Signal Project

Presented for ELC2030 Project
Presented to:
Dr.Michael Melek Abdel-Sayed

ID	Section	Name
9220640	3	محمد إبراهيم يوسف الدميري
9220849	4	مصطفى محمد مصطفى السلكاوي

CONTENTS

List of figures for image part

List of figures for audio part

Image part

Problem A:

Problem:

It is required to:

- Read the image.
- Extract its three color components.
- Plot 3 components and original one in one figure.

Approach:

- Read our image using imread() built in function.
- Extract each color component.
- Merge each component with 2d zeros matrix using cat() to can plot them.
- Plot 3 color components and original one using subplot(n,m,p) to divide. grid in n*m cells, and using imshow() to plot them .

Result:

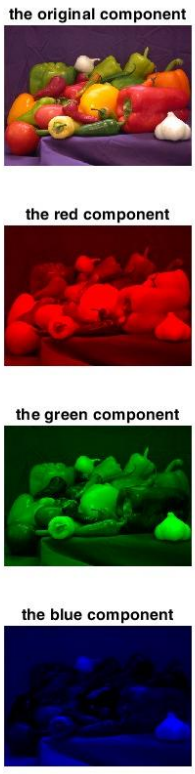


Figure 1: Result of problem A

Problem B

Problem:

It is required to perform the following operations on the image using convolution:

- Edge detection: detect and display the edges in the image
- Image sharpening: enhance the image details and edges.
- Blurring (averaging): hide the fine details in the image.
- Motion blurring: simulate motion blur in the horizontal direction.

Approach:

Edge detection:

Choosing suitable kernel for horizontal and vertical edges.

We choose vertical edges to be: $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

And horizontal edges to be: $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

Why we use these kernels?

We use prewitt filters. It is one of the most famous filters in image processing using to detect the orientation and edges.

How does this kernel work?

These Prewitt kernels are convolved with the image to calculate the gradient intensity at each pixel. Where, the horizontal kernel computes the changes in the gradient intensity from left to right, i.e., along the horizontal direction. Whereas, the vertical kernel computes the variations in the gradient intensity from top to bottom i.e., along the vertical direction. Finally, the two gradient images are combined to get the resulting image with highlighted edges.

- Convert our images to double format and normalize them using im2double() function.
- Convolve 3 image component with our vertical and horizontal kernel.

- Merge the 3 channel of horizontal and vertical images using cat() function and take magnitude of the to obtain the final image.

- Plot the edge detected image using imshow() function.

Image sharping:

Choosing suitable kernel for image sharping

We choose the kernel $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Why we use this kernel?

We use lablacian kernel $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
but we change the Center entry to 9 to make the summation of all entries =1 to high the brightness of the image.

How does lablacian filter sharpen the image and detect its edges?

Lablacian filter enhances the high frequency components of an image by calculating the second derivatives of the image intensity at each pixel by make convolution between each pixel and the kernel .the kernel coefficients is designed to approximate the second derivatives.

- Convolve 3 image components with our kernel
- Merge the 3 channels of sharpen image using cat() function and plot it using imshow() function

Image blurring:

Choosing suitable kernel for image blurring.

We choose the kernel to be $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Why we use this filter?

It is “box blur” filter which is very popular filter in image processing area to blur an image.

How does this filter work?

The resulting image after convolution with this filter is an image each pixel in the average of its neighboring pixels.

- Convolve 3 image components with our kernel.
- Merge the 3 channels of blurred image using cat() function and plot it using imshow() function.

Motion blurring:

Choosing suitable kernel for image blurring.

We choose the kernel to be (15*15) matrix with all zeros except the row number 8 to equal $\frac{1}{15}$.

Why we use this filter?

This filter will spread the pixel of the image in horizontal direction which Will simulate the motion effect. By taking the average of horizontal row and put it in center pixel.

- Convolve 3 image color components with our kernel.

- Merge the 3 channels of moved blurred image using cat() function and plot it using imshow() function.

Results:

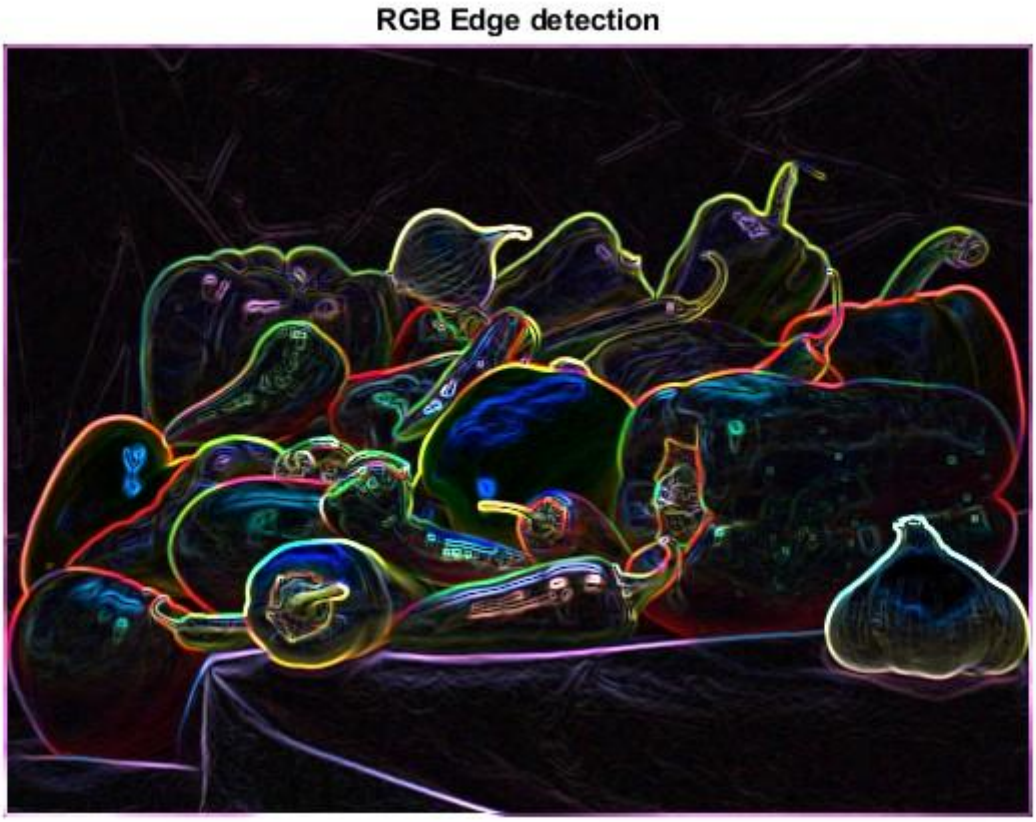


Figure 2: RGB Edge detected image

Figure 3: RGB Sharp image

Figure 4: RGB Blurred image

Figure 5: RGB Motion blurring image

Problem C

Problem:

It is required to restore the original image from the motion-blurred image.

Approach:

- Transform the 3 color components to frequency domain using fft () function.
- Transform the kernel to frequency domain and adjust its size to image size.
- Operate division operation between each transformed image color component and transformed kernel. The outputs of these operations are 3 color components of original image in frequency domain.
- Transform the outputs to space domain.
- Merge the 3 channels of theimage using cat() function and plot it using imshow() function.

Results:

Figure 6: Restored image

Audio part

Problem A:

Problem:

Using Matlab, record two segments of your voice of about 10 seconds each. Select appropriate
values for sampling frequency and bit depth. Justify your choice of such values. Save the audio
files as ‘input1.wav’ and ‘input2.wav’.

Explanation:

- first, we make a separate file to record the first audio file using audiorecorder() function and we selected a suitable Fs and bit depth, we will show why we selected these values, and make the same with the second audio file.
- we used the function recordblocking() to determine the duration of recording (10 seconds).
- then we saved them using audiowrite() function.
- put Fs (sampling frequency) = 44100 Hz
- put bit depth = 16 bits

Why we choose Fs = 44100 Hz?

We used it as it is commonly used in audio applications, such as music recording and playback, and it satisfies the requirements of the human auditory system and provides a good balance between audio quality and data size.

Why we choose bit depth = 16 bits?

We used it because it offers a good balance between dynamic range and file size. Also 16 bits give [(2^16) - 1] amplitude levels, this range is more than adequate for most audio applications, as it covers the full range of human hearing and provides sufficient resolution to accurately capture and reproduce audio signals.

