



Final Project

Team Members

Full Name	Sec.	BN	Role in the project
Adel Mohamed Abd-Elhmid Rizq	1	33	Task1
Ahmed Ashraf Hamdy Ahmed	1	2	Task2
Abdullah Ahmed Hemdan Ahmed	2	1	Task3 (Encoder a-b-f)
Ahmed Mohamed Mohamed Mahmoud Mahboub	1	7	Task3 (Decoder d-e-c)



Table of contents:

Project Tasks:	3
<i>Task One: Echo generation and removal</i>	3
<i>Task Two: Audio steganography</i>	4
<i>Task Three: Image compression</i>	5
Appendices	10
<i>Appendix A: Codes for Task One</i>	10
<i>Appendix B: Codes for Task Two</i>	12
<i>Appendix C: Codes for Task Three</i>	13
References:	17

List of Figures

Figure 1: Impulse response	3
Figure 2: Magnitude spectrum	4
Figure 3: Original image	6
Figure 4: Red component for original image	6
Figure 5: Green component for original image	7
Figure 6: Blue component for original image	7
Figure 7: Decopressed image with $m = 1$	8
Figure 8: Decopressed image with $m = 2$	8
Figure 9: Decopressed image with $m = 3$	9
Figure 10: Decopressed image with $m=4$	9



Project Tasks:

Task One: Echo generation and removal

All the required results and answers to questions.

All the required figures.

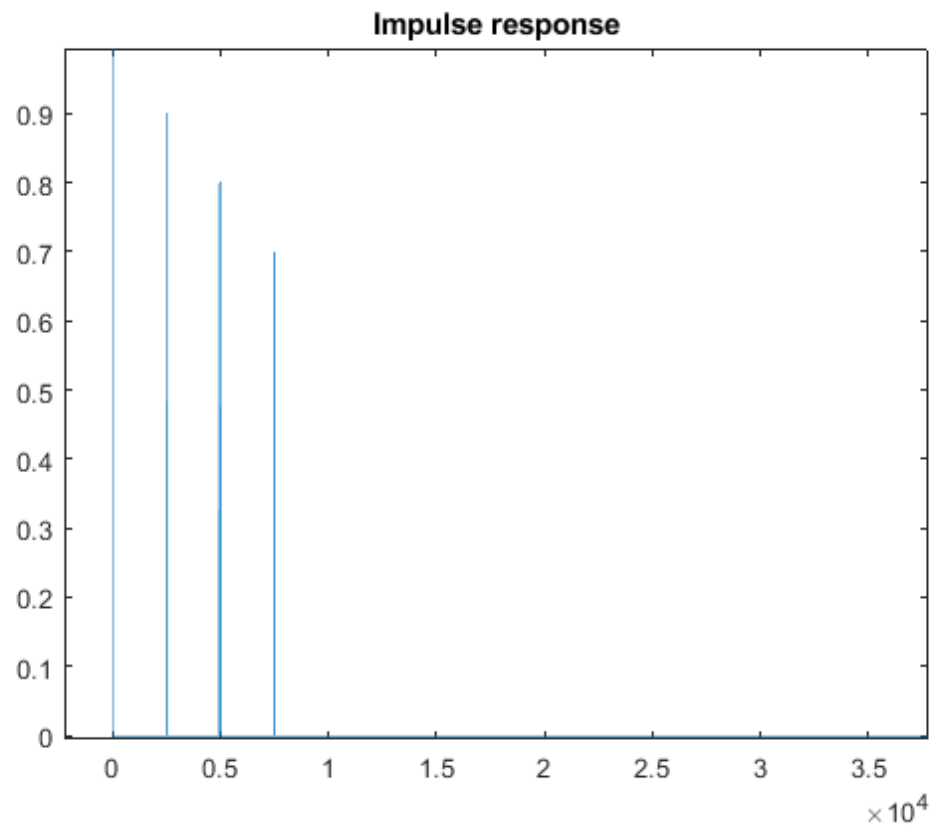


Figure 1: Impulse response



Task Two: Audio steganography

All the required figures.

