Cairo University- Faculty of Engineering
Computer Engineering Department
Signals and Systems (ELC 225B) – Spring 2020

Final Project

Team Members

Full Name	Sec.	BN	Role in the project
Hossam Alaa Ramadan	**1**	**23**	**Task1**
Ali Abdel-Halim Ali	**2**	**6**	**Task1**
Mahmoud Amr Muhammad Refaat	**2**	**25**	**Task2**
Muhammad Ahmed Hesham	**2**	**17**	**Task3**
Muhammad Alaa Abdel-Khaliq	**2**	**24**	**Task3**

Table of contents:

List of Figures

Project Tasks:

Task One: Echo generation and removal

Input before any modifications and the output after adding echo and removing it

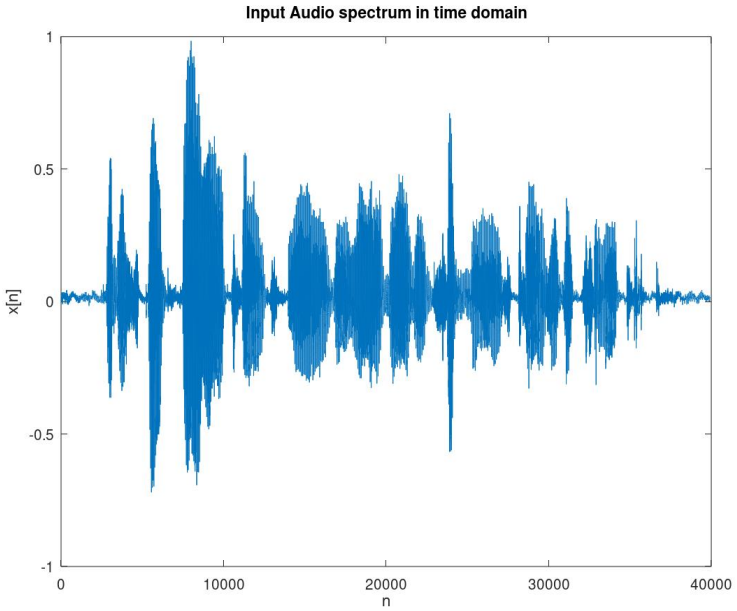

Figure 1 : input audio

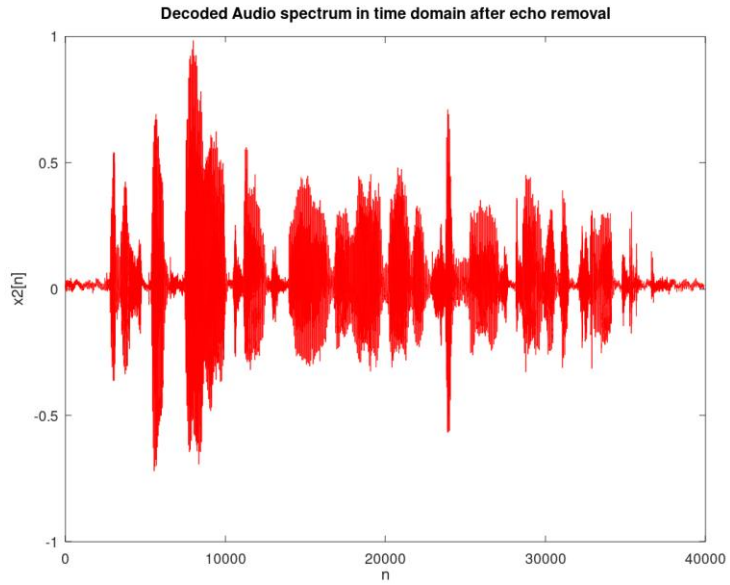

Figure 2 : output audio

Plot of impulse response of $y(t) = x(t) + 0.9\,x(t - 0.25) + 0.8\,x(t - 0.5) + 0.7\,x(t - 0.75)$