Problem B:

Problem:

Limit the maximum frequency of both signals to a suitable value. Design a LPF filter to filter each of the signals. You should test different cutoff frequencies, listen to the filtered audio, and select an appropriate value such that the quality of audio is not significantly affected. Plot the frequency response of the filter [Lecture 10].

Explanation:

- first we will get the length of the signal, (length of the first signal equals the length of the second signal) using the function length(), determine the frequency we will show the signal on:

  F = (-N/2 : N/2 - 1) * fs / N,

  it is from -N/2 to N/2 -1 as we will shift it to symmtric around y-axis.

- we will get the fft (fast fourir transform) of the signal to represent it in frequency domain, and determine the suitable range that will be used in filterDesigner (Fpass and Fstop). They will be like that:

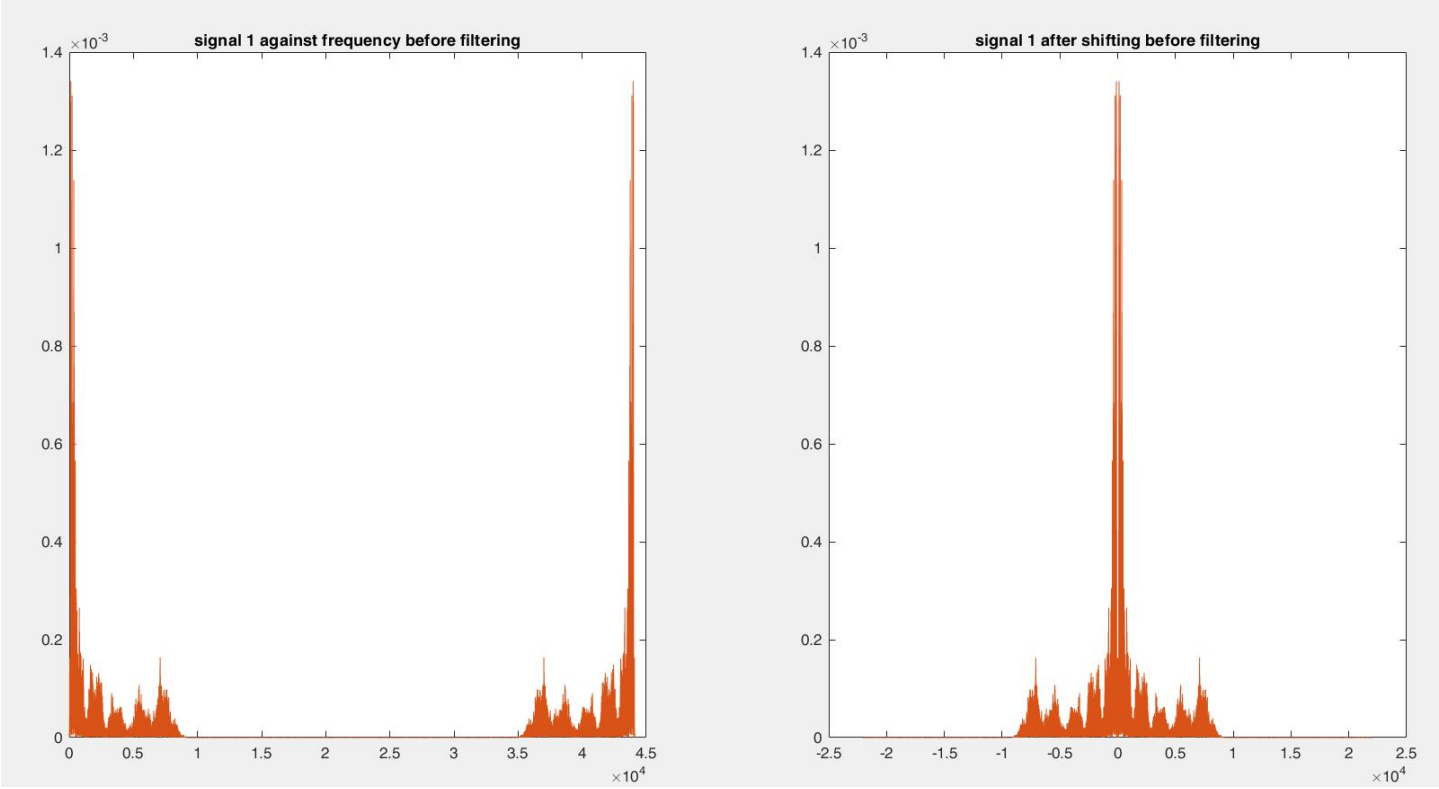


Figure 1: the first signal in frequency domain

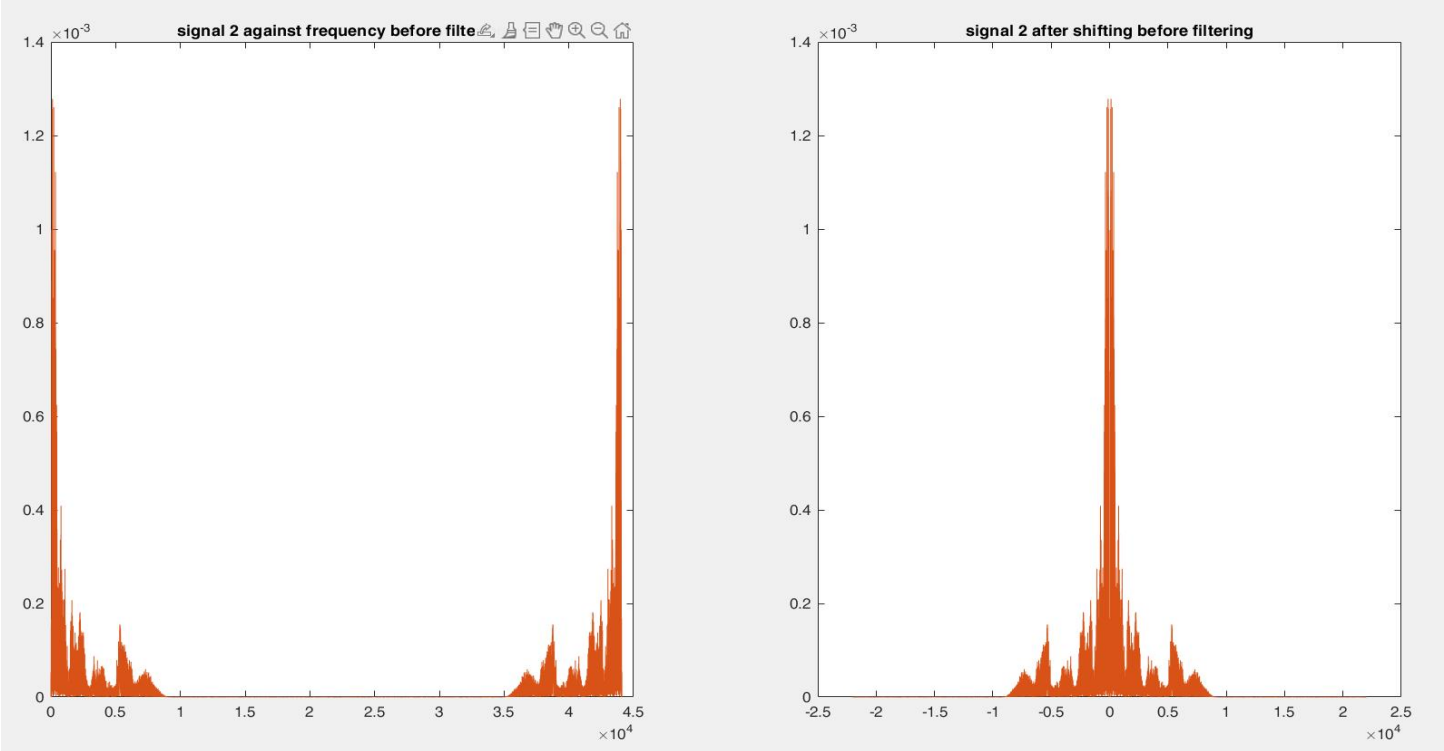


Figure 2: the second signal in frequency domain

- then we determined (from the two figures) the suitable frequency for the filter, and as the two signal has the same bandwidth, we will make a single filter for both of them.
- after we try the suitable Fpass and Fstop that will keep the quality of the audio , they were : Fpass = 3500 Hz, Fstop = 4000 Hz.