- The magnitude Spectrum of $X[n]$

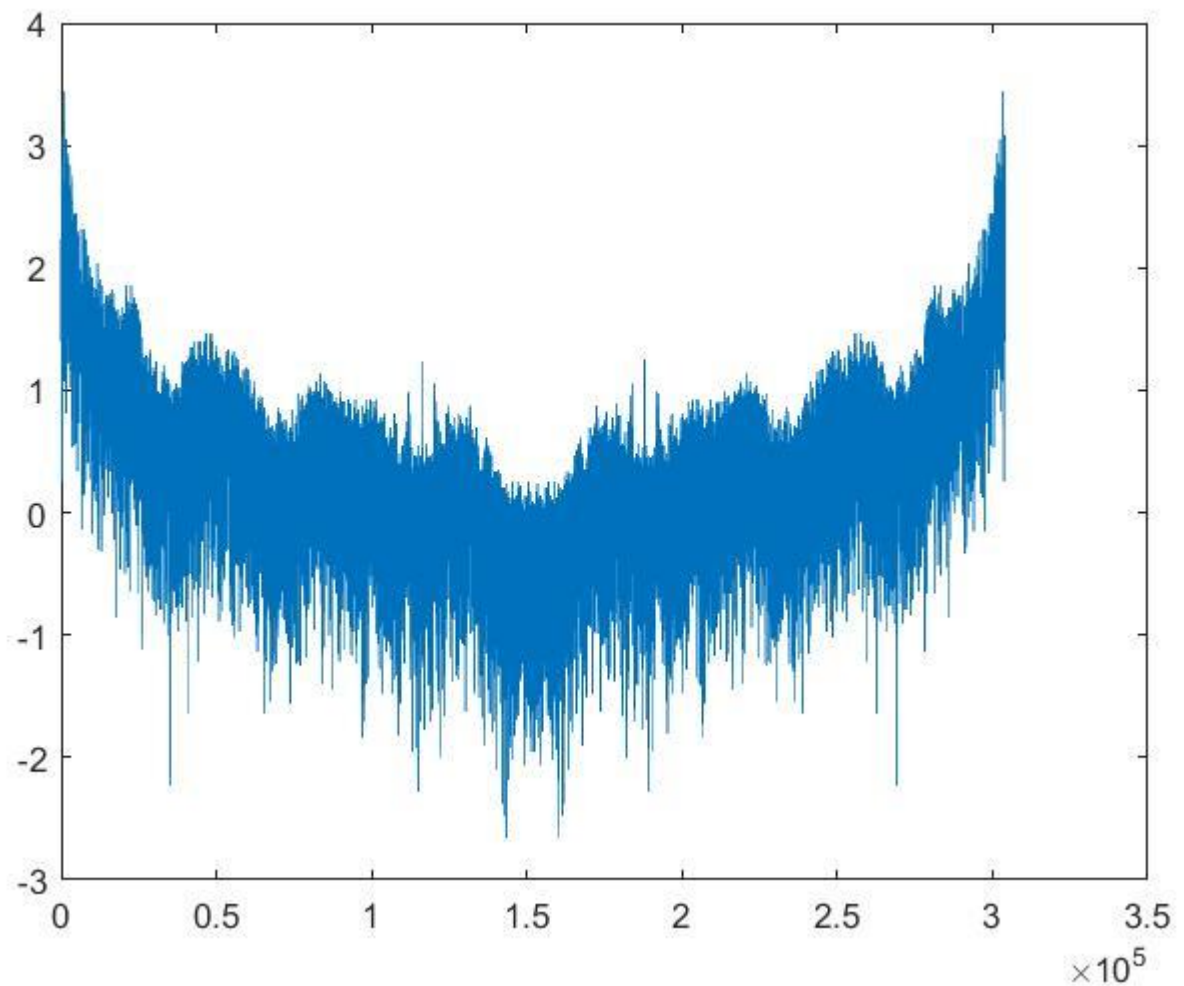


Figure 2: magnitude Spectrum of $X[n]$



Task Three: Image compression

C)

Original image size = 6,220,928 bytes

M	size
1	194,528 bytes
2	777,728 bytes
3	1,749,728 bytes
4	3,110,528 bytes

E)

M	PSNR
1	31.515999113230823
2	32.096097015340824
3	32.59529600923095
4	33.25038629758039

- F)
1. The main advantage of DCT over DFT is that DCT has a very high degree of spectral compaction at the qualitative level.
 2. DCT signal's representation intends to have most of its energy concentrated in a very small number of coefficients for natural photographic images so that the other coefficients having a small amplitude may be discarded easily when compared to DFT.
 3. In addition to that DCT calculations are much faster than DFT since complicated and complex calculations are avoided.



