Signals Project

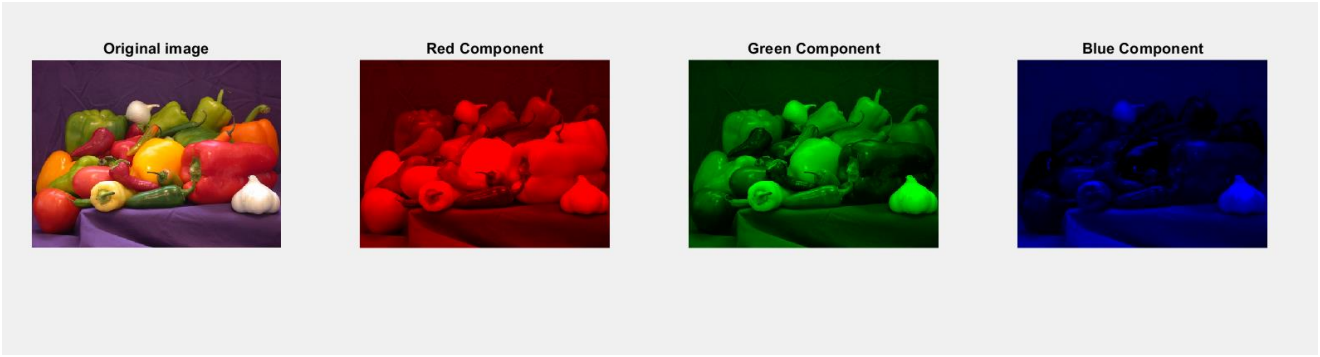
ID	الاسم
9220392	شهاب الدين طارق فؤاد
9210687	علي محمد محمد سيد

Instructor : Dr. Michael Melek Abdel-Sayed

Task A :

-I have read the ‘pepper.png’ image and I store the image in the variable ‘`img_SA`’

Then I deal with the image as 3x3 matrix then I take each matrix (red,green,blue) and store each matrix in separate variable then I make variable called ‘a’ and make it zero matrix with the same size of the original image to clear the the another components to show a specific components then I will separate each components and put it in anew 3x3 matrix by concatenating each component with two ‘a’ matrices then I showed the three images and this is the output :

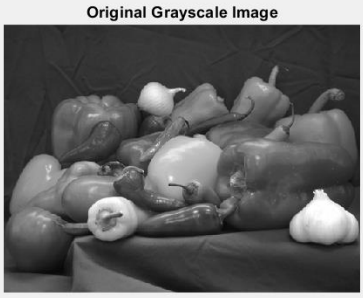


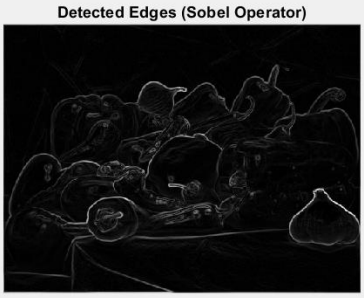
Task B.1 :

-First we convert the image to the grey scale to convolute it with the kernel then we get the **Sobel Kernels** which is a convolution kernel used in image processing and computer vision for edge detection. It is specifically designed to highlight edges in an image by computing the gradient of the image intensity. The Sobel operator is commonly used to find the approximate absolute gradient of the image intensity function. The Sobel operator consists of two separate 3x3 convolution kernels, one for detecting changes in the horizontal direction (often denoted as Sobel_x or G_x) and the other for changes in the vertical direction (often denoted as Sobel_y or G_y)

Then I convolute the double form of the grey scale Image to make each pixel value in float form to be more accurate once with the horizontal kernel with the

same size of the original image and store it in a variable and make the same process with the vertical kernel then I take the magnitude of the two matrices the I normalizing the image by dividing the image by the highest value in it to make all the pixels between 0 and 1 then showing the images as follows :



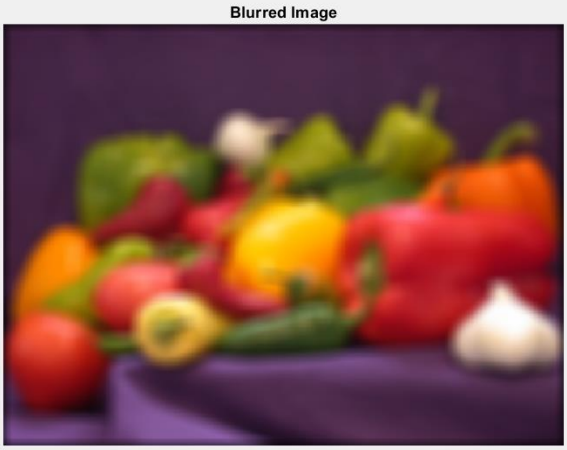


Task B.2:

- I take the image and then defining the Laplacian kernel which often used for edge detection in image processing, is a 3x3 convolution kernel that highlights regions of rapid intensity change in an image. It calculates the second spatial derivative of the image, emphasizing areas where the intensity changes abruptly and then I separate each component in a separate matrix and convolute the kernel with each component in double form to make each pixel value in float form to be more accurate and store it in a new variable of the sharpened image then showing the sharpened image as follows :

Task B.3 :

-Defining the blur kernel which is matrix of ones divide each pixel by the count of the matrix elements which means averaging each pixel then separate each component of the image in a separate variable then convolute each component with the blur kernel in a double form to be more accurate the concatenate the three components in new image and showing the image as follows :

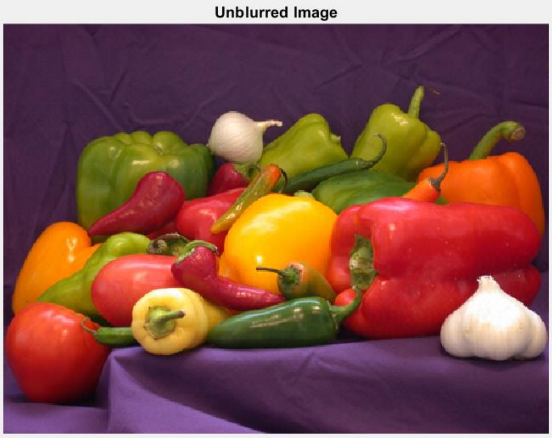


Task B.4 :

-First I read the image then I define the motion kernel which is 21x21 matrix (horz_blur) is initialized with all elements set to zero. This matrix will be used as the kernel for horizontal blurring The insertShape function is used to insert a white horizontal line into the previously initialized matrix. The line is defined by the parameters [21, 10, 0, 10], where (21, 10) and (0, 10) are the endpoints of the line. The color of the line is set to white ([255 255 255]), and the line width is set to 1 pixel. Finally, the kernel is normalized by dividing each element by the sum of all elements in the matrix. This normalization ensures that the sum of the elements in the kernel is equal to 1, which is a common practice for convolution kernels in image processing, Then I separate the components of the image in three variables then I convoluted each component with the kernel then I concatenated the three convoluted components into one image and print it as follows :

Task C :

-I define the Fourier transform of the motion kernel then I define the Fourier transform of the motion blured image then I defined a constant and add it to all elements of the motion kernel Fourier transformed to avoid divide any element by zero which is ‘C’ and equals 0.00009 an this value I get it by trying random values until I found this suitable value then I divide the motion blured image by the kernel element by element in the frequency domain the I restore the image by inverse Fourier transform then I take the absolute to get the magnitude then I modify the restored image to be with the same size as same as the original image and the restored image is as follows :



II. Communication system simulation:

Task a :

-First I have chosen a frequency sample equals 44100 To make sure that all the frequency components of the signal are detected by our digital system(Computer) we should choose a sampling frequency which is slightly twice the maximum frequency component that can be found in our signal. Since our signal is a human voice whose frequency components ranges from 20HZ to 20KHZ so that our

sampling frequency must be slightly greater than 40KHZ, for example 43KHZ. We have choose 44.1KHZ which is the most common used sampling frequency