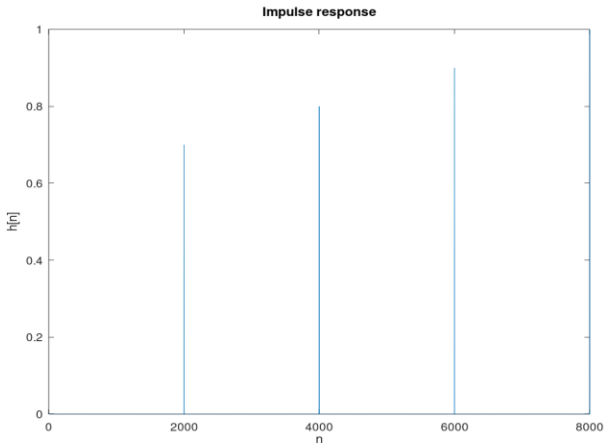

Figure 3 : impulse response plot

Audio in frequency domain after adding echo

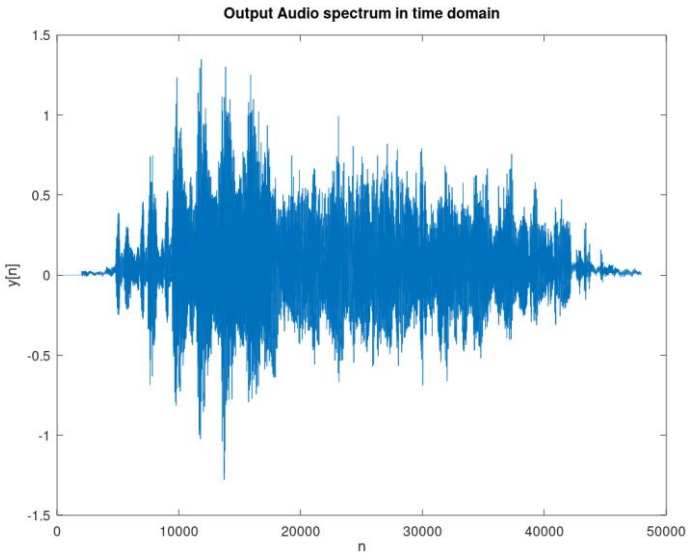

Figure 4 : audio after adding echo in frequency domain

Task Two: Audio steganography

Result of resampling the two audio files at 52100 Hz to shift audio 1 (message) to the range from 22050 Hz to 26050 Hz which is inaudible to the human ear & not overlapping with audio 2 spectral range (carrier) where Fs = 2 * max frequency (Nyquist rate)

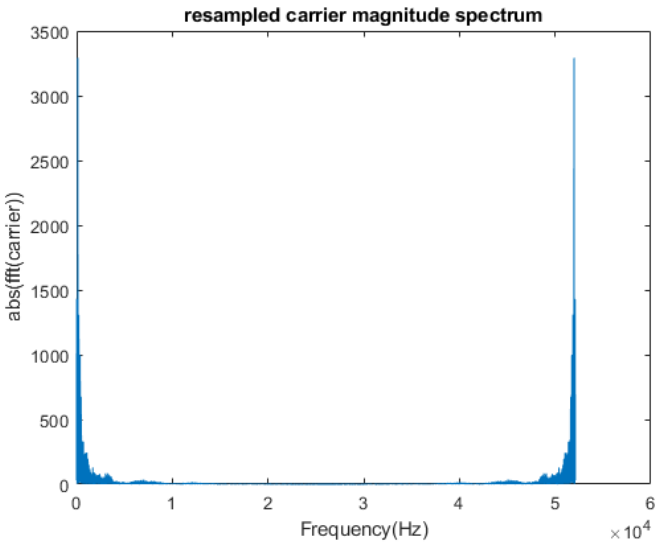

Figure 5: resampled carrier (audio 2)

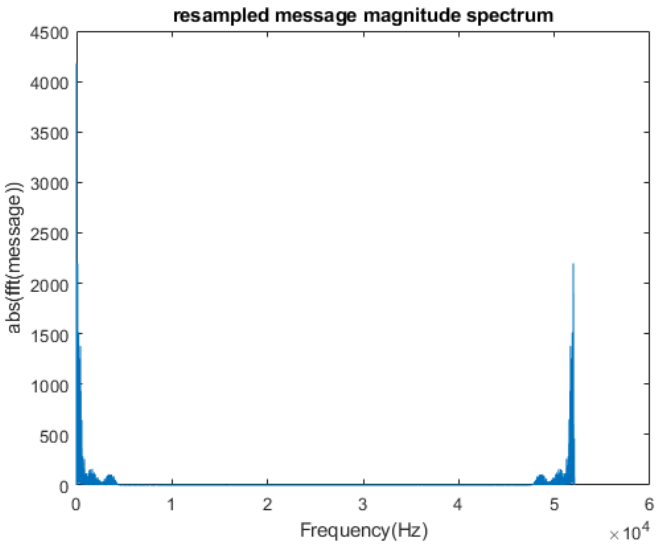

Figure 6: resampled message (audio 1)

Results of hiding the message inside the carrier (magnitude spectrum of x[n])