Figure3: Original image

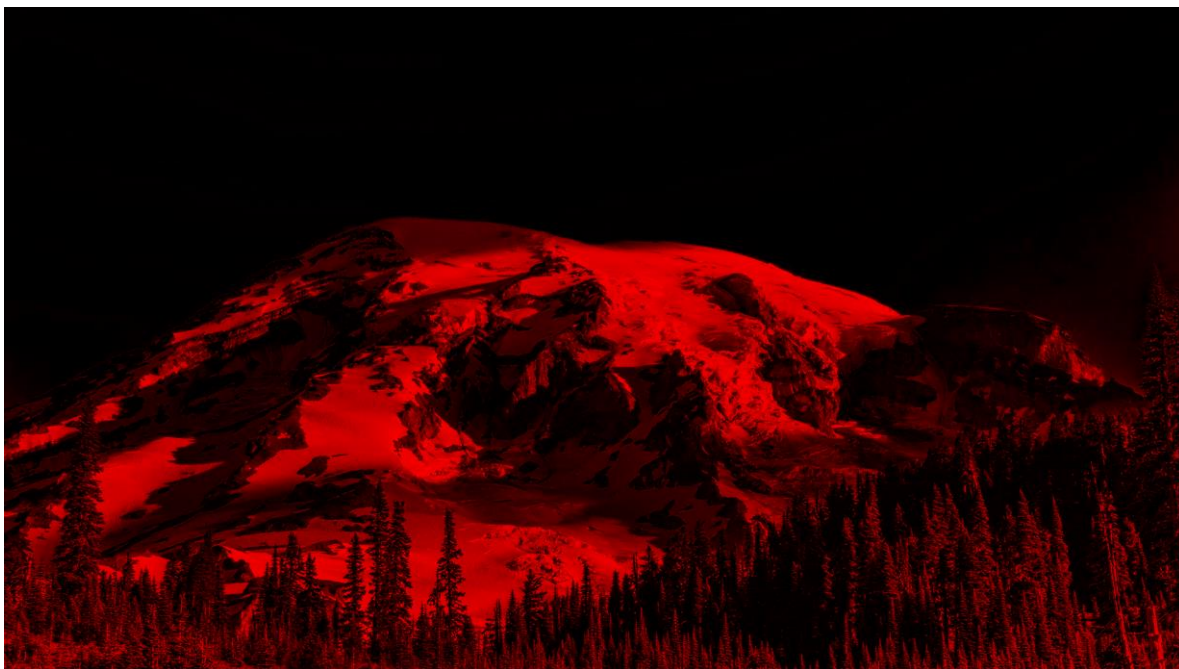


Figure 4: red component for original image



Figure5: green component for original image

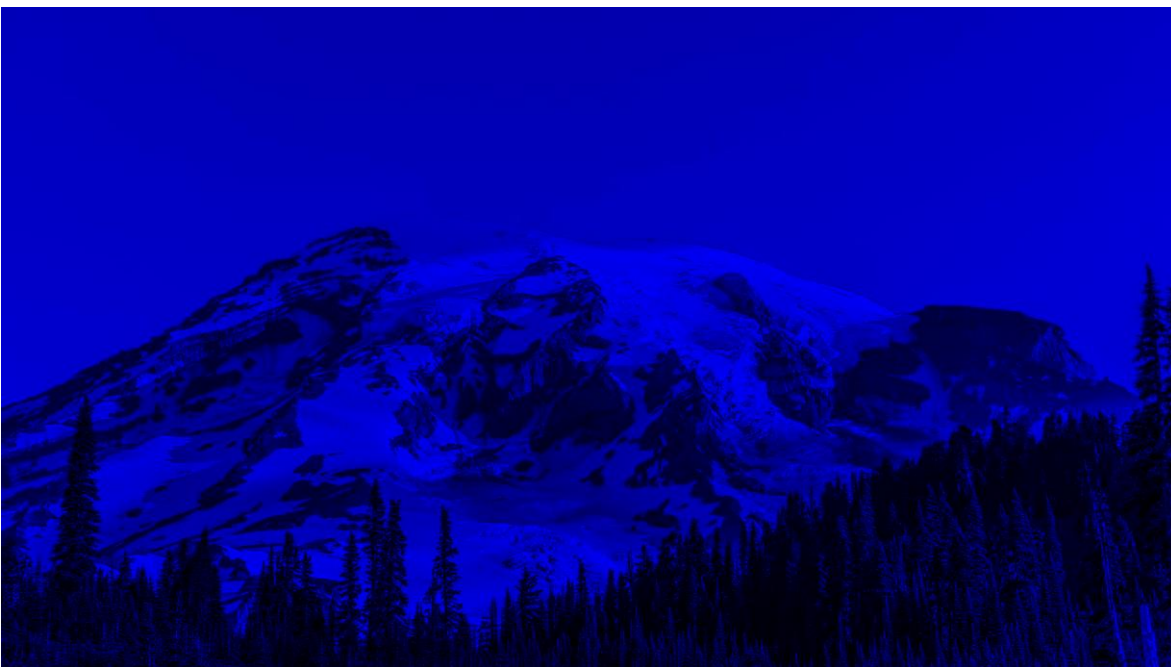


Figure6: blue component for original image



Figure7: Decompressed Image with $m=1$



Figure8: Decompressed Image with $m=2$



Figure9r: Decompressed Image with $m=3$



Figure10: Decompressed Image with $m=4$



Appendices

Appendix B: Codes for Task One: [Code Link](#)

```
%% Read the Audio
[x, fs] = audioread("audio1.wav");

%% Get y in time domain and add the Echo
y_t = zeros(length(x), 1);

for i = 1:length(x)
    y_t(i) = x(i);

    if i >= 2501
        y_t(i) = y_t(i) + x(i - 2500) * .9;
    end

    if i >= 4991
        y_t(i) = y_t(i) + x(i - 4990) * .8;
    end

    if i >= 7486
        y_t(i) = y_t(i) + x(i - 7485) * .7;
    end

end

sound(y_t); % the audio with echo

%% Get the impulse response

imp = zeros(length(x), 1);
imp(1) = 1;

h = imp;

for i = 1 : length(h)
    h(i, 1) = imp(i, 1);

    if i >= 2501
        h(i) = h(i) + imp(i - 2500) * .9;
    end

    if i >= 4991
        h(i) = h(i) + imp(i - 4990) * .8;
    end

    if i >= 7486
        h(i) = h(i) + imp(i - 7485) * .7;
    end
end
```



```
end
end

plot(h);
title('Impulse response');

%% Get y using convolution

y = conv(x, h);

%% Remove the echo using DFT

Y = fft(y);
H = fft(h);

H = imresize(H, [length(Y), 1]);

X = Y ./ H;

original_x = real(ifft(X));
sound(original_x(1: length(original_x) / 2));
```