-for human voice records and Our system uses 16 bits as our bit depth and this is sufficient to have an accurate digital representation for each sample of the recorded audio

Task b , c:

-first I read the audio and store the amplitude values in the variable ‘amp’ and the store the frequency of sampling of the audio in the variable ‘Fs’,

-Store the length of the audio in the variable ‘N’

-compute the Fourier transform of the audio and store it in ‘`Original_Audio_Freq`’

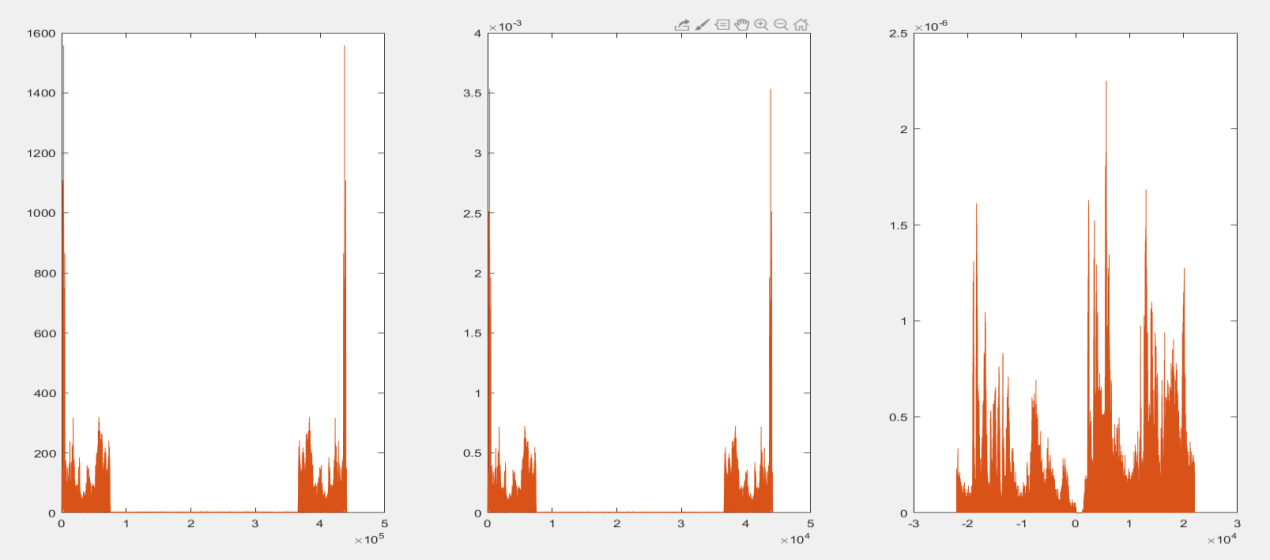
-plotting the single sided of the audio with the k vector which is vector of the length of the audio

- Then I make the vector of the frequencies of the audio from zero to Fs/N then

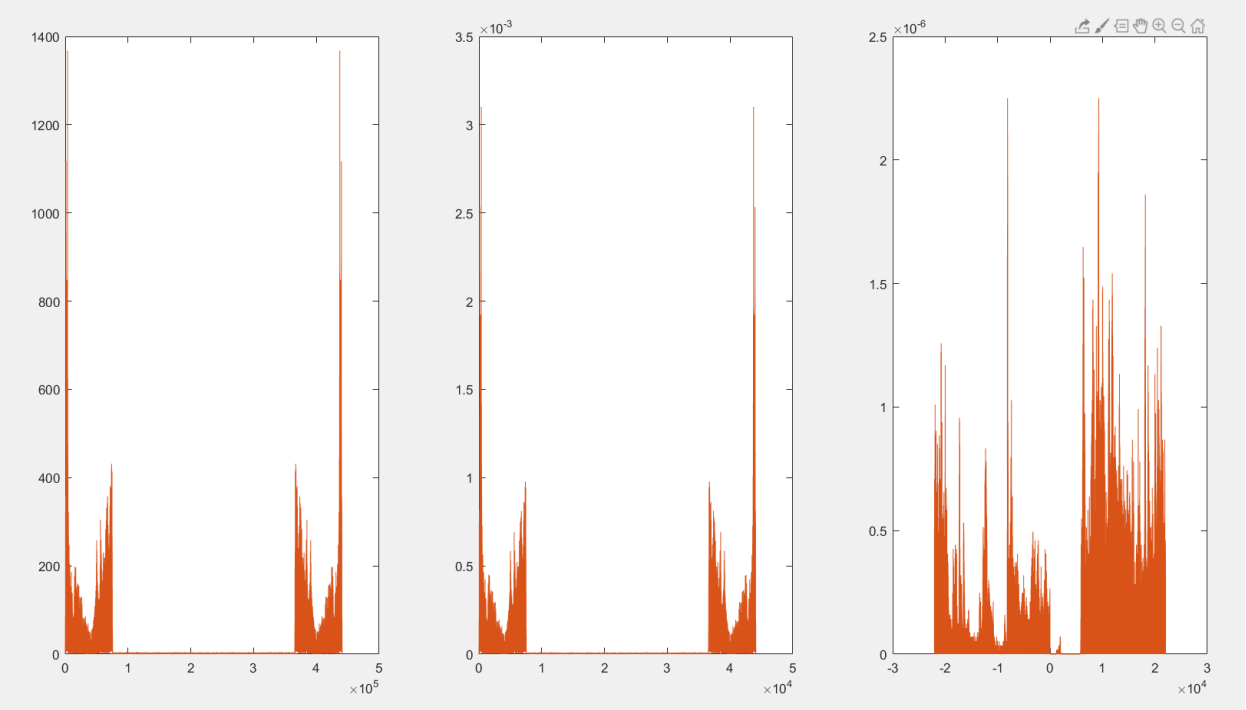
-plotting the audio divided by the length to normalize to ensure proper scaling of the amplitude values in the frequency domain.

-make a vector of length from –(N/2) to N/2 to plot the double sided of the audio in the frequency domain and plot the audio in the frequency domain and shifted to the center of the figure at the origin

-and the single and the double sided plot of the input1 are as follows :

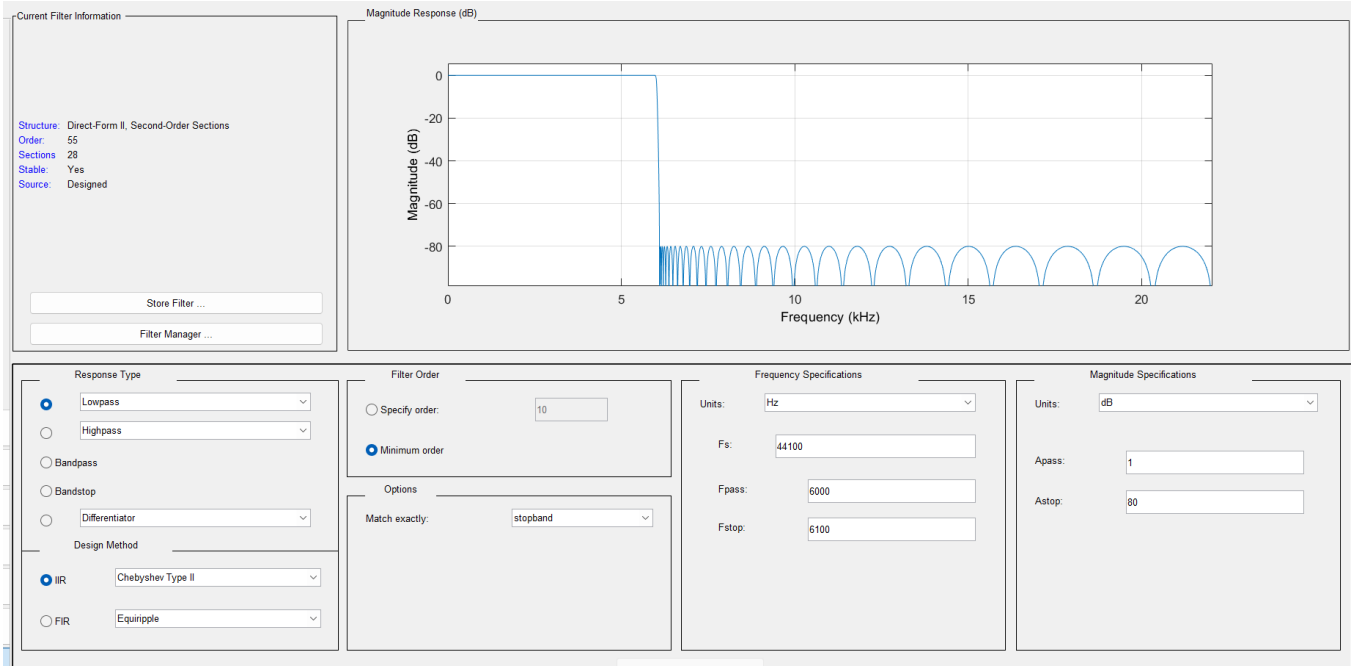


-and the single and the double sided plot of the input2 are as follows :