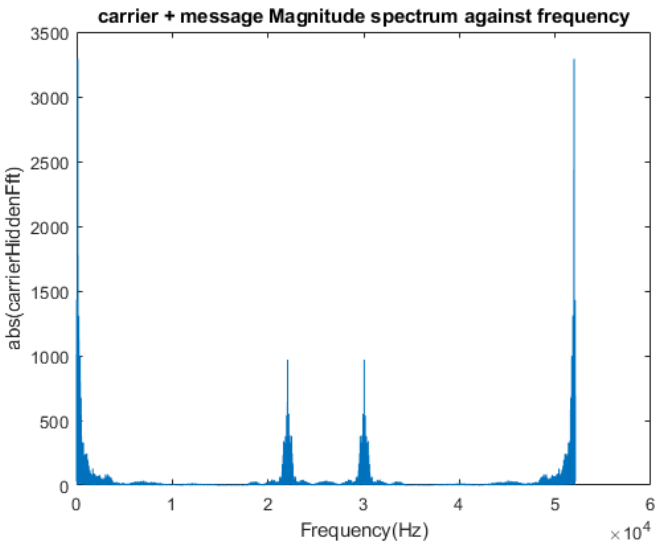

Figure 7: message inside carrier magnitude spectrum against frequency

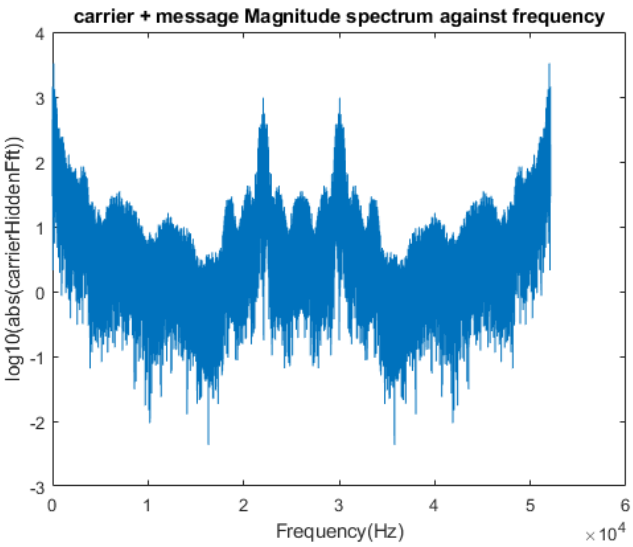

Figure 8: message inside carrier against frequency taking log

Retrieved message before and after removing the unwanted range from the frequency domain (On retrieving the hidden message, multiplication by a factor is done to reverse the effect of attenuation before removing the unwanted range)

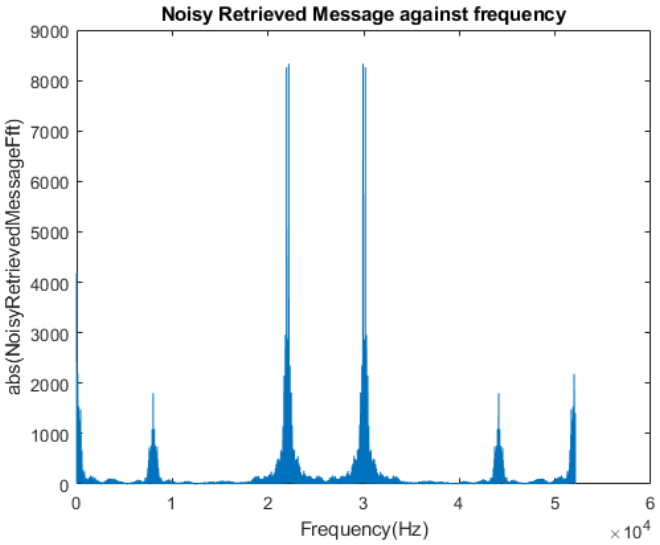

Figure 9: retrieved Message before removing the unwanted range of frequencies

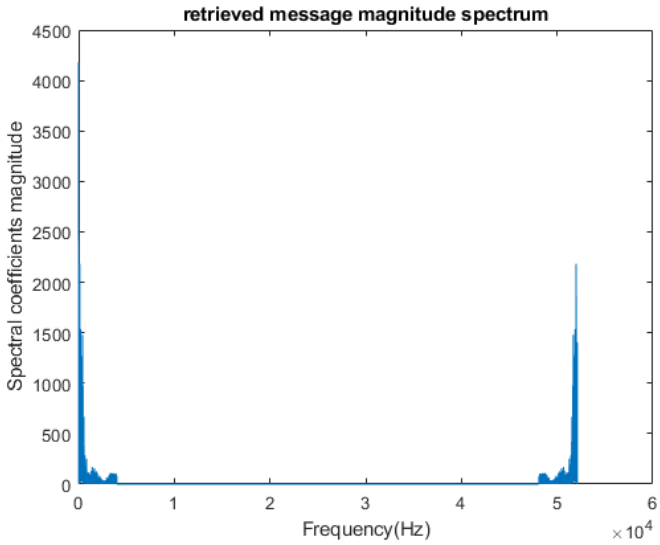

Figure 10: retrieved Message after removing the unwanted range of frequencies

Notes:

- All graphs are plotted against frequency where frequency is calculated from the output k of the fast Fourier transform (bin frequencies) as follows:
  At any point k from [0: number of samples – 1], corresponding frequency = (delta frequency * k)
  Where delta frequency is the increment between two frequencies and is calculated as follows:
  Delta frequency = sampling frequency / number of samples
- The output of the MATLAB code are two audio files (**hidden.wav & retrieved.wav)**
  Produced in the current folder in MATLAB where **hidden** is the wav file of the carrier file containing the message, and **retrieved** is the wav file containing the message after retrieval

Task Three: Image compression

A) Color components of the image

Figure 12 Green Channel

Figure 11 Red Channel

Figure 13 Blue Channel

B) compression Ratio with different m (input file size / encoded file size)