Appendix B: Codes for Task Two:

```
clear all;
% Author: Ahmed Ashraf Hamdy

% Read two audio files
% %audio1 duration is less than audio2
[x1, fs1] = audioread("audio1.wav");
number_of_samples1 = length(x1);
duration_in_seconds1 = floor(number_of_samples1 / fs1);
% sound(x1, fs1);
% pause(duration_in_seconds1);

[x2, fs2] = audioread("audio2.wav");
number_of_samples2 = length(x2);
duration_in_seconds2 = floor(number_of_samples2 / fs2);

% Hide audio1 in audio2
[~,peaks1] = findpeaks(x1);
N1 = mean(diff(peaks1));
[~,peaks2] = findpeaks(x2);
N2 = mean(diff(peaks2));
omega = 2.4; %2pi/period
A = 0.05;

X = x2;
for n = 1 : number_of_samples1
    X(n) = x2(n)+A*x1(n)*cos(omega*n);
end

% Plot the magnitude spectrum of X
figure; plot(log10(abs(fft(X))));
sound(X, fs2);
pause(duration_in_seconds2);
audiowrite("newAudio2.wav",X,fs2);
Y = linspace(0, duration_in_seconds1, number_of_samples1);
for n = 1 : min(length(x1), length(x2))
    Y(n) = X(n)*cos(omega*n);
end
% frequency domain; fourier transform
Yfft = fft(Y);
% % multiply the range of Y[k] by zeros.
range = 10;
for k = floor(length(Yfft)/range) : (range)*floor(length(Yfft)/range)
    Yfft(k) = 0;
end
for n = 1 : length(Yfft)
    Yfft(n) = 4 * (Yfft(n) / A);
end
% % inverse fourier transform
Yifft = real(ifft(Yfft));
sound(Yifft, fs1);
pause(duration_in_seconds1);
audiowrite("newAudio1.wav",Yifft,fs1)
```



Appendix C: Codes for Task Three:

This code is written in python:

Packages needed:

- Math
- SciPy
- Cv2
- NumPy

```
from scipy.fft import dct ,idct
from math import log10
import numpy as np
import cv2

# Steps to follow

# 1. Encoder
# 1.1 Read the image file 'image1.bmp'. => Done
# 1.2 Extract and display each of its three color components. => Done
# 1.3 Convert range of each component to [-128, 127] => Done
# 1.4 Form a matrix for the outImage with the new size => Done
# 1.5 Process each color component in blocks of 8x8 pixels. => Done
# 1.6 Obtain 2D DCT of each block. => Done
# 1.7 Retain only the top left square of the 2D DCT coefficients of size  $m \times m$ , The rest of
coefficients are ignored. => Done
# 1.8 Compare the size of the original and compressed images. => Done

# 2. Decoder
# 2.1 load the out-image=>Done
# 2.2 display the compressed image=>Done
# 2.3 Form a matrix for the deCompressed image with the original size => Done
# 2.4 Get each block of to be decompressed.=>Done
# 2.5 apply inverse dct on each block=>Done
# 2.6 re-range the out image by adding 128 ranges from [0 : 255] => Done
# 2.7 display the decompressed image and Compare them => Done
# 2.8 quality of the decompressed image is measured using the Peak Signal-to-Noise Ratio PSNR)
implementation => Done
# 2.9 display PSNR for each m => Done
# 2.9 technical report (advantages of using DCT instead of DFT)

# Step 1.3
def reRange(inputImage):
    print("inputImage before", inputImage)
```




```
inputImage = inputImage.astype('int')
inputImage -= 128
print("inputImage after", inputImage)
return inputImage

# Step 1.2
def getComponent(inputImage, no):
    # 1. no = 0 => red