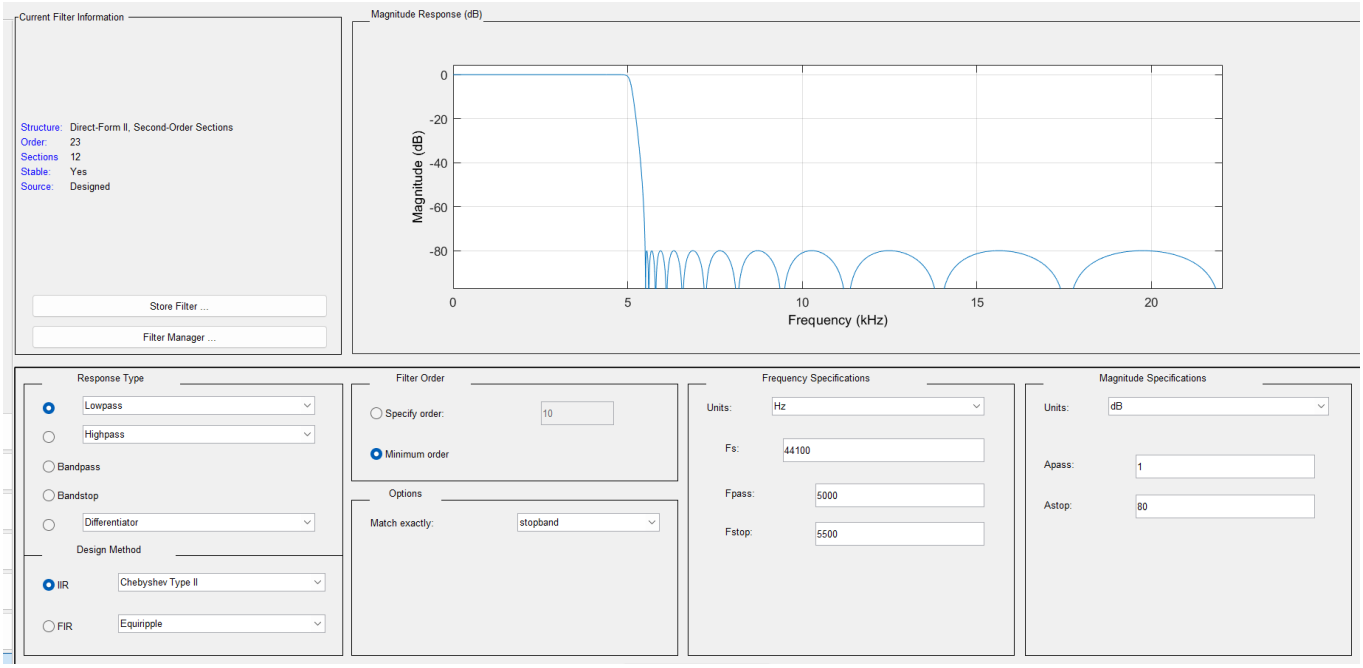


-Limiting the freq of the audio by designing filter to limit the freq by frequency of 7824Hz for the input1 and freq of 7501HZ for imput2

-designing low pass filter which limit the low freq only to cancel all the high freq of the noise of the input1 and the cut off freq of the filter is 6000Hz :

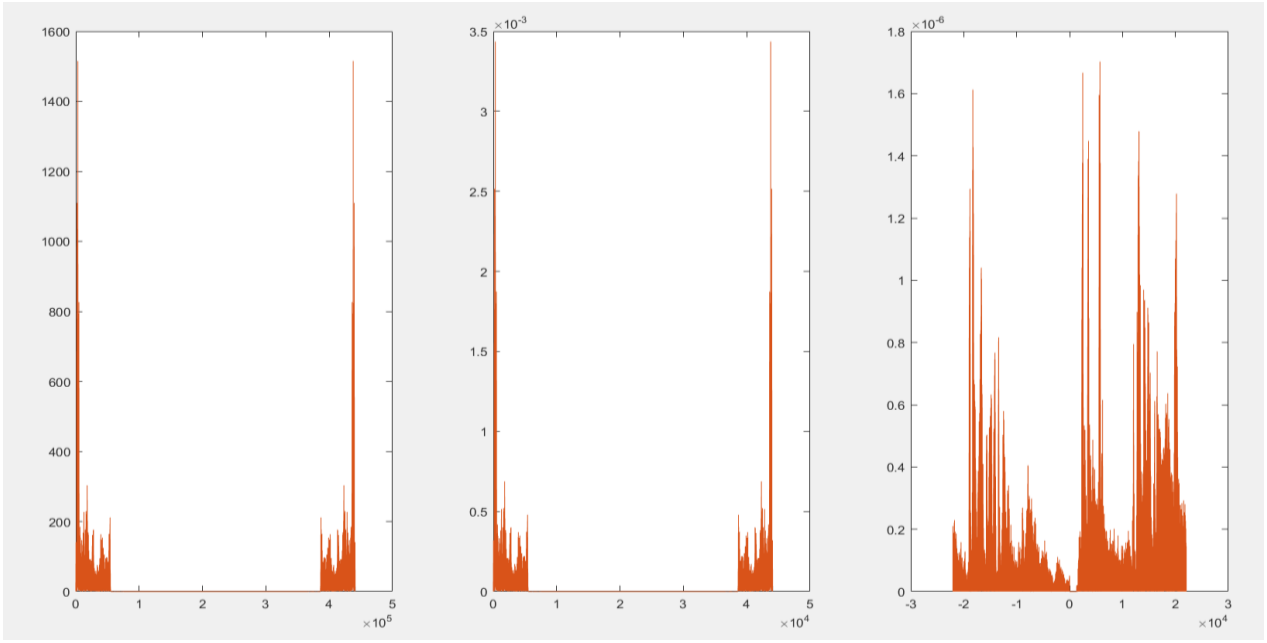


-and this is the low pass of the input2 and the cut off freq of the filter is 5000Hz :