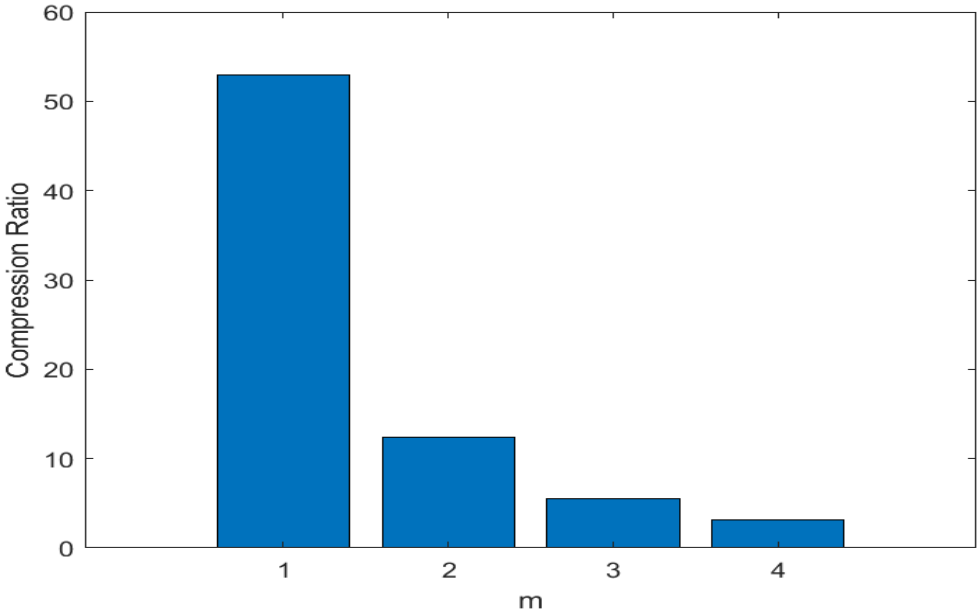

Figure 14 compression Ratio (input file size/encoded file size)

m value	Size
original	6,220,854 bytes
1	117,435 bytes
2	500,612 bytes
3	1,122,146 bytes
4	1,991,079 bytes

Note:

the size of the encoded file is the size needed to store the m*m coefficient of each block and the coefficients are mostly floating point numbers not integers, so knowing the fact that matlab has the data type half (half precision) which is the smallest data type they support that can store floats so each coefficient takes 2 bytes but if each coefficient took 1 byte the size would be even smaller.

C) Decompressed images with different m values compared to the original

Figure 15 Decoded image with m=1 on the left compared to the original

Figure 16 Decoded image with m=2 on the left compared to the original

Figure 17 Decoded image with m=3 on the left compared to the original

Figure 18 Decoded image with m=4 on the left compared to the original

D) PSNR values for different m:

m value	PSNR
1	19.8638
2	21.9009
3	23.7548
4	25.9031

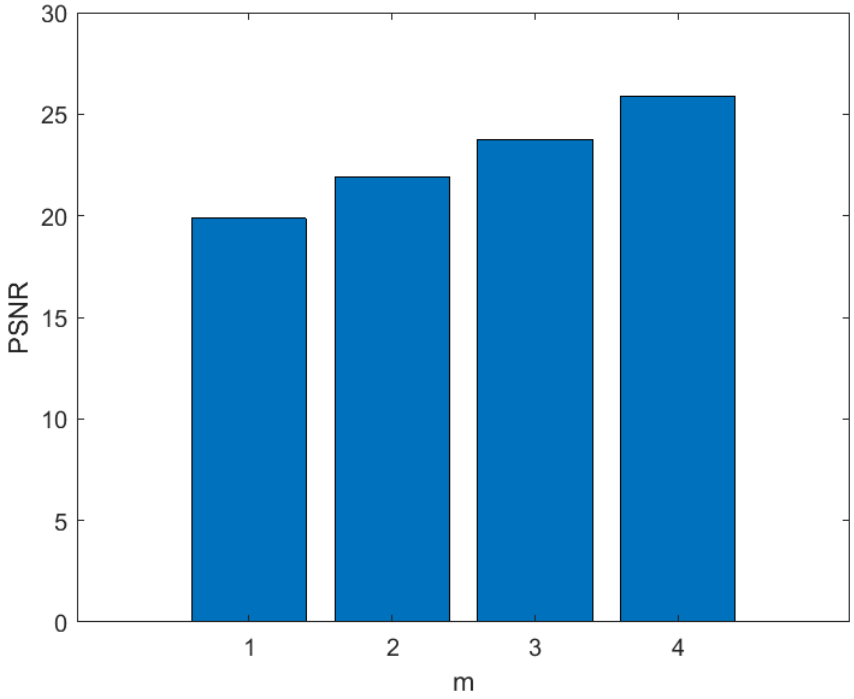

Figure 19 PSNR value for different m values

E) The advantages of using DCT instead of DFT:

A) DFT:

- It is the most used because of its computational efficiency.
- But, it has critical disadvantages for some applications as:
  - It is complex.
  - It has low **energy compaction**.