    # 2. no = 1 => green
    # 3. no = 2 => blue
    cpy = inputImage.copy()
    for i in range(3):
        if(i != no): # not need => Just make it zeros
            cpy[:, :, i] = 0
    return cpy

def imageCompression(inputImage, m, row, col):
    # Step 1.4
    outImage = np.zeros(
        (int((row / 8) * m), int((col / 8) * m), 3), dtype=np.float16)

    blockRow = int(row / 8)
    blockCol = int(col / 8)
    blockComponents = 3
    noIterations = 0

    # Step 1.5
    for x in range(0, blockRow):
        for y in range(0, blockCol):
            for z in range(0, blockComponents):
                noIterations += 1
                currentBlock = inputImage[x *
                    8: x * 8 + 8, y * 8: y * 8 + 8, z]

                # Step 1.6, 1.7
                blockDCT = dct(dct(currentBlock.T, norm='ortho').T, norm='ortho')[0:m, 0:m]
                outImage[x * m: x * m + m, y * m: y * m + m, z] = blockDCT
    print("no Iterations", noIterations)
    print("outImage", outImage)
    return outImage

# implement 2D IDCT
def idct2(a):
    return idct(idct(a.T, norm='ortho').T, norm='ortho')
# step 2.6
def deReRange(deCoproessedImage):

    deCoproessedImage += 128
    deCoproessedImage = deCoproessedImage.astype('int')
    return deCoproessedImage

def imageDeCompression(toBeCompressedImage ,m , row ,col):
    # Step 2.3
    deCompressedImage = np.zeros((int((row / m) * 8), int((col / m) * 8), 3), dtype=np.float16)

    blockRow = int(row / m)
    blockCol = int(col / m)
```



```
blockComponents = 3
noIterations = 0

# Step 2.4
for x in range(0, blockRow):
    for y in range(0, blockCol):
        for z in range(0, blockComponents):

            noIterations += 1
            currentBlock = toBeCompressedImage[x * m: x * m + m, y * m: y * m + m, z]
            deCompressedBlock = np.zeros((int(8), int(8)), dtype=np.float16)
            deCompressedBlock[0: m, 0: m] = currentBlock

            # Step 2.5
            blockIDCT = idct2(deCompressedBlock)
            deCompressedImage[x*8:x*8+8, y*8:y*8+8, z] = blockIDCT

# Step 2.6
deCompressedImage = deReRange(deCompressedImage)
return deCompressedImage

# Step 2.8
def getPSNR(original, compressed):

    MSE = np.mean((original - compressed) ** 2)
    max_pixel = 255.0
    PSNR = 10 * log10((max_pixel*max_pixel) / MSE)
    return PSNR

# Step 1.1
inputImage = cv2.imread('./image1.bmp')
np.save("inputImage", inputImage)

row = inputImage.shape[0]
col = inputImage.shape[1]
m = int(input('Enter the value of m between [1 - 4] : '))

cv2.imshow("Input Image", inputImage)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Step 1.2
# Get Red Component
redComponent = getComponent(inputImage, 2)
cv2.imwrite("redComponent.bmp", redComponent)
cv2.imshow("Red Component", redComponent)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Get Green Component
greenComponent = getComponent(inputImage, 1)
cv2.imwrite("greenComponent.bmp", greenComponent)
cv2.imshow("Green Component", greenComponent)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