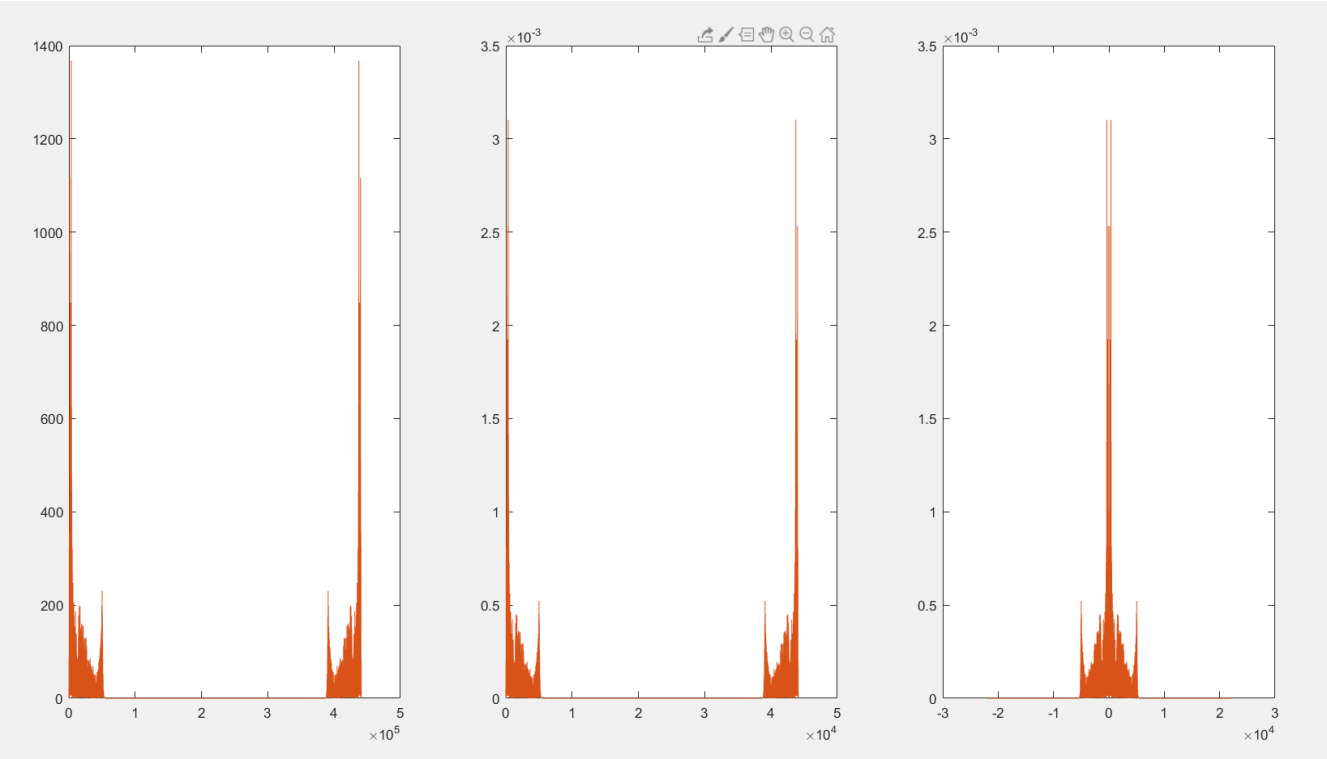


-then I apply the filter on each audio after limiting their frequencies

-Plotting the filtered input1 with Freq domain once with its original amplitude and once the amplitude normalized and once I plot it double sided by fftshift :



-And this the plotting of the filtered input2 :



Task d:

-Read the two audios and store the amplitudes of the audio in a variable and store the sampling freq in another variable the choose the min sampling freq and store it in ‘Fs’ and resample the two audios with the ‘Fs’

-choose the carrier freq of the input1 and it is 145KHz and for the input2 is 100Hz and they are chosen to avoid interference between the two signals while modulation

-defining the time vector from zero to the length of the audio minus one divided by the sampling frequency to be normalized and each step by 1/Fs

-Limiting the freq of each audio by the filters that done in a previous tasks

-modulate each signal by multiplying it by cosine signal that has frequency of the carrier freq which is chosen previously and it will shifted into to two signals at the two sides of the y-axis

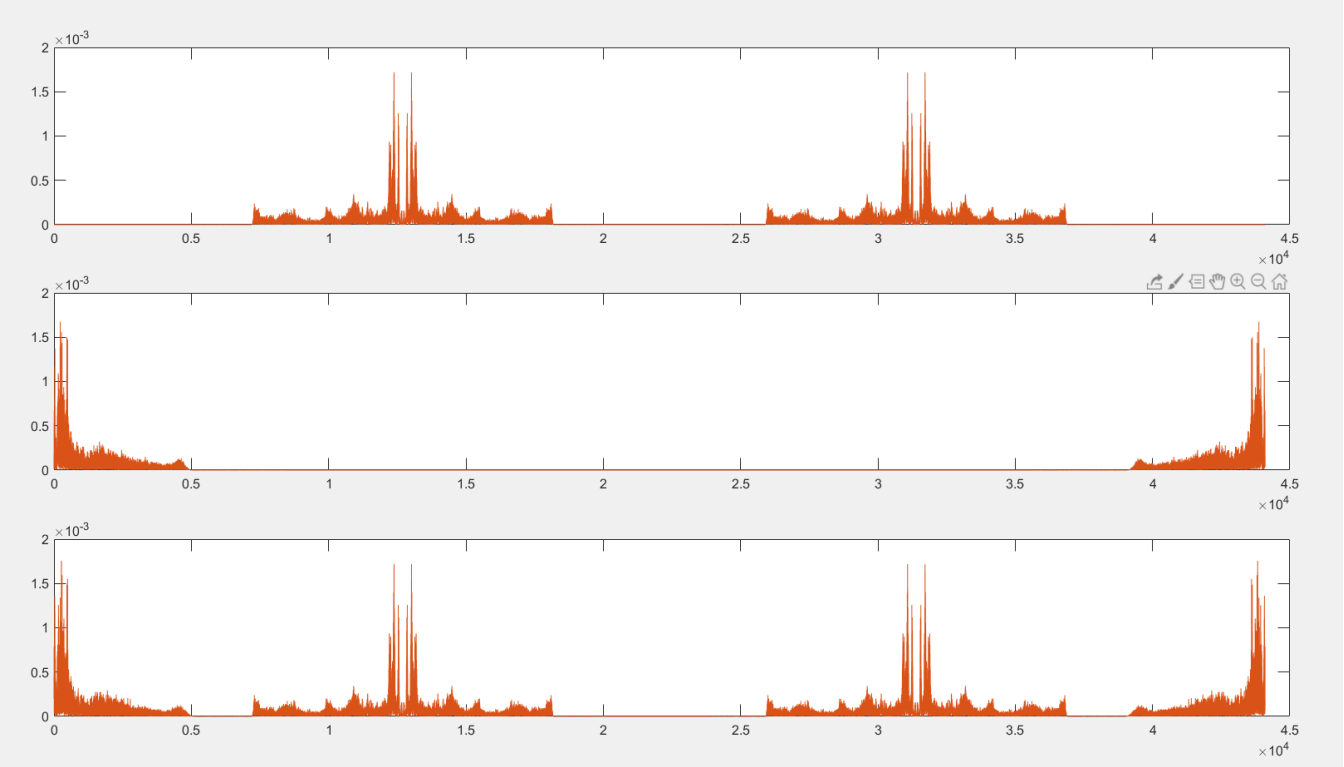
-then I make fourier transform to the signal modulated to plot it

-plotting the modulated signal normalized with the length of the signal

-adding the two signals to make it one signal

-then I make fourier transform to the transmitted signal then plotting it normalized