Energy compaction:

- It is the ability of grouping a spatial sequence of energy into the fewest possible frequency coefficients.
- It is very used in image compression.
- We apply transformation on image to transform it into the frequency domain.
- If the image has a very high compaction, we just need to transmit few coefficients instead of the whole set of pixels.

B) DCT:

- It is much better from the **DFT**
- it has a much more concentrated histogram which is obvious in the histogram of amplitude spectra below.

Figure 20 image to get DCT and DFT coefficients

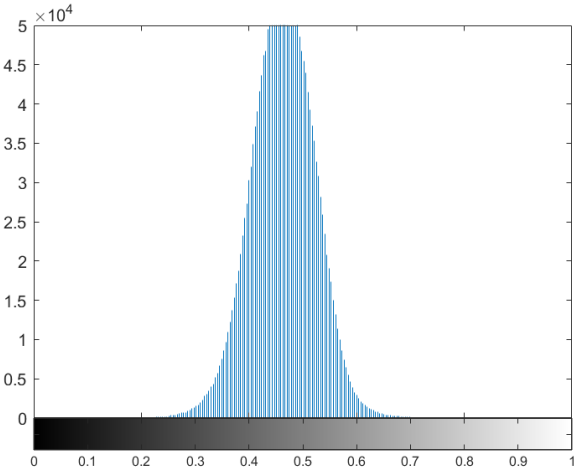

Figure 23 histogram of DFT coefficients

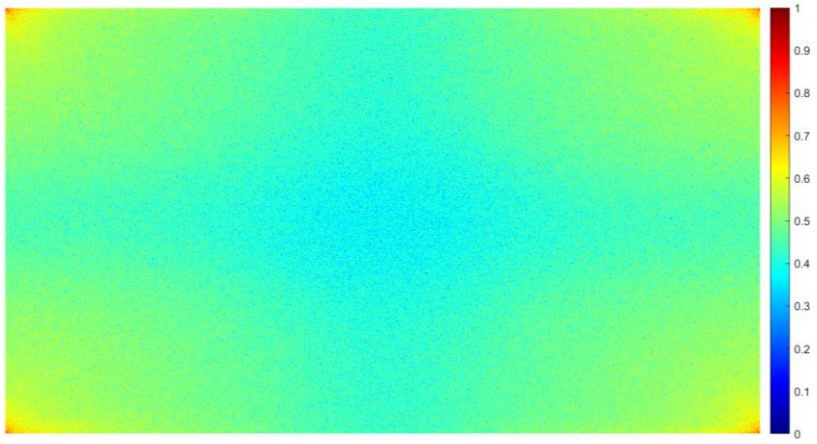

Figure 21 DFT coefficients scaled

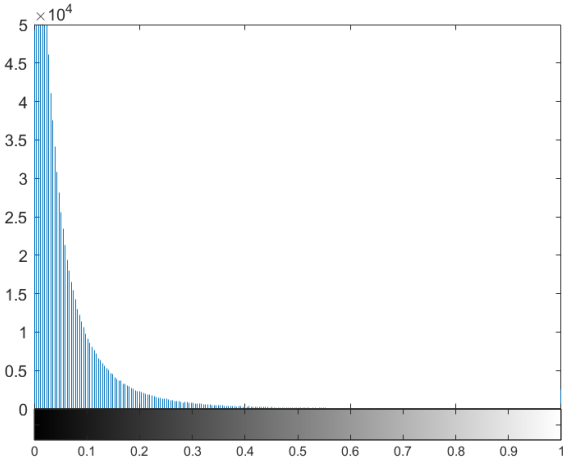

Figure 24 histogram of DCT coefficients

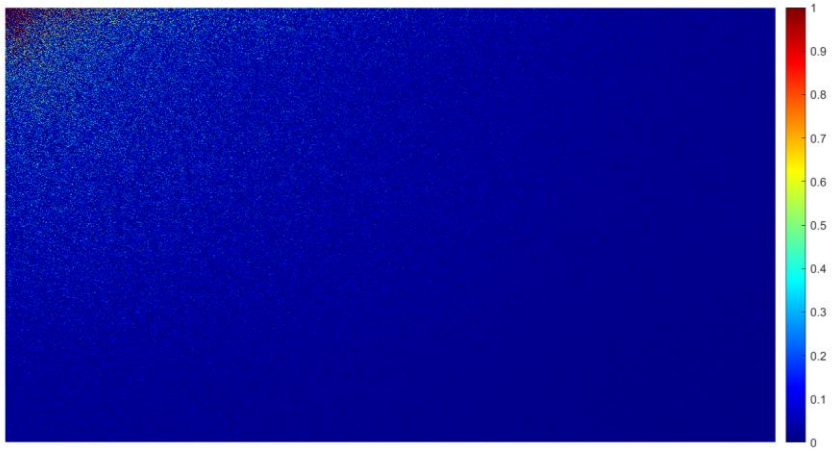

Figure 22 DCT coefficients scaled

why is energy compaction important?