```
# Get Blue Component
blueComponent = getComponent(inputImage, 0)
cv2.imwrite("blueComponent.bmp", blueComponent)
cv2.imshow("Blue Component", blueComponent)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Step 1.3
inputImage = reRange(inputImage)

# Step 1.8
outImage = imageCompression(inputImage, m, row, col)
print("Output Image", outImage)
np.save("outImage", outImage)

# Step 2.1
toBeCompressedImage = np.load("outImage.npy")

# Step 2.2
print("decompressed", toBeCompressedImage)
deRow = toBeCompressedImage.shape[0]
deCol = toBeCompressedImage.shape[1]

deCompressedImage = imageDeCompression(toBeCompressedImage, m, deRow, deCol)
print("deCompressedImage", deCompressedImage)
print("inputImage", inputImage)
print(deCompressedImage.shape[0], deCompressedImage.shape[1])

# Step 2.7
cv2.imwrite("deCompressedImage.bmp", deCompressedImage)
deCompressedImage = cv2.imread("./deCompressedImage.bmp")
cv2.imshow("deCompressed Image", deCompressedImage)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Step 2.9
inputImage = cv2.imread('./image1.bmp')
deCompressedImage = cv2.imread("./deCompressedImage.bmp")
PSNR = getPSNR(inputImage, deCompressedImage)
print("PSNR", PSNR)
```



References:

1. References for question (F) What are the advantages of using DCT instead of DFT for compression.
 - i. The Discrete Cosine Transform(DCT) Theory and Application Research prepared by Syed Ali Khayam from Department of Electrical & Computer Engineering Michigan State University [Research download link](#).
 - ii. Answer of question “what is the difference between DCT and DFT” from Research Gate website [Answer Link](#).
 - iii. Answer from Quora about “why DCT preferred over DFT” [Answer Link](#).
 - iv. Answer from Stack Exchange website [Answer Link](#).