The following figure shows all info about Hd low pass filter:

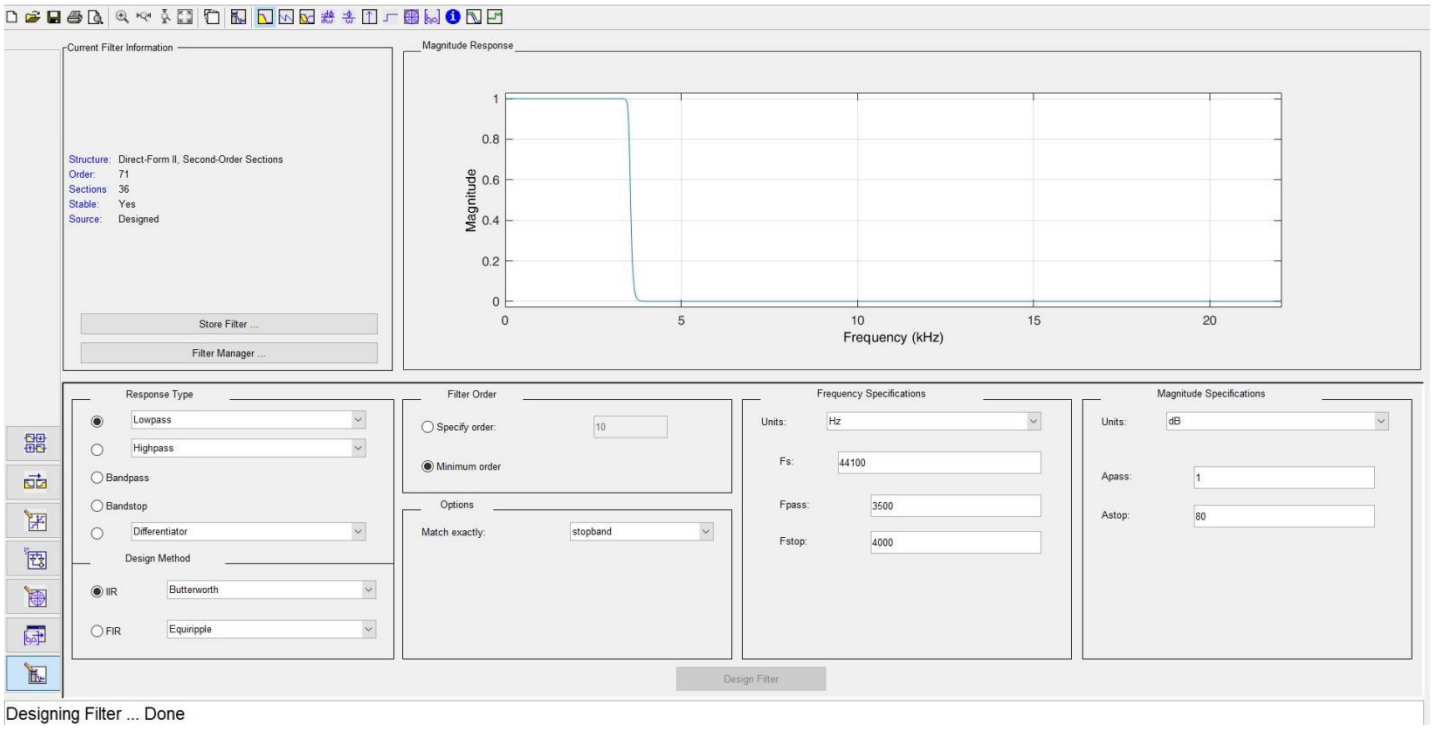


Figure 3: Hd low pass filter

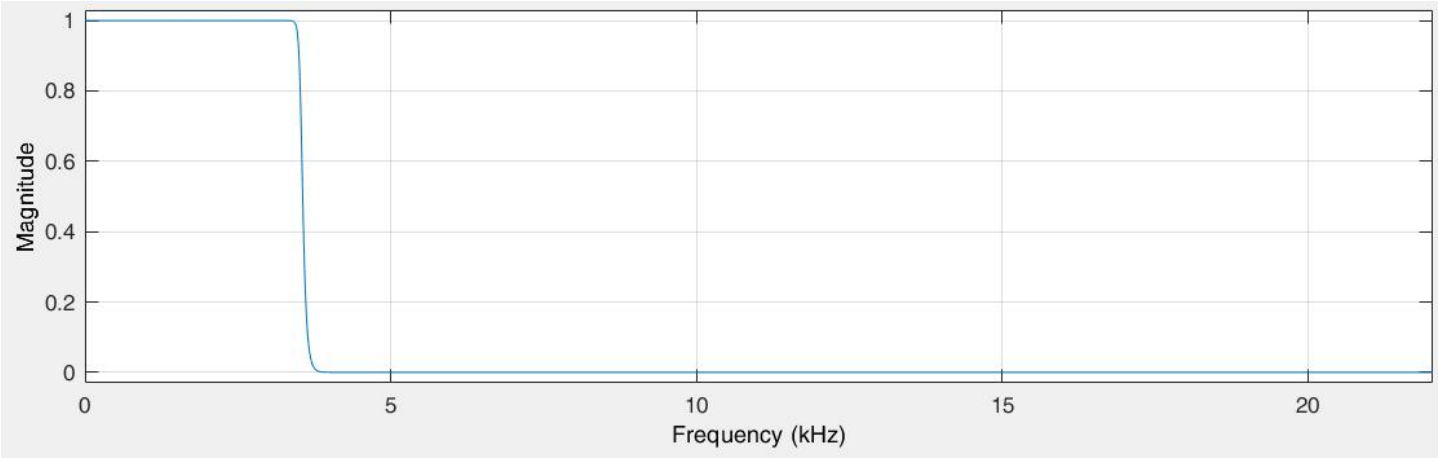


Figure 4: magnitude of Hd filter against frequency

- Then we make a variable for each filtered audio and get the fft of them using fft function.

Problem C:

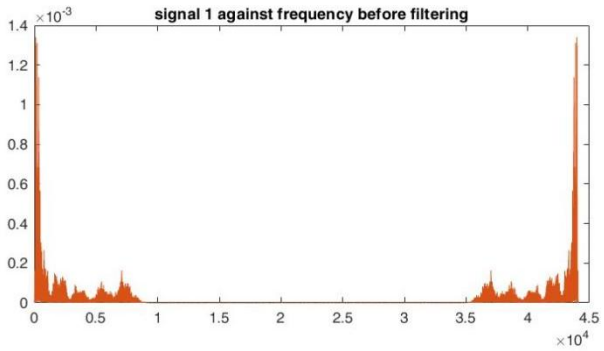
Problem:

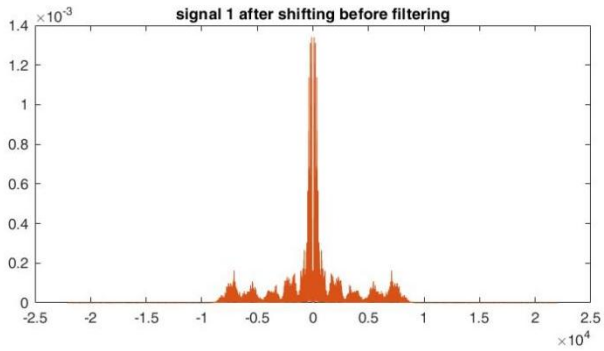
Plot the magnitude spectrum of both signals before and after filtering against the frequency in Hz using FFT [Project lecture]. Use the function 'fftshift' to make the zero frequency in the center of the plot.

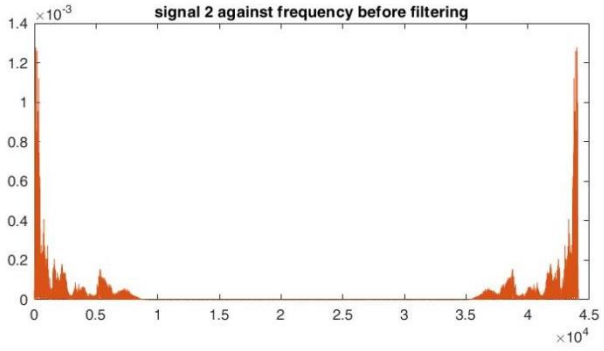
Explanation:

- first we will convert our two signals by fft, then from the previous problem we have the filtered signals and fft of the filtered signals. Then all we need is to plot them.
- for ploting we will divide the screen by the function subplot(), then use fftshift to make the zero frequency in the center of the plot.