- the main reason is image compression.
- turns out to be beneficial in other applications (Vasconcelos)

Appendices

Appendix A: Codes for Task One: Octave code

```
#variables
# x:input signal in time domain(original audio)
# X:input signal in frequency domain
# h:impulse response
# H:frequency response
# y:output audio in time domain(audio with echo)
# Y:output audio in frequency domain
# x2:audio after echo removal in time domain

#read audio file
[x,fs] = audioread("audio1.wav");
original_audio_length = length(x)

#plot spectrum of signal
figure(1)
plot(x);
title("Input Audio spectrum in time domain")
ylabel("x[n]")
xlabel("n")
```

```
#calculate h[n]
h=zeros(fs,1);
h(cast(ceil(0.25*fs),"int32"))=0.7;
h(cast(ceil(0.5*fs),"int32"))=0.8;
h(cast(ceil(0.75*fs),"int32"))=0.9;
h(fs)=1;

#plot h[n]
figure(2)
plot(h);
title("Impulse response")
ylabel("h[n]")
xlabel("n")

#calculate echo function using convolution
y=conv(x,h);
figure(3)
plot(y)
title("Output Audio spectrum in time domain")
ylabel("y[n]")
xlabel("n")
```

```
#write output audio file after apply echo

audiowrite("EchoAudio.wav",y,fs);


#Echo removal part

L=max([length(y);length(h)]);

#make sure that length is not less than y by padding the difference with zeros

H=fft([h;zeros(L-length(h),1)]);

Y=fft([y;zeros(L-length(y),1)]);


#find x2(decoded audio) using inverse of forier transform with original length

x2 = real(ifft(Y ./ H))


#removes very small values due to error in precision of calculations which causes redundant memory and time

x2 = x2(abs(x2) > 10^-15);


#plotting decoded audio after echo removal

figure(4)

plot(x2,"r");

title("Decoded Audio spectrum in time domain after echo removal")

ylabel("x2[n]")

xlabel("n")


#write decoded file

audiowrite("decodedAudio.wav",x2,fs)
```

Appendix B: Codes for Task Two: MATLAB code (MATLAB R2018a)

```
%%
% Audio Steganography Task 2
%
% Read and plot the Message audio file (audio1)

[message, messageFs] = audioread("audio1.wav");
%%
% Read and plot the Carrier audio file (audio2)

[carrier, carrierFs] = audioread("audio2.wav");
%%
% Resamping the audio files to a new sampling rate newFs to satisfy the
% sampling theorem when moving the message to a higher frequency band
%%
% - Frequencies at the range 20 KHz should be inaudible to the human ear
% - The carrier has frequency components from 0 to carrierFs/2 which is 22050
Hz
% - The Message has frequency components from 0 tp messageFs/2 which is 4000
Hz
% - Inorder to hide the message from 22050 Hz to 26050 Hz we need to sample
both signals
% with the same samping rate 2 * 26050 = 52100 to be able to add them
newFs = 52100;

[p, q] = rat(newFs/messageFs);
message = resample(message, p, q);
% get the message number of samples after resampling and the scales to use for
plotting
nMessageSamples = length(message);
messageFftFrequencyScale = (newFs/nMessageSamples) * (0:nMessageSamples - 1);
messageTimeScale = (1/newFs) * (0:nMessageSamples - 1);

[p, q] = rat(newFs/carrierFs);
carrier = resample(carrier, p, q);
% get the carrier number of samples after resampling and the scales to use for
plotting
nCarrierSamples = length(carrier);
carrierFftFrequencyScale = (newFs/nCarrierSamples) * (0:nCarrierSamples - 1);
carrierTimeScale = (1/newFs) * (0:nCarrierSamples - 1);
% plot the message in frequency domain
figure(2)
plot(messageFftFrequencyScale, abs(fft(message)))
xlabel("Frequency(Hz)")
ylabel("abs(fft(message))")
title("resampled message magnitude spectrum")
```

```
% plot the carrier in frequency domain
