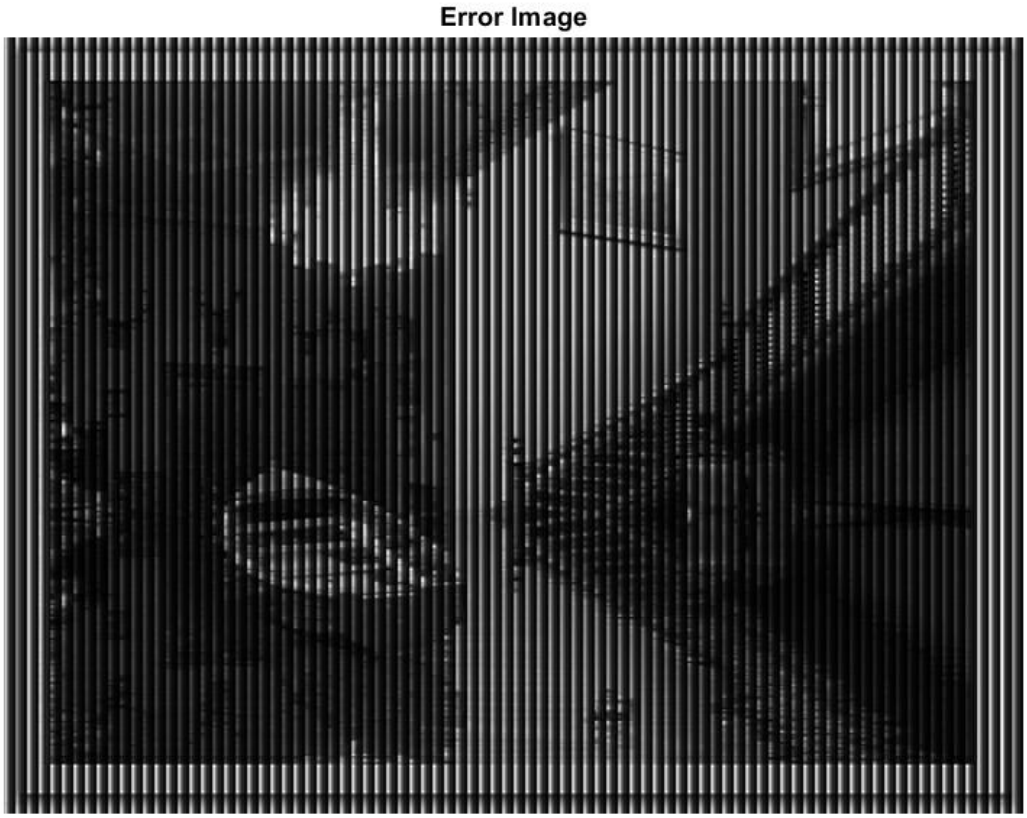
Figure 7: Reconstructed RGB image



As we can see, we can't hardly tell the difference between the original and the reconstructed one. This is because the process is very efficient and achieves a decent image but much more compressed.

3.2.3 Error Image

Figure 8: Error Image



This image represents the error of the reconstructed image and the original one in the Y component.

3.2.4 PSNR for the luminance component of the decoded image

These are the results obtained:

MSE = 7.8009e+05
PSNR = -10.7906

As we can see, the MSE is quite big and the PSNR is quite small. This makes sense because when MSE get's bigger, PSNR gets smaller due to the formula.

4 Conclusion

To sum up, with this assignment I have learned the imoportance of image compression in JPEG images getting to know how to encode and decode them making use of DCT and quantization

mainly.

It as helped me to understand better some of the commands used such as blockproc and also to make clearer some concepts seen in the theory class.

The most difficult thing for me was creating the zigzag function and reconstructing the image. At first, I thought that I didn't need to use blockproc or upsample the image again and it took me a while to understand how to do it.

From my point of view this assignment was really interesting as well as useful not only to learn the concepts but also to get better at coding in Matlab.

5 References

[1] https://www.google.com/url?sa=trct=jq=esrc=ssource=webcd=ved=2ahUKEwjG3e3lxOjzAhXuGDQIHfVHDsgQFnoECAgQAQurl=https

[2] https://www.mathworks.com/help/images/block-processing-large-images.html.

[3] https://www.google.com/url?sa=trct=jq=esrc=ssource=webcd=ved=2ahUKEwj1nqer7ebzAhVGs54KHZASByUQFnoECBcQAQurl=https

[4] https://www.mathworks.com/help/images/ref/imshow.html.

[5] COMPE-565. *Image Compression (JPEG).*