Then after we did that, the results were the figures below:







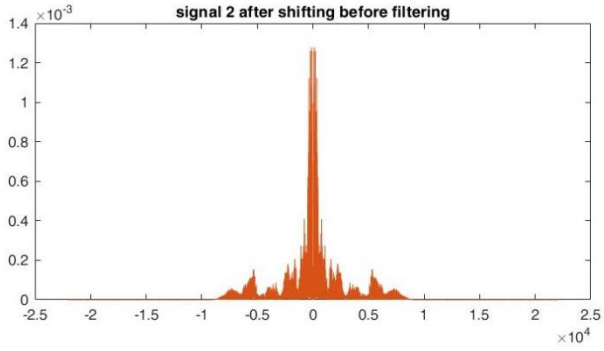
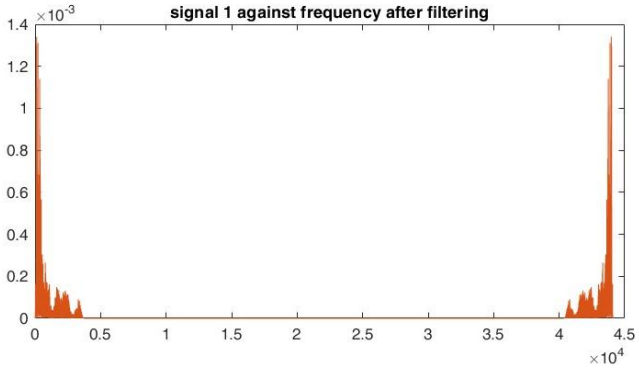
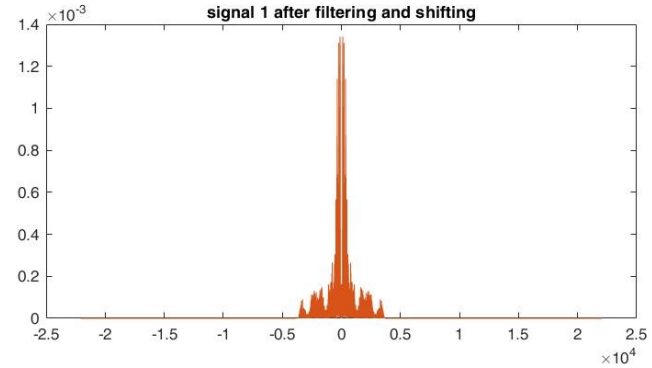
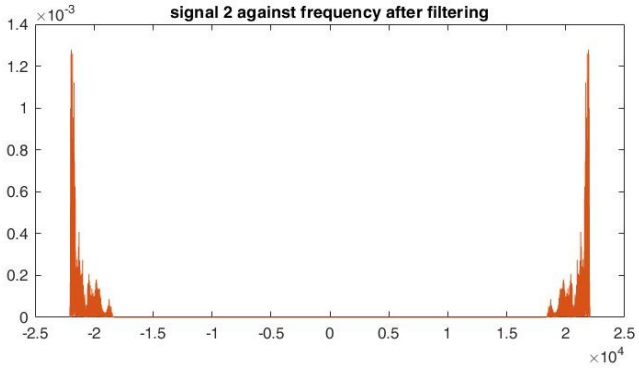


Figure 5: the two signals in before filtering







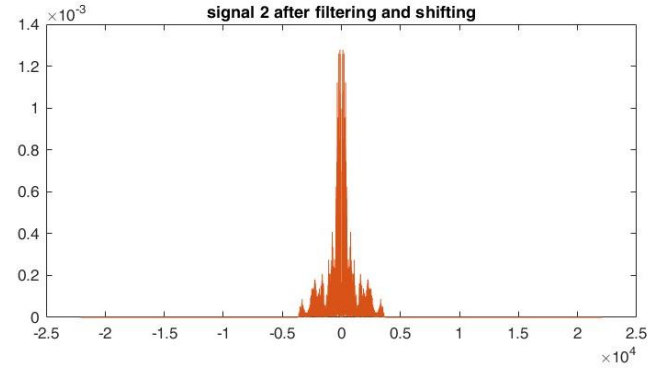


figure 6: the two signals in after filtering

Problem D:

Problem:

Perform amplitude modulation on both signals in a frequency-division multiplexing system, using suitable carrier frequencies [Lecture 10]. Justify your choice of carrier frequencies. Plot the magnitude spectrum of the transmitted signal.

Explanation:

- First we will define a time vector t = 0 : 1/fs : (N - 1)/fs.
- Then we used a carrier frequency for the first signal = 6500 Hz, and for the second signal we used carrier frequency = 16000 Hz
- Then we wrote the carrier equation : cos(2*pi*carrierFreq*t);
- Then we get the transpose of the carrier, to perform the multiplication operation correctly.
- Making a variable(modulated signal for each signal) which equal the dot product of the filtered signal and its carrier transpose.
- Get the transmitted signal by get the sum of the modulated signals.
- Then we will get the the fft of the 3 signals (modulated signal 1, modulated signal 2, transmitted signal).
- After we did that, we plot the result to make sure that all steps were correct and the result were as the following:

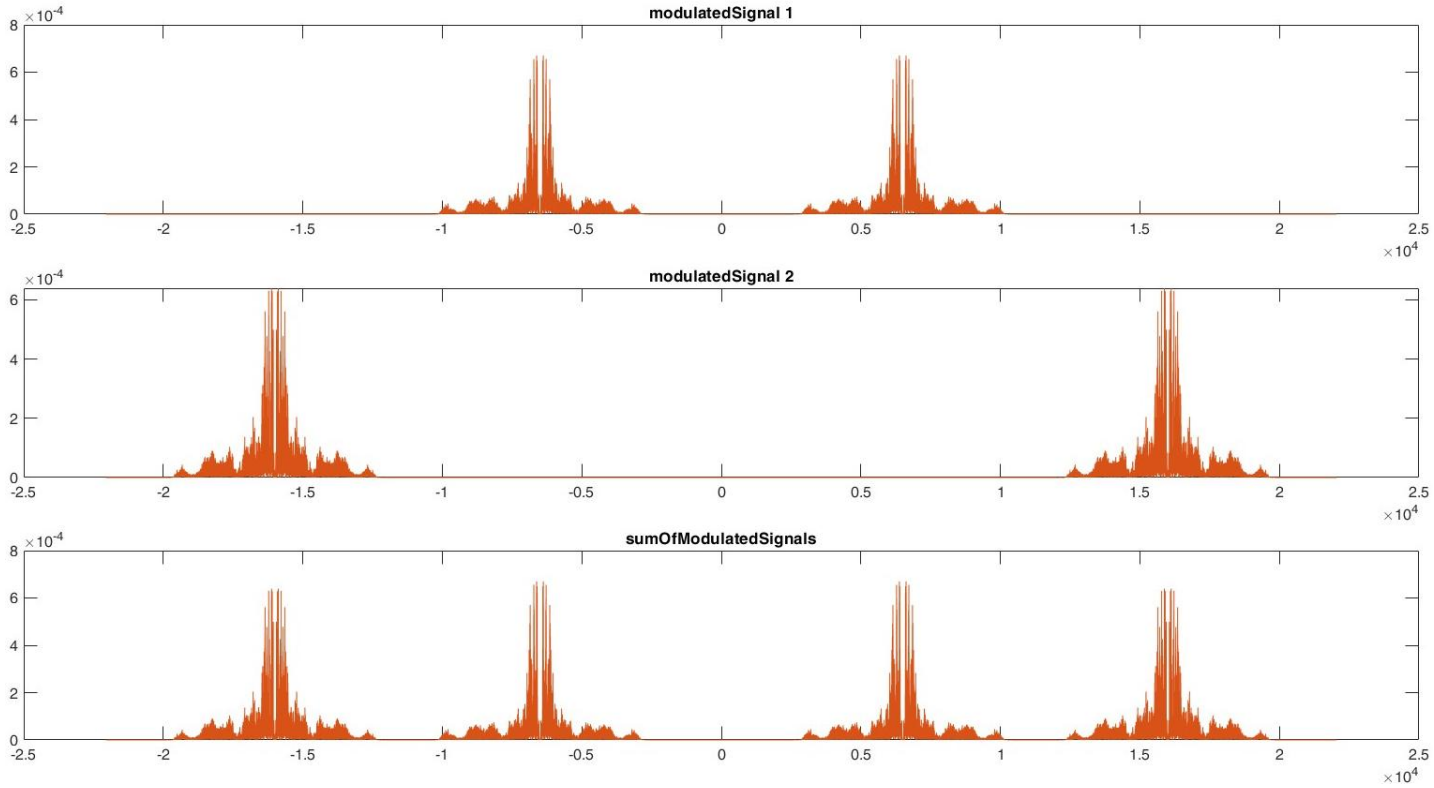


Figure 7: modulated signals and transmitted signal

Then why we used carrier frequencies with these values?