figure(3)
plot(carrierFftFrequencyScale, abs(fft(carrier)))
xlabel("Frequency(Hz)")
ylabel("abs(fft(carrier))")
title("resampled carrier magnitude spectrum")
%%
% Preparing the parameters to be used for hiding the message into the carrier
%%
attenuationFactor = 0.5;
shiftingFrequency = 22050;

shiftingCosineMessage = cos(2 * pi * shiftingFrequency * messageTimeScale)';
%%
% Hiding the message into the carrier using the equation to shift the message
% frequency band to the higher inaudible frequency components where they
% don't overlap with the carrier frequency components and attenuating them
%%
carrierHidden = zeros(1, nCarrierSamples);
carrierHidden = carrierHidden + carrier';
shiftedMessage = attenuationFactor * (message .* shiftingCosineMessage);
carrierHidden(1:nMessageSamples) = carrierHidden(1:nMessageSamples) +
shiftedMessage';

% Write the carrier containing the message to an audio file
audiowrite('hidden.wav', carrierHidden, newFs);

carrierHiddenFft = fft(carrierHidden);
% plot the magnitude spectrum of the fft of the carrier holding the message
figure(4)
plot(log10(abs(carrierHiddenFft)))
xlabel("k")
ylabel("log10(abs(carrierHiddenFft))")
title("carrier + message Magnitude spectrum")

% plot the magnitude spectrum of the fft with the frequency scale taking
% log
figure(5)
plot(carrierFftFrequencyScale, (log10(abs(carrierHiddenFft))))
xlabel("Frequency(Hz)")
ylabel("log10(abs(carrierHiddenFft))")
title("carrier + message Magnitude spectrum against frequency")

% plot the magnitude spectrum of the fft with the frequency scale
figure(6)
plot(carrierFftFrequencyScale, abs(carrierHiddenFft))
xlabel("Frequency(Hz)")
ylabel("abs(carrierHiddenFft)")
title("carrier + message Magnitude spectrum against frequency")
```

```
% plot the magnitude spectrum of the fft with the frequency scale
%%
% Retrieving the hidden message
%%
shiftingCosineCarrier = cos(2 * pi * shiftingFrequency * carrierTimeScale);

NoisyRetrievedMessage = 4 * carrierHidden .* shiftingCosineCarrier;

%plot the fft of the noisy message
figure(7)
NoisyRetrievedMessageFft = fft(NoisyRetrievedMessage);
plot(abs(NoisyRetrievedMessageFft))
xlabel("k")
ylabel("abs(NoisyRetrievedMessageFft)")
title("Noisy Retrieved Message in frequency domain")

figure(8)
NoisyRetrievedMessageFft = fft(NoisyRetrievedMessage);
plot(carrierFftFrequencyScale, abs(NoisyRetrievedMessageFft))
xlabel("Frequency(Hz)")
ylabel("abs(NoisyRetrievedMessageFft)")
title("Noisy Retrieved Message against frequency")
%%
% Removing the noise in Frequency domain  to restore the original message
%%
% filter from freq 0 Hz to 4000 Hz from Left hand side & its mirrored part on
% Right hand side of fft magnitude spectrum since the original sampling rate
% of the message signal was 8000 Hz
% specify the region in terms of Bin frequencies k
kStart = cast(4000 * (nCarrierSamples / newFs), 'int32');
kEnd = cast((newFs - 4000) * (nCarrierSamples / newFs), 'int32');

retrievedMessageFft = NoisyRetrievedMessageFft;
retrievedMessageFft(kStart: kEnd) = retrievedMessageFft(kStart: kEnd) * 0;

retrievedMessage = ifft(retrievedMessageFft, 'symmetric');
figure(9)
plot(carrierFftFrequencyScale, abs(retrievedMessageFft))
xlabel("Frequency(Hz)")
ylabel("Spectral coefficients magnitude")
title("retrieved message magnitude spectrum")

% Remove the last extra two seconds from the audio file to restore the length
of the old
% audio file
retrievedMessage = retrievedMessage(1: nMessageSamples);
audiowrite('retrieved.wav', retrievedMessage ,newFs)
```