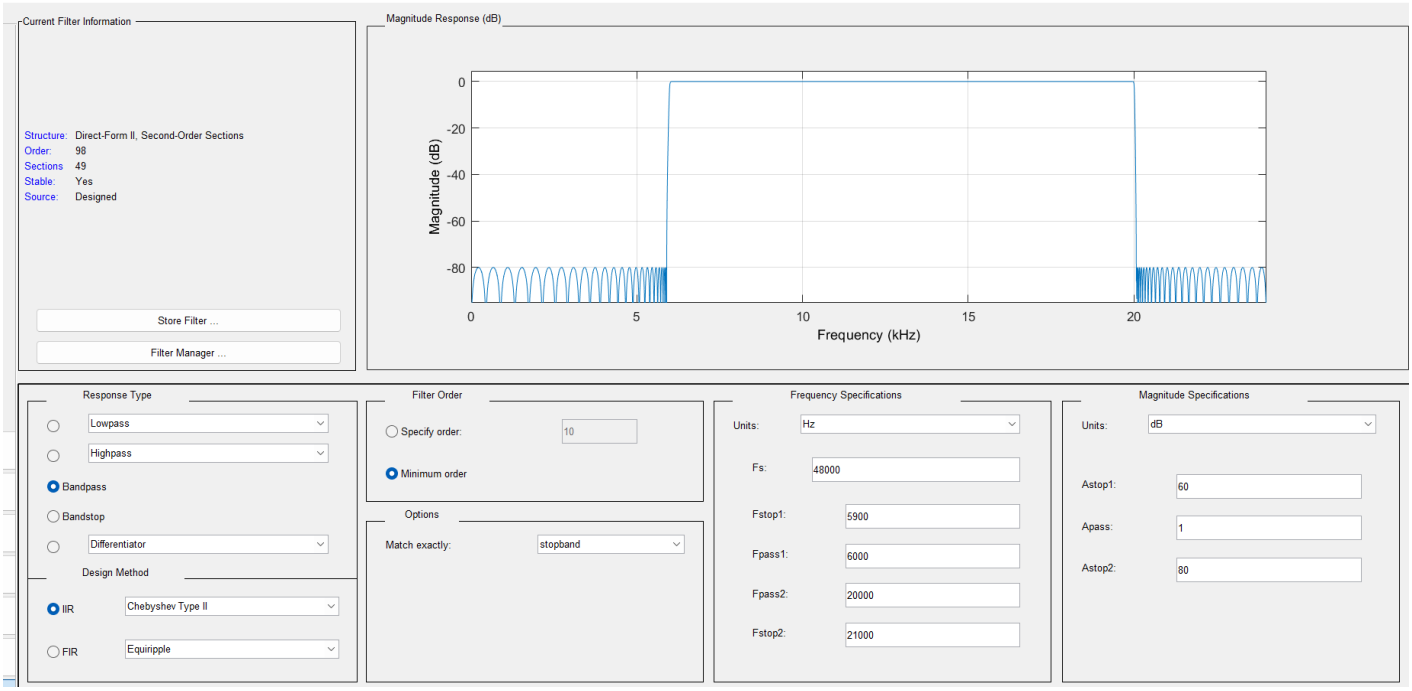
-and these are the signals :



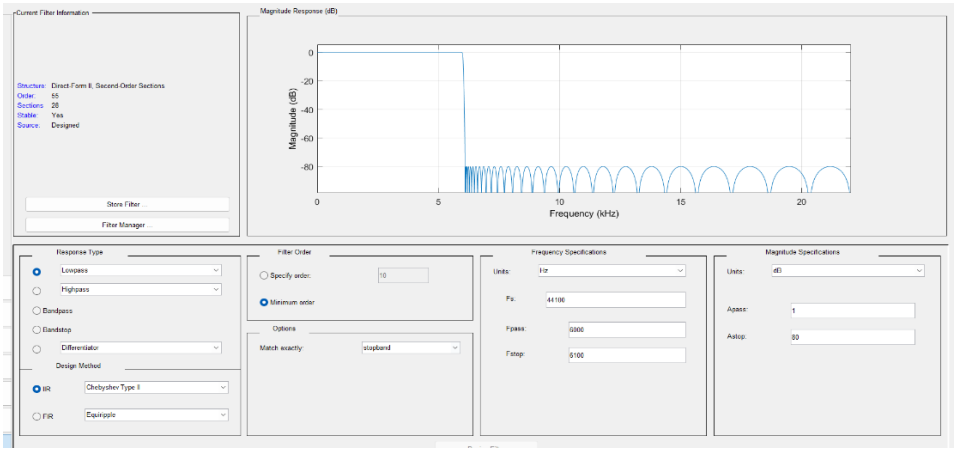
Task e:

-I designed band pass to choose the first signal then I demodulate it by multiplying it by cosine has the carrier freq to shift it to its original location then I apply fourier transform on it

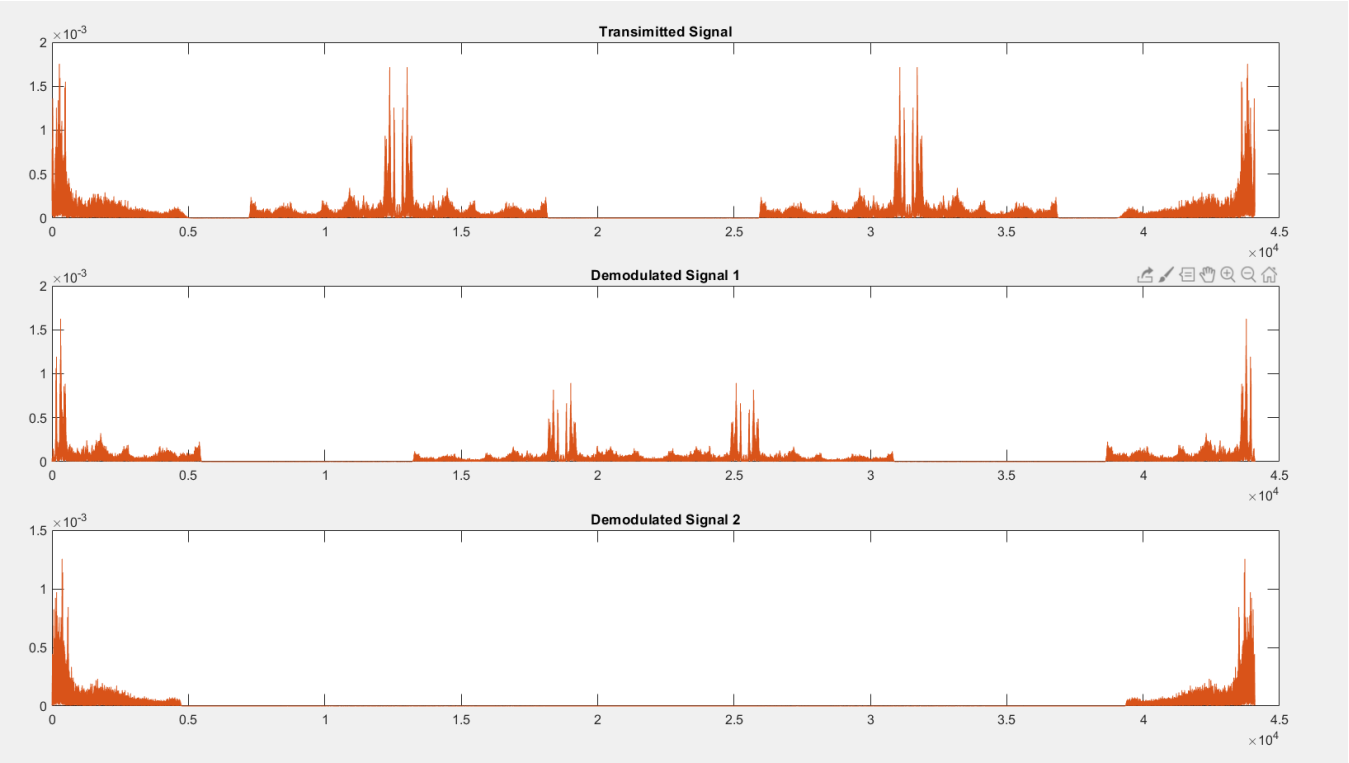
-the band pass passes between 5900 Hz & 20KHz :



- I designed low pass to choose the second signal then I demodulate it by multiplying it by cosine has the carrier freq to shift it to its original location then I apply fourier transform on it , the Low Pass filter has cut off freq 6000Hz :



-Then I plot each of the demodulated signals and each of the signals has noise after demodulation :



Equations in frequency domain:

-lets say that eaxh audio signal is x(t) and I want to modulate it,first I will apply fourier transform on it and apply fourier transform on the cosine signal :

X(ω)=FFT(x(t))

FFT(cos(ω_ct))=$\frac{1}{2}$(ƍ(ω-ω_c)+ƍ(ω+ω_c))

-then I will convolute the two signals to get the modulated signal Y(ω):