We know that the carrier frequency should be greater than the maximum frequency of the signal, then the maximum frequency of the first signal is : 3650 Hz, then we selected a carrier equals to 6500 Hz, to make sure that there is no interference between the shifted signals.

And for the second one, its carrier should be greater than the the double of the maximum frequency of the first signal (band width of the signal) plus the carrier frequency of the first one, then we selected carrier equals to 16000 Hz

Problem E:

Problem:

Design a receiver to obtain each of the two signals from the transmitted signal. Save the audio files as 'output1.wav' and 'output2.wav'. Plot the magnitude spectrum of each. Draw a block diagram of both transmitter and receiver. Explain the operation of the receiver with equations both in time domain and in frequency domain.

Explanation:

- First we will design a band pass filter, that passes the first signal and reject the second one, then we need to know $Fpass_{1,2}$ and $Fstop_{1,2}$, we can obtain them from the graph we plotted above, then after we did that, we designed the filter as the figure below:

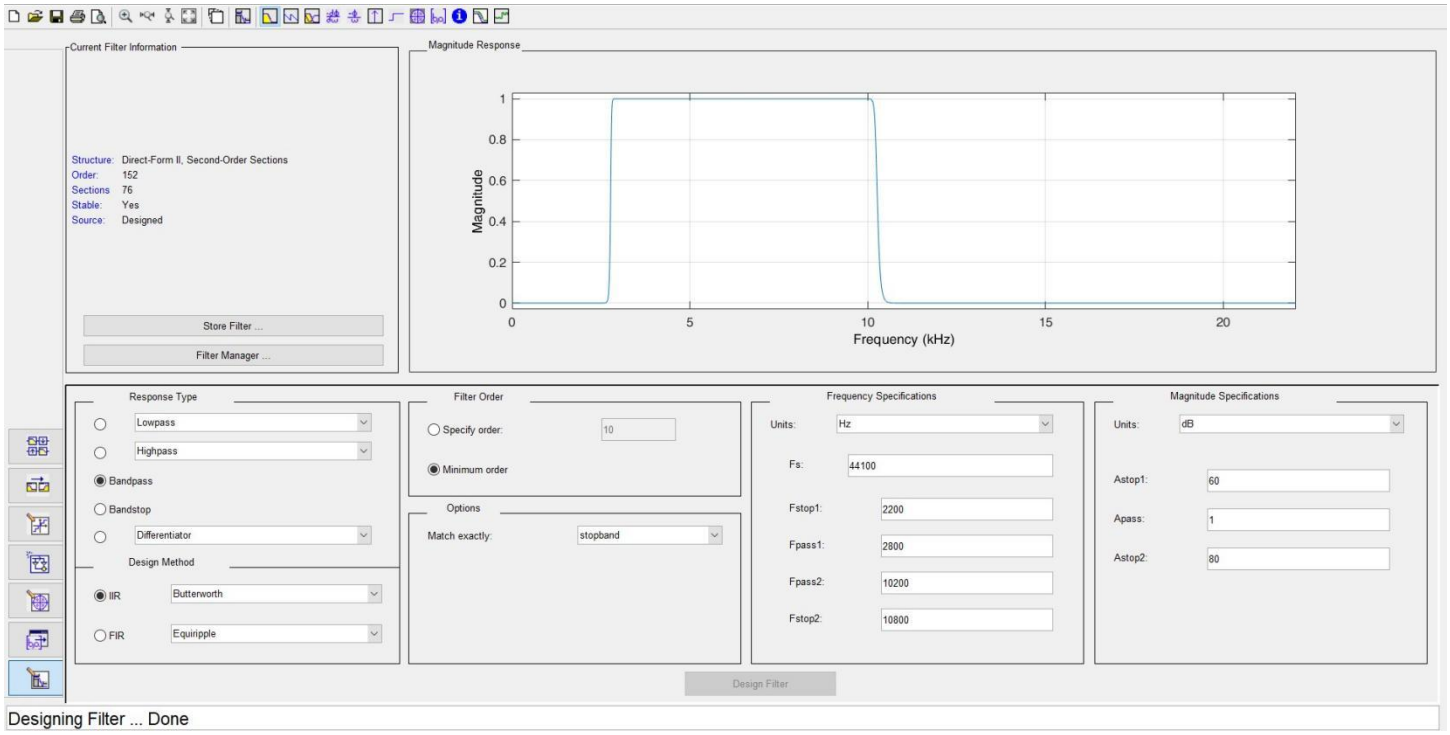


Figure 8: the band pass filter for the first signal

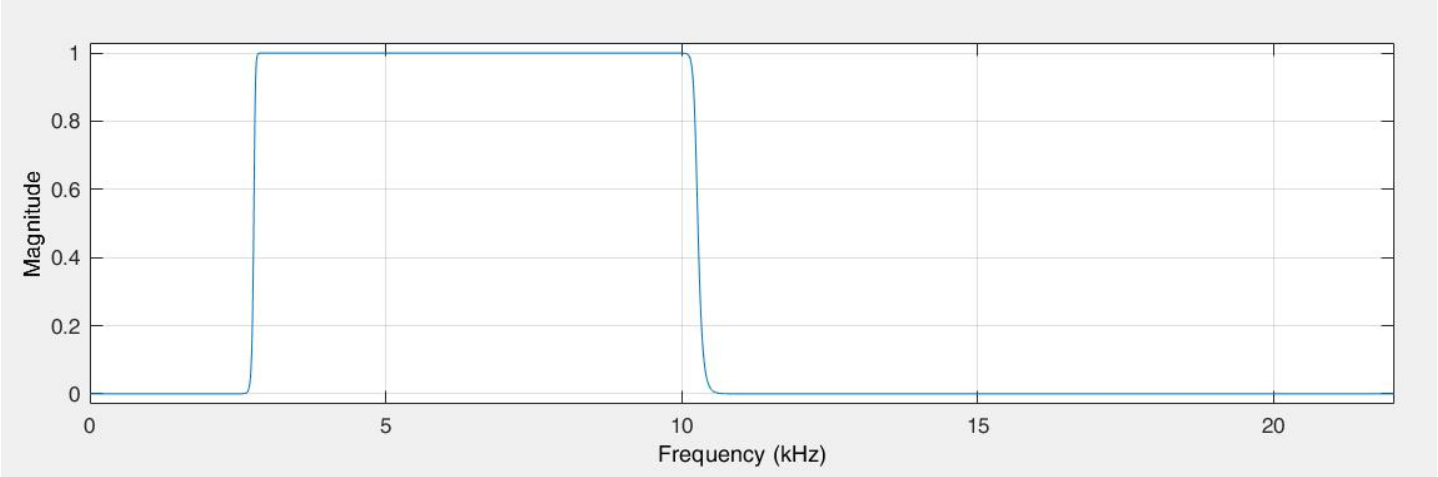


Figure 9: magnitude spectrum of the first band pass filter

- Then we did the same thing for second signal:

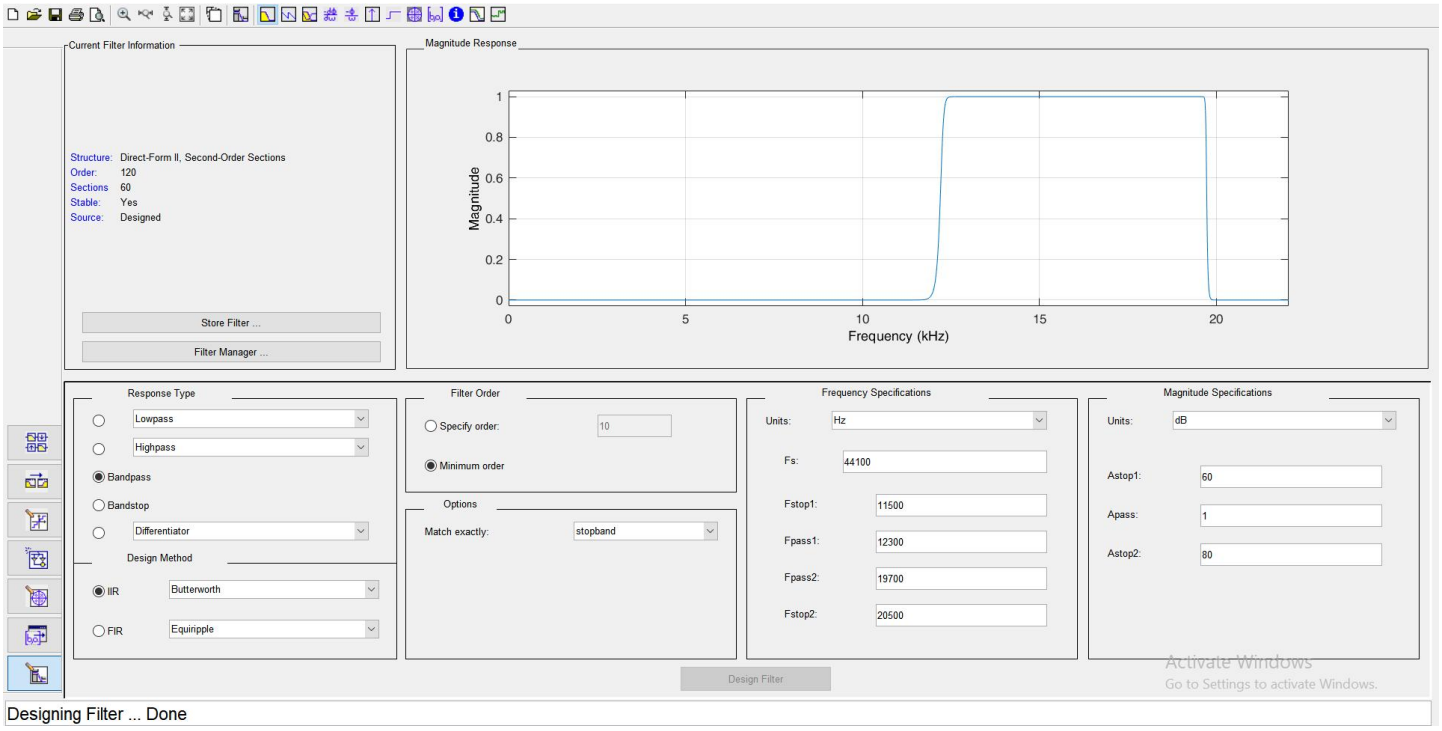


Figure 10: the band pass filter for the second signal

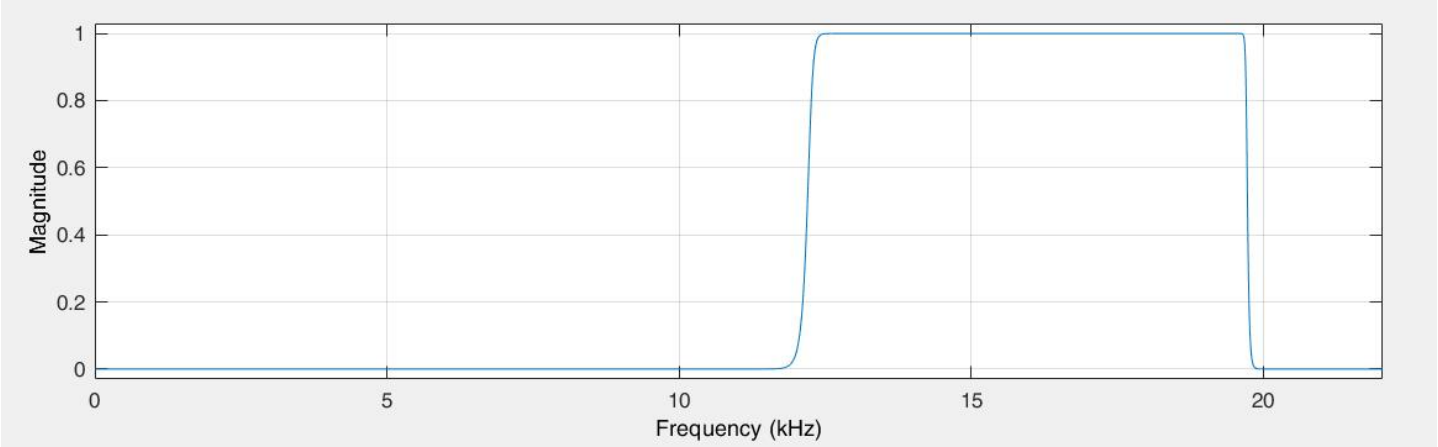


Figure 11:magnitude spectrum of the second band pass filter

- Then we will apply these filters on the transmitted signal, we got the first demultiplexed signal from the first filter, and the second demultiplexed signal from the second filter.
- The results are the following:

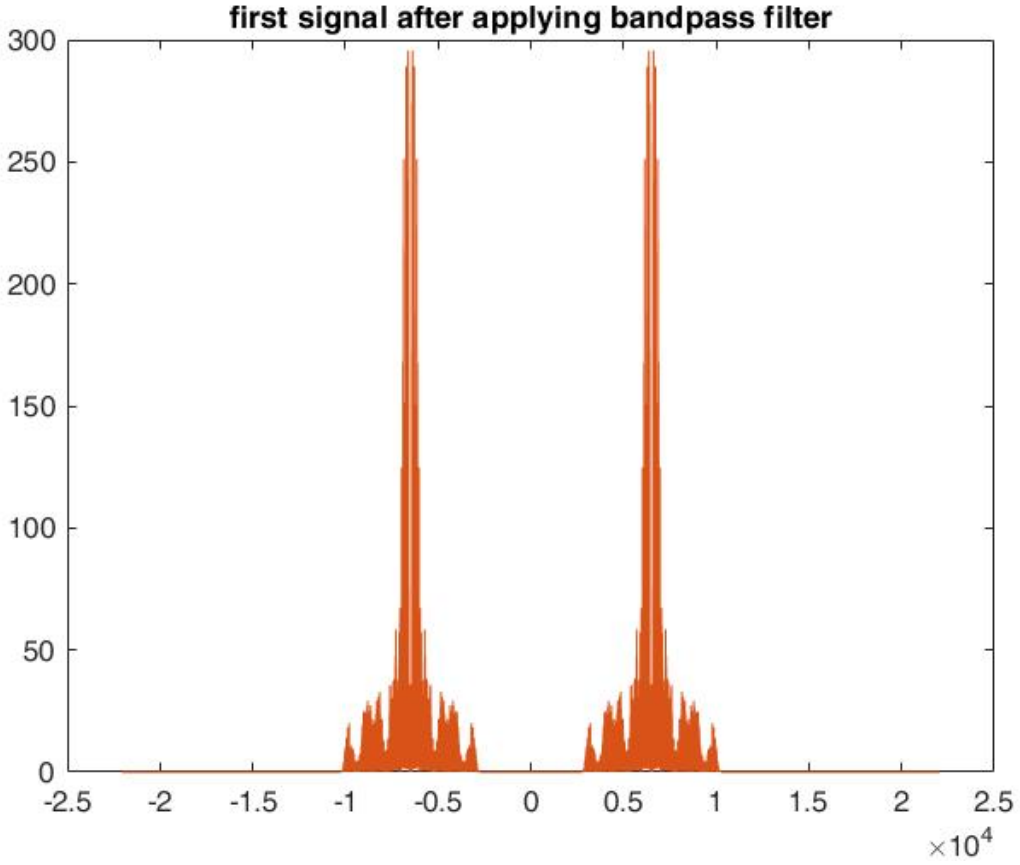


Figure 12: first demultiplexed signal

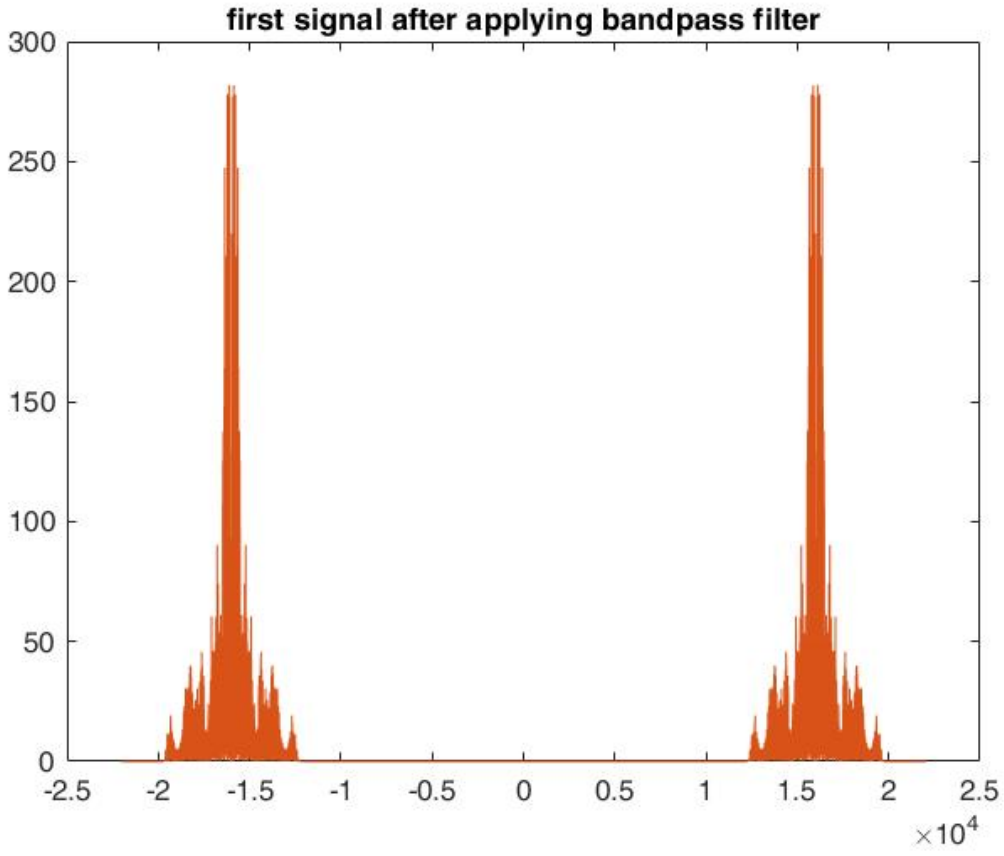


Figure 13: second demultiplexed signal

- After that we will multiply them by their carriers again:

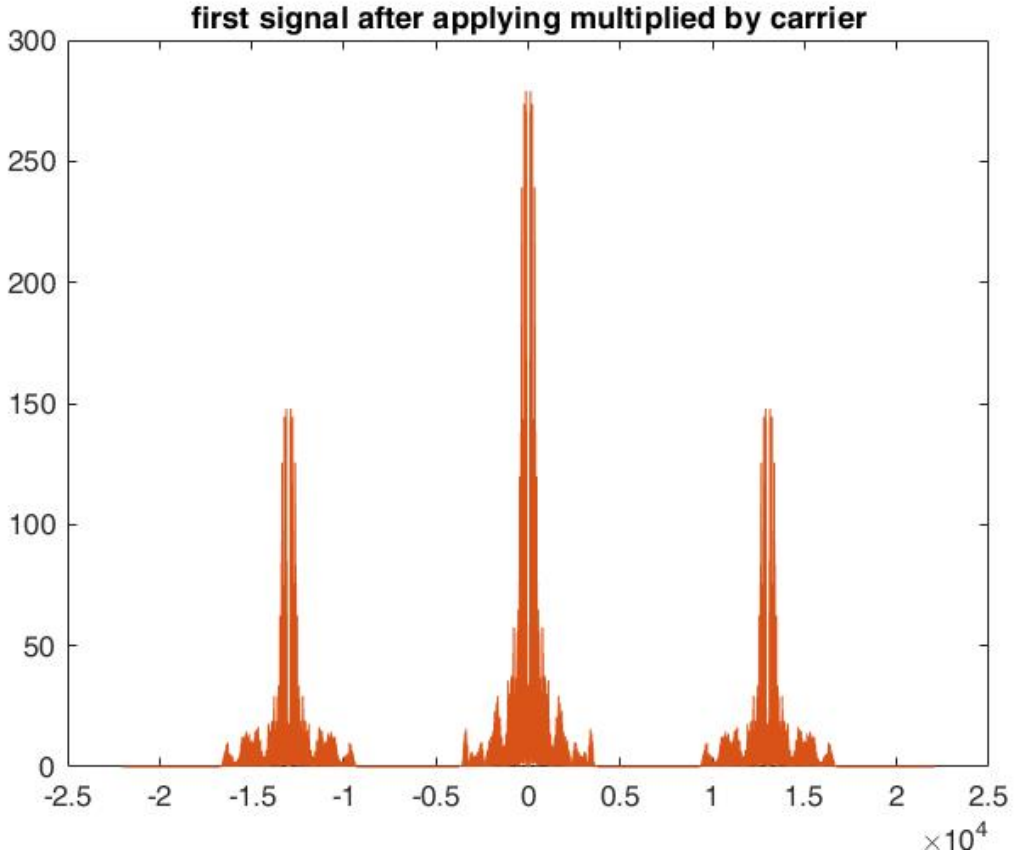


Figure 14:first demultiplexed signal multiplied by its carrier

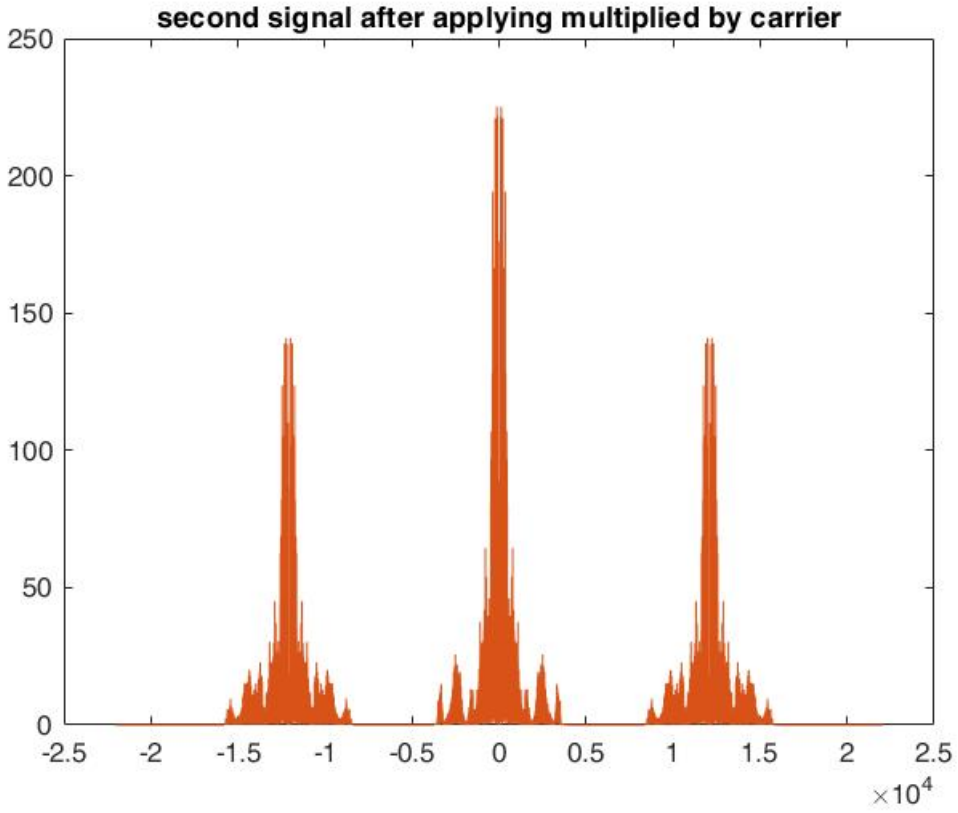


Figure 15: second demultiplexed signal multiplied by its carrier

- Demodulation step: then we designed a low pass filter to get the signal we need, and multiplied it by 2, as when we shifted it, its amplitude decreases by factor 2.
- The Fpass and Fstop of the low pass filter was determined from the second demultiplexed signal multiplied by its carrier and we found that the two signals have the same band width, then we designed a single filter for both.
- The low pass filter:

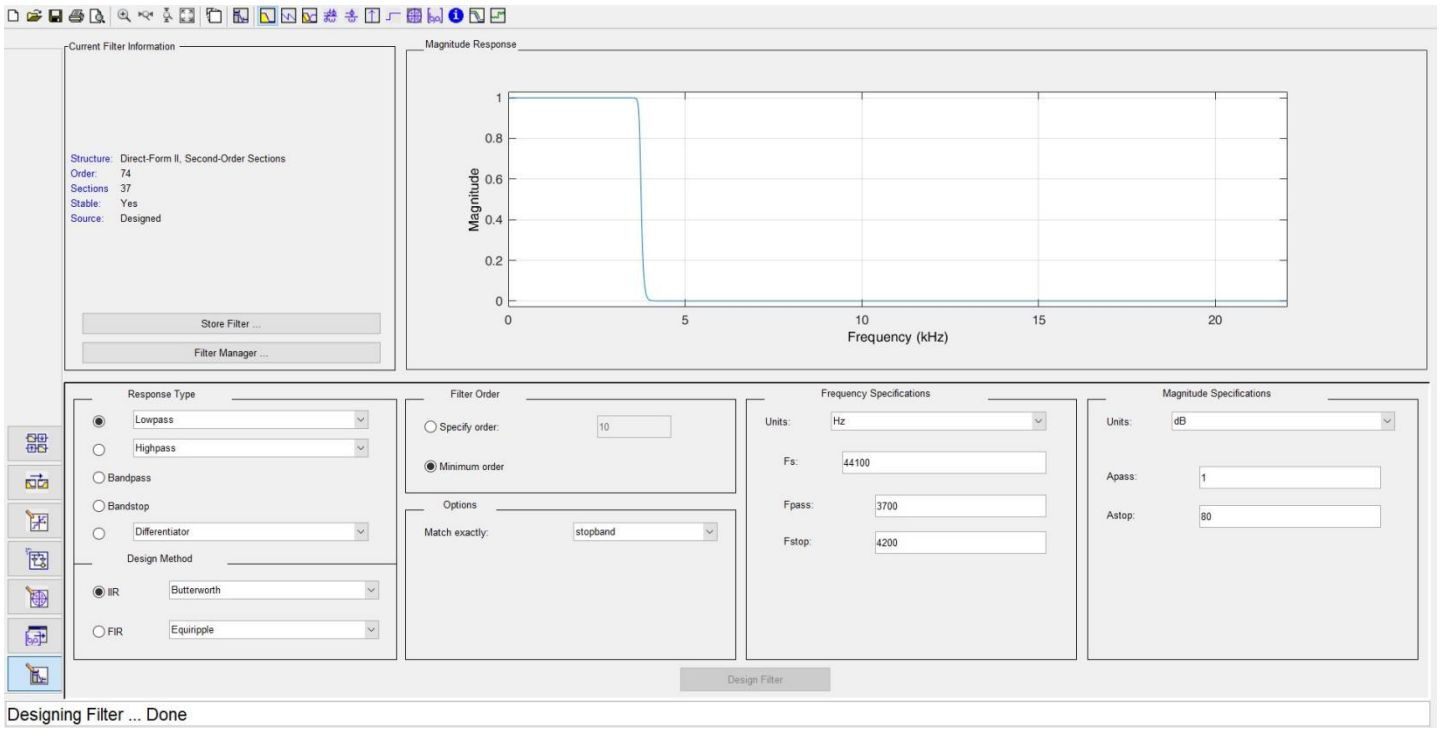


Figure 16: the low pass filter for demodulation

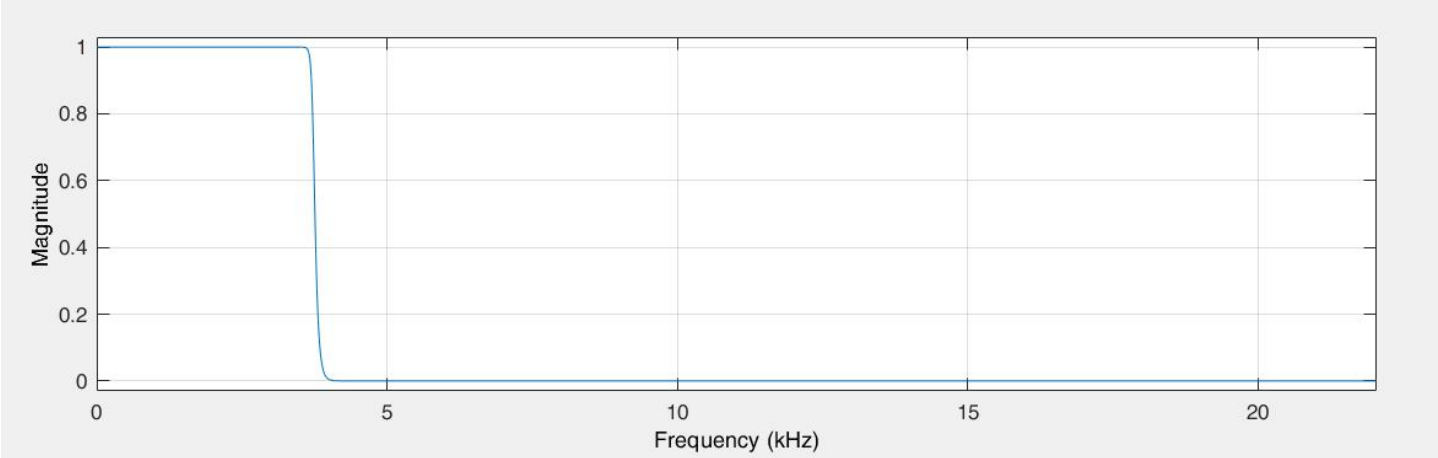


Figure 17: magnitude spectrum of the low pass filter for demodulation

- The demodulated signals from this filter are the restored signals, the following figures shows them:

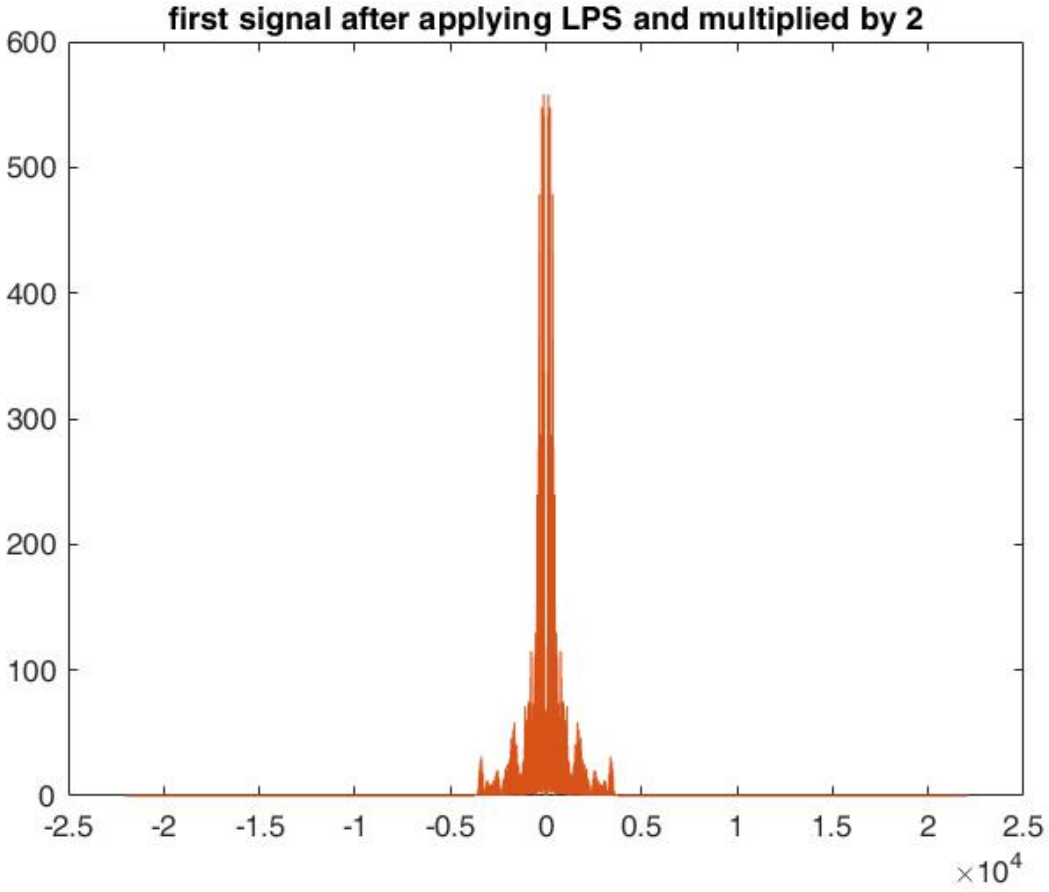


Figure 18: the first restored signal