Appendix C: Codes for Task Three: MATLAB code (MATLAB R2018a)

```
% Read in original RGB image.
global rgbImage;
rgbImage = (im2double(imread('image1.bmp')));
[height, width, numberofChannels] = size(rgbImage);

% get the size of the original image
fid = fopen('image1.bmp');
fseek(fid, 0, 'eof');
originalSize = ftell(fid);
fclose(fid);

% Extract color channels.
redChannel = rgbImage(:,:,1);
greenChannel = rgbImage(:,:,2);
blueChannel = rgbImage(:,:,3);

% Display each channel
figure('Name','red Channel','NumberTitle','off');
imshow(redChannel);
figure('Name','green Channel','NumberTitle','off');
imshow(greenChannel);
figure('Name','blue Channel','NumberTitle','off');
imshow(blueChannel);

% get the dct matrix of dimension 8*8
global T;
T = dctmtx(8);

max = 4; % the maximum m to be used
psnr = zeros(max,1);  % to store psnr of the different m used
compressionRatio = zeros(max, 1);   % to store the compression ratio

for i=1:max
    compressedSize = DCTCompress(i, height, width, redChannel, greenChannel,
blueChannel);
    compressionRatio(i) = originalSize / compressedSize;
end

for i=1:max
    thePSNR = DCTDecompress(i, height, width);
    psnr(i) = thePSNR;
end

% display the psnr of the different m values
x = 1:1:max;
figure('Name','PSNR','NumberTitle','off');
bar(x, psnr);
xlabel('m');
ylabel('PSNR');

% display the compression ratio of the different m values
figure('Name','Compression Ratio with different m','NumberTitle','off');
bar(x, compressionRatio);
xlabel('m');
ylabel('Compression Ratio');
```

```
function compressedSize = DCTCompress(m, height, width, redChannel, greenChannel,
blueChannel)

    % make three matrices to hold the r, g and b dct coefficients
    newWidth = width / (8 / m);
    newHeight = height / (8 / m);
    newRedChannel = zeros(newHeight, newWidth);
    newBlueChannel = zeros(newHeight, newWidth);
    newGreenChannel = zeros(newHeight, newWidth);

    % dct compress and mask coefficients around the m*m block of the 8*8
    % block (keeping only m*m coefficient) of each 8*8 coefficient
    for i=1:height/8
        for j=1:width/8
           startX = (i-1) * m + 1;
           endX =  i * m;
           startY = (j-1) * m + 1;
           endY = j * m;

           block = compressBlock(blueChannel, i, j, m);
           newBlueChannel(startX:endX, startY:endY) = block;

           block = compressBlock(redChannel, i, j, m);
           newRedChannel(startX:endX, startY:endY) = block;

           block = compressBlock(greenChannel, i, j, m);
           newGreenChannel(startX:endX, startY:endY) = block;
        end
    end

    % save the coefficients to a file
    compressedRgbImage = (cat(3,newRedChannel,newGreenChannel,newBlueChannel));
    compressedRgbImage = half(compressedRgbImage);
    s1 = mfilename('fullpath');
    s2 = 'encodedm';
    s3 = '.mat';
    save([s1 s2 int2str(m) s3], 'compressedRgbImage');

    % compressed size is the size needed to store the coefficients and
    % because they are float minimum size in matlab is 2 bytes per
    % coefficient as minimum size float is (half) in matlab
    fid = fopen([s1 s2 int2str(m) s3]);
    fseek(fid, 0, 'eof');
    compressedSize = ftell(fid);
    fclose(fid);

end

function block = compressBlock(Channel, start_x, start_y, m)
  global T;
  temp = Channel((start_x-1) * 8 + 1: start_x * 8, (start_y-1) * 8 + 1: start_y * 8);
  temp = T * temp * T';
  block = temp(1:m, 1:m);

end
```

```
function thePSNR = DCTDecompress(m, height, width)
    % make matrices to hold the channels of the decompressed image
    newRedChannel = zeros(height, width);
    newBlueChannel = zeros(height, width);
    newGreenChannel = zeros(height, width);

    % load the coefficients from the stored comrpessed file
    s1 = mfilename('fullpath');
    s2 = 'encodedm';
    s3 = '.mat';
    rgbImageCompressed = load([s1 s2 int2str(m) s3]);
    rgbImageCompressed = double(rgbImageCompressed.compressedRgbImage);

    % Extract color channels.
    encodedRedChannel = rgbImageCompressed(:,:,1);
    encodedGreenChannel = rgbImageCompressed(:,:,2);
    encodedBlueChannel = rgbImageCompressed(:,:,3);

    % inverse dct for each block 8*8 which consists of m*m coefficient and
    % zeros around that m*m block
    for i=1:height/8
        for j=1:width/8
           block = DecompressBlock(encodedBlueChannel, i, j, m);
           newBlueChannel((i-1) * 8 + 1: i * 8, (j-1) * 8 + 1: j * 8) = block;

           block = DecompressBlock(encodedRedChannel, i, j, m);
           newRedChannel((i-1) * 8 + 1: i * 8, (j-1) * 8 + 1: j * 8) = block;

           block = DecompressBlock(encodedGreenChannel, i, j, m);
           newGreenChannel((i-1) * 8 + 1: i * 8, (j-1) * 8 + 1: j * 8) = block;
        end
    end
    % save the decompressed image to file
    DecompressedRgbImage = (cat(3,newRedChannel,newGreenChannel,newBlueChannel));
    s1 = mfilename('fullpath');
    s2 = '.bmp';
    imwrite(DecompressedRgbImage, [s1 int2str(m) s2]);

    % calculate the psnr and compare the output with the original visually
    global rgbImage;
    thePSNR = psnr(DecompressedRgbImage, rgbImage);
    s1 = 'comparison between original and compressed image with m = ';
    figure('Name',[s1 int2str(m)],'NumberTitle','off');
    imshowpair(DecompressedRgbImage, rgbImage, 'montage');
end

function block = DecompressBlock(Channel, start_x, start_y, m)
  global T;
  block = zeros(8, 8);
  block(1:m, 1:m) = Channel((start_x-1) * m + 1: start_x * m, (start_y-1) * m + 1:
start_y * m);
  block = T' * block * T;
end
```

Code used to get the histogram and energy distribution of DCT and DFT

```
rgbImage = rgb2gray(im2double(imread('image1.bmp')));
figure
imshow(rgbImage)

y = abs(fft2(rgbImage));
y = log(y); % Use log, for perceptual scaling
y = mat2gray(y); %Use mat2gray to scale the image to (0, 1)
figure
imhist(y)
ylim([0 50000])


figure
imshow(y)
colormap(gca,jet(512))
colorbar


y = ((dct2((rgbImage))));
figure
imhist(y)
ylim([0 50000])

figure
imshow(y)
colormap(gca,jet(512))
colorbar
```

References

Vasconcelos, N. (n.d.). *Discrete Cosine Transform.* Retrieved from UC San Diego: http://www.svcl.ucsd.edu/courses/ece161c/handouts/DCT.pdf