Y(ω)=X(ω) * $\frac{1}{2}$(ƍ(ω-ω_c)+ƍ(ω+ω_c))= $\frac{1}{2}$ (X($\omega - \omega_c$) + X($\omega + \omega_c$))

-then the transmitted signal is the summation of the two modulated signals

-then to get ach signal back I will apply Band pass filter to select each signal

-then to get the signal back I will convolute the Y(ω) with the same cosine signal :

X(ω) = Y(ω) * $\frac{1}{2}$(ƍ(ω-ω_c)+ƍ(ω+ω_c))= $\frac{1}{2}$ $(Y(\omega - \omega_c) + Y(\omega + \omega_c))$

-then I apply low pass filter with gain of 2 to select one period of the signal which is at the origin only not shifted

-then apply inverse fourier transform to get x(t):

x(t)=IFFT(X(ω))

Equations in Time domain:

-lets say that each audio signal is x(t) and I want to modulate it,I will multiply it by cosine signal has the carrier frequency to get the modulated signal y(t) :

y(t)=x(t).cos(ω_ct)= $\frac{1}{2}$ $(x(t)e^{i\omega_c t} + x(t)e^{-i\omega_c t})$

-then apply fourier transform on the y(t) signal:

Y(ω)=FFT(y(t))= $\frac{1}{2}$ $(X(\omega - \omega_c) + X(\omega + \omega_c))$

-then I will sum the two modulated signals to make the transmitted signal

-then to get each signal back I will apply Band Pass Filter on the transmitted signal to choose the signal

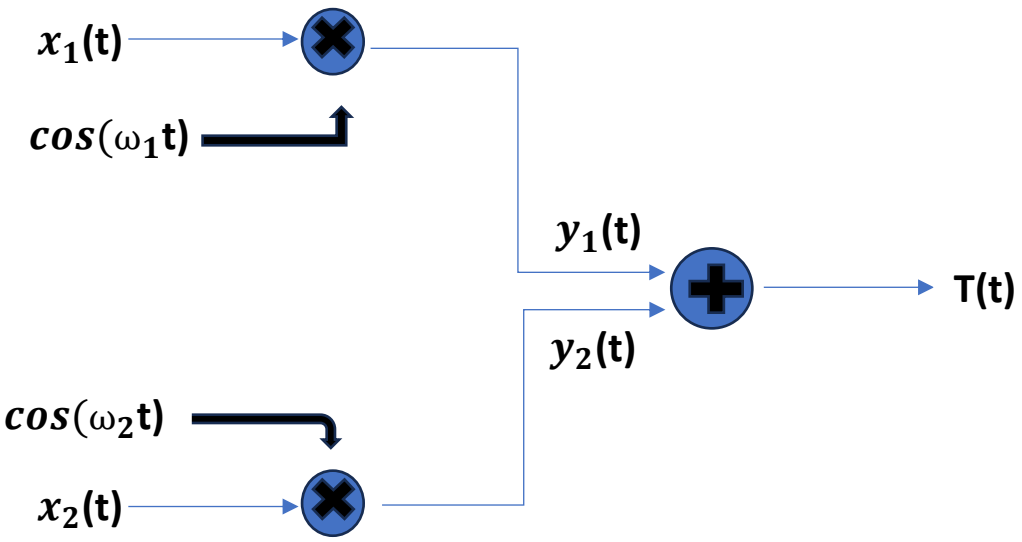
-then I will apply LPF to choose one period which is at origin not shifted

-then apply inverse fourier transform to get x(t):

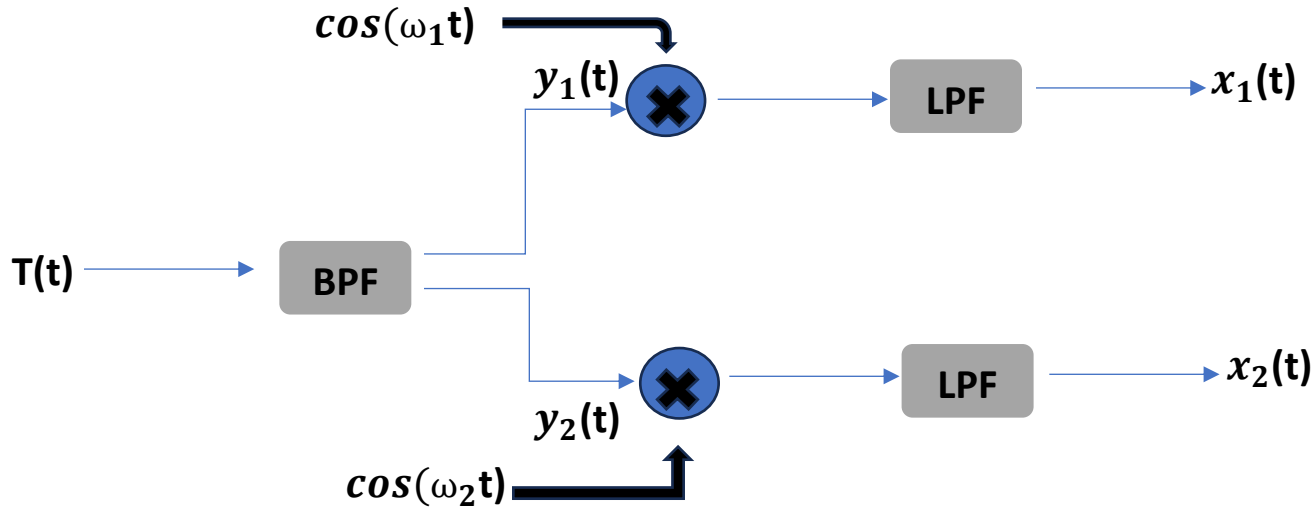
x(t)=IFFT(X(ω))

The Block diagram :

Transmitter :



Receiver :



The Codes :

-Task a :

```
%i will take the image and put it in variable
img_SA = imread('C:\Users\sheha\OneDrive\Desktop\Project
signals\Image\peppers.png');
%making figure to plot the images
figure;
subplot(1, 4, 1);
imshow(img_SA)
title('Original image');
%put the each component of the image in a seperate matrix
Red_Comp_SA = img_SA(:,:,1);
Green_Comp_SA = img_SA(:,:,2);
Blue_Comp_SA = img_SA(:,:,3);
%make a zero matrix with the same size of the original image
a = zeros(size(img_SA, 1), size(img_SA, 2));
%concatinate the red component in a 3x3 matrix with another two zero
matrix to show only the red of the image
Just_Red_Comp = cat(3, Red_Comp_SA, a, a);
subplot(1, 4, 2);
%showing the red component image
imshow(Just_Red_Comp);
title('Red Component');
Just_Green_Comp = cat(3, a, Green_Comp_SA, a);
subplot(1, 4, 3);
imshow(Just_Green_Comp);
title('Green Component');
Just_Blue_Comp = cat(3, a, a, Blue_Comp_SA);
subplot(1, 4, 4);
imshow(Just_Blue_Comp);
title('Blue Component');
```

-Task B.1 :

```
%get the original image as input in a variable
ImagePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\peppers.png';
Original_Image = imread(ImagePath);
%converting the image to the grey scale
Gray_Scale_Image_Version = rgb2gray(Original_Image);
%inserting the values of the horizontal and vertical Sobel kernels in two variables
sobelHorizontal = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
sobelVertical = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
%Convolute the double values of the grey scale of the image with the horizontal and
the vertical
%kernels with the same size of the original image
edgesHorizontal = conv2(double(Gray_Scale_Image_Version), sobelHorizontal, 'same');
edgesVertical = conv2(double(Gray_Scale_Image_Version), sobelVertical, 'same');
%taking the magnitude of the two convoluted images
edgeMagnitude = sqrt(edgesHorizontal.^2 + edgesVertical.^2);
%This line divides each element in the original matrix edgeMagnitude by the maximum
value computed from the entire matrix
edgeMagnitude = edgeMagnitude / max(edgeMagnitude(:));

% Display the original image and the detected edges image
figure;

subplot(1, 3, 1);
imshow(Original_Image);
title('Original Image');

subplot(1, 3, 2);
imshow(Gray_Scale_Image_Version);
title('Original Grayscale Image');

subplot(1, 3, 3);
imshow(edgeMagnitude);
title('Detected Edges (Sobel Operator)');

%saving the image
savePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\';
saveName = 'image1.png';
fullFilePath = fullfile(savePath, saveName);
imwrite(edgeMagnitude,fullFilePath);
```

Task B.2 :

```
imagePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\peppers.png';
Orignal_Image = imread(imagePath);

% Define the Laplacian sharpening kernel
laplacianKernel = [0, -1, 0; -1, 5, -1; 0, -1, 0];

%seperate each component in a seperate variable
Red_Comp_SA = Orignal_Image(:,:,1);
Green_Comp_SA = Orignal_Image(:,:,2);
Blue_Comp_SA = Orignal_Image(:,:,3);



% Apply convolution to each component of the image to sharpen the image
sharpenedImage(:,:,1) = conv2(double(Red_Comp_SA), laplacianKernel, 'same');
sharpenedImage(:,:,2) = conv2(double(Green_Comp_SA), laplacianKernel, 'same');
sharpenedImage(:,:,3) = conv2(double(Blue_Comp_SA), laplacianKernel, 'same');


% Display the original image and the sharpened image
figure;

subplot(1, 2, 1);
imshow(Orignal_Image);
title('Original Grayscale Image');

subplot(1, 2, 2);
imshow(uint16(sharpenedImage));
title('Sharpened Image (Laplacian Kernel)');

%saving the image
savePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\';
saveName = 'image2.png';
fullFilePath = fullfile(savePath, saveName);
imwrite(uint16(sharpenedImage),fullFilePath);
```

Task B.3:

```
% Read image
imagePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\peppers.png';
Original_Image = imread(imagePath);


% Define a blurring kernel (e.g., a 5x5 averaging filter)
blurKernel = ones(15)/(15^2);

Red_Comp_SA = Original_Image(:,:,1);
Green_Comp_SA = Original_Image(:,:,2);
Blue_Comp_SA = Original_Image(:,:,3);

% Apply convolution to blur the image

 Blured_Image_Red_Comp_SA = conv2(double(Red_Comp_SA), blurKernel, 'same');
 Blured_Image_Green_Comp_SA = conv2(double(Green_Comp_SA), blurKernel, 'same');
 Blured_Image_Blue_Comp_SA = conv2(double(Blue_Comp_SA), blurKernel, 'same');

 Blured_Image = cat(3, Blured_Image_Red_Comp_SA, Blured_Image_Green_Comp_SA,
Blured_Image_Blue_Comp_SA);



% Convert the result back to uint16 for display
Blured_Image = uint16(Blured_Image);


% Display the original and blurred images
figure;

subplot(1, 2, 1);
imshow(Original_Image);
title('Original Image');

subplot(1, 2, 2);
imshow(Blured_Image);
title('Blurred Image');

%saving the image
savePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\';
saveName = 'image3.png';
fullFilePath = fullfile(savePath, saveName);
imwrite(Blured_Image,fullFilePath);
```

Task B.4 & C: [1]

```
% Read an example grayscale image
imagePath ='C:\Users\sheha\OneDrive\Desktop\Project signals\Image\peppers.png';
Original_Image = imread(imagePath);

% Horizontal Motional Blur kernel
horz_blur = zeros(21, 21);
horz_blur = insertShape(horz_blur, 'Line', [21, 10, 0, 10], 'Color', [255 255 255],
'LineWidth', 1);
horz_blur = double(rgb2gray(horz_blur));

% Normalize the kernel
horz_blur = horz_blur / sum(horz_blur(:));

%seperate the components of the images
Red_Comp_SA = Original_Image(:,:,1);
Green_Comp_SA = Original_Image(:,:,2);
Blue_Comp_SA = Original_Image(:,:,3);

% Apply convolution to blur the image
Motion_Blured_Image_Red_Comp_SA = conv2(double(Red_Comp_SA), horz_blur );
Motion_Blured_Image_Green_Comp_SA = conv2(double(Green_Comp_SA), horz_blur);
Motion_Blured_Image_Blue_Comp_SA = conv2(double(Blue_Comp_SA), horz_blur);
 %concatenate the image components in one image
Motion_Blured_Image = cat(3, Motion_Blured_Image_Red_Comp_SA,
Motion_Blured_Image_Green_Comp_SA, Motion_Blured_Image_Blue_Comp_SA);



% Convert the result back to uint16 for display
 Motion_Blured_Image = uint16(Motion_Blured_Image);


% Display the original and blurred images
figure;

subplot(1, 2, 1);
imshow(Original_Image);
title('Original Image');

subplot(1, 2, 2);
imshow(Motion_Blured_Image);
title('Motion Blurred Image');

%saving the image
savePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\';
saveName = 'image4.png';
fullFilePath = fullfile(savePath, saveName);
imwrite(Motion_Blured_Image,fullFilePath);
```

```
%------------------------------- The Unbluring process----------------

%Fourier transform for the kernel and the blured image
horz_blur_FT=fft2(horz_blur,size(Motion_Blured_Image,1),size(Motion_Blured_Image,2));

Motion_Blured_Image_ft=fft2(Motion_Blured_Image);
%defining constant to add it to each element in the kernel to avoid
%dividing by zero
C=.00009;
%divide each element in the blured image by each element in the kernel to
%get the original image in the frequency domain
original_image_ft =Motion_Blured_Image_ft ./(horz_blur_FT+C);
%restore the original image by making invesre fourier transform for the
%image to get it in the time domaine then take the magnitude of it
Restored_image = abs(ifft2(original_image_ft));
%modify the image to be in the same size as the original image
Restored_image
=Restored_image(1:size(Original_Image,1),1:size(Original_Image,2),1:size(Original_Ima
ge,3));




% Display the original and Unblurred images
figure();

subplot(1, 2, 1);
imshow(Original_Image);
title('Original Image');

subplot(1, 2, 2);
imshow(uint16(Restored_image));
title('Unblurred Image');

%saving the restored image
savePath = 'C:\Users\sheha\OneDrive\Desktop\Project signals\Image\';
saveName = 'image5.png';
fullFilePath = fullfile(savePath, saveName);
imwrite(uint16(Restored_image),fullFilePath);
```

II. Communication system simulation:

Task a :

```
fs=44100; %sampling frequency ,it is commonly used with audio
bit_depth=16; %most common with audio
record_length =10; %record length in seconds
channel =2; %choosing number of channels (stereo mode)
recorder=audiorecorder(fs,bit_depth,channel);
disp('Start speaking..')
%Record audio
recordblocking(recorder,record_length);
disp('End of Recording.');
audio_data=getaudiodata(recorder,'uint8');%recorded audio data
audiowrite('input1.wav',audio_data,fs);%audio file
```

Task b , c :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Working on the first audio%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load Hd_1.mat
load Hd_1_1.mat

%%%%%%%%%%%%%%%%%%%%%%Reading the original audio%%%%%%%%%%%%%%%%%%%
[amp, Fs] = audioread("C:\Users\Ali Mohamed\Downloads\Project2\input1.wav"); %get the
file
om=audioplayer(amp,Fs);   %make orignal audio object
N=length(amp);   %get the amplitude length
Original_Audio_Freq=fft(amp,N);   %frequency shift the filter
%%%%%%%%%%%%%%%%%%%%%%Plotting the original audio%%%%%%%%%%%%%%%%%%%%%%
%Single sided with k
figure();       %opens a figure
k=0:N-1;        %calculating k
subplot(1,3,1); %divide it into half and choose the first half
plot(k,abs(Original_Audio_Freq));%draw the orignal audio
pause(3);       %wait 3 seconds
%Single sided with frequencyf
F_1=(0:N-1)*Fs/N;     %calculate the frequency to plot it
subplot(1,3,2);       %divide it into half and choose the first half
plot(F_1,abs(Original_Audio_Freq)/N);  %draw the orignal audio
%Double sided with frequency
F_2=(-N/2:N/2-1)*Fs/N;         %center the frequency to plot it
subplot(1,3,3);                %divide it into half and choose
plot(F_2,abs(fftshift(fft(amp)))/N);%draw the orignal audio


% %%%%%%%%%%%%%%%%%%%%Limit the maximum frequency%%%%%%%%%%%%%%%%%%
Limited_Freq_Audio=filter(Hd_1,amp);        %Limit the frequency the audio

% %%%%%%%%%%%%%%%%%%%%Getting the filtered audio%%%%%%%%%%%%%%%%%%
```

```
filtered_audio=filter(Hd_1_1,Limited_Freq_Audio);        %filter the audio
fm=audioplayer(filtered_audio,Fs);   %make filter audio object
Filered_Audio_Freq=fft(filtered_audio,N);


% %********************Plotting the Filtered audio*********************
% %Single sided with k
figure();
subplot(1,3,1); %second half
plot(k,abs(Filered_Audio_Freq));%draw the filter audio
%Single sided with frequency
subplot(1,3,2);     %second half
plot(F_1,abs(Filered_Audio_Freq)/N);  %draw the filter audio
pause(3);           %wait 3 seconds
%Double sided with frequency
subplot(1,3,3);              %second half
plot(F_2,abs(fftshift(fft(filtered_audio)))/N);%draw the filter audio

audiowrite("output1_filtered.wav",filtered_audio,fs); %save the filtered file



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%*******************************Working on the Second audio***************
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load Hd_2.mat
%*********************Reading the original audio*******************
[amp, Fs] = audioread("C:\Users\Ali Mohamed\Downloads\Project2\input2.wav"); %get the
file
om=audioplayer(amp,Fs);   %make orignal audio object
N=length(amp);   %get the amplitude length
Original_Audio_Freq=fft(amp,N);   %frequency shift the filter
%*********************Plotting the original audio**********************
%Single sided with k
figure();       %opens a figure
k=0:N-1;        %calculating k
subplot(1,3,1); %divide it into half and choose the first half
plot(k,abs(Original_Audio_Freq));%draw the orignal audio
pause(3);       %wait 3 seconds
%Single sided with frequencyf
F_1=(0:N-1)*Fs/N;     %calculate the frequency to plot it
subplot(1,3,2);       %divide it into half and choose the first half
plot(F_1,abs(Original_Audio_Freq)/N);  %draw the orignal audio
%Double sided with frequency
F_2=(-N/2:N/2-1)*Fs/N;         %center the frequency to plot it
subplot(1,3,3);                %divide it into half and choose
plot(F_2,abs(fftshift(fft(amp)))/N);%draw the orignal audio


% %*********************Limit the maximum frequency******************
Limited_Freq_Audio=filter(Hd_1,amp);        %Limit the frequency the audio

% %*********************Getting the filtered audio******************
filtered_audio=filter(Hd_2,Limited_Freq_Audio);        %filter the audio
```

```
fm=audioplayer(filtered_audio,Fs);   %make filter audio object
Filered_Audio_Freq=fft(filtered_audio,N);


% %********************Plotting the Filtered audio*********************
% %Single sided with k
figure();
subplot(1,3,1); %second half
plot(k,abs(Filered_Audio_Freq));%draw the filter audio
%Single sided with frequency
subplot(1,3,2);     %second half
plot(F_1,abs(Filered_Audio_Freq)/N);  %draw the filter audio
pause(3);           %wait 3 seconds
%Double sided with frequency
subplot(1,3,3);             %second half
plot(F_2,abs(fftshift(fft(filtered_audio)))/N);%draw the filter audio

audiowrite("output2_filtered.wav",filtered_audio,fs); %save the filtered file
```

Task e ,d :

```
[amp_1, Fs_1] = audioread("C:\Users\Ali Mohamed\Downloads\Project2\input1.wav");

[amp_2, Fs_2] = audioread("C:\Users\Ali Mohamed\Downloads\Project2\input2.wav");

load Hd_1.mat
load Hd_1_1.mat
load Hd_2.mat
load Band_Pass.mat
load Low_Pass.mat

% Ensure both signals have the same sampling rate
Fs = min(Fs_1, Fs_2);
amp_1 = resample(amp_1, Fs, Fs_1);
amp_2 = resample(amp_2, Fs, Fs_2);

% Perform amplitude modulation
carrier_frequency1 = 145000;  % Choose a suitable carrier frequency for signal 1
carrier_frequency2 = 100; % Choose a suitable carrier frequency for signal 2

t = 0:1/Fs:(length(amp_1)-1)/Fs;

%********************Getting the filtered audio******************
amp_1_Limited_freq=filter(Hd_1_1,amp_1);        %filter the audio


% %********************Getting the filtered audio******************
amp_2_Limited_freq=filter(Hd_2,amp_2);        %filter the audio


% Modulate signal 1
modulated_signal1 = amp_1_Limited_freq .* cos(2*pi*carrier_frequency1*t)';

N_1=length(modulated_signal1);   %get the amplitude length
```

```
modulated_signal1_Freq=fft(modulated_signal1,N_1);
%Single sided with frequency
figure();
F_1=(0:N_1-1)*Fs/N_1;     %calculate the frequency to plot it
subplot(3,1,1);       %divide it into half and choose the first half
plot(F_1,abs(modulated_signal1_Freq)/N_1);  %draw the orignal audio


% Modulate signal 2
modulated_signal2 = amp_2_Limited_freq .* cos(2*pi*carrier_frequency2*t)';


N_2=length(modulated_signal2);   %get the amplitude length
%Single sided with frequency
modulated_signal2_Limited_freq_Freq=fft(modulated_signal2,N_2);
subplot(3,1,2);       %divide it into half and choose the first half
plot(F_1,abs(modulated_signal2_Limited_freq_Freq)/N_2);  %draw the orignal audio



% Combine the modulated signals
transmitted_signal = modulated_signal1 + modulated_signal2;


N_3=length(transmitted_signal);   %get the amplitude length
transmitted_signal_Audio_Freq=fft(transmitted_signal,N_3);
%********************Plotting the original audio**********************
%Single sided with frequencyf
subplot(3,1,3);       %divide it into half and choose the first half
plot(F_1,abs(transmitted_signal_Audio_Freq)/N_3);  %draw the orignal audio



%**********************Demodulation steps*****************************

% Demodulate the first signal
demodulated_signal1 = filter(Band_Pass, transmitted_signal);
demodulated_signal1 = demodulated_signal1 .* cos(2*pi*carrier_frequency1*t)';
demodulated_signal1_Freq=fft(demodulated_signal1,N_3);


% Demodulate the second signal
demodulated_signal2 = filter(Low_Pass, transmitted_signal);
demodulated_signal2 = demodulated_signal2 .* cos(2*pi*carrier_frequency2*t)';
demodulated_signal2_Freq=fft(demodulated_signal2,N_3);


% Plot the demodulated signals
figure;

subplot(3, 1, 1);
plot(F_1, abs(transmitted_signal_Audio_Freq)/N_3);
title('Transimitted Signal');

subplot(3, 1, 2);
```

```
plot(F_1, abs(demodulated_signal1_Freq)/N_3);
title('Demodulated Signal 1');

subplot(3, 1, 3);
plot(F_1, abs(demodulated_signal2_Freq)/N_3);
title('Demodulated Signal 2');


audiowrite("output1.wav",demodulated_signal1,Fs); %save the filtered file
audiowrite("output2.wav",demodulated_signal2,Fs); %save the filtered file
```

References :

[1] https://medium.com/@itberrios6/how-to-apply-motion-blur-to-images-75b745e3ef17#:~:text=The%20horizontal%20blur%20kernel%20(kernel,appear%20to%20be%20motion%20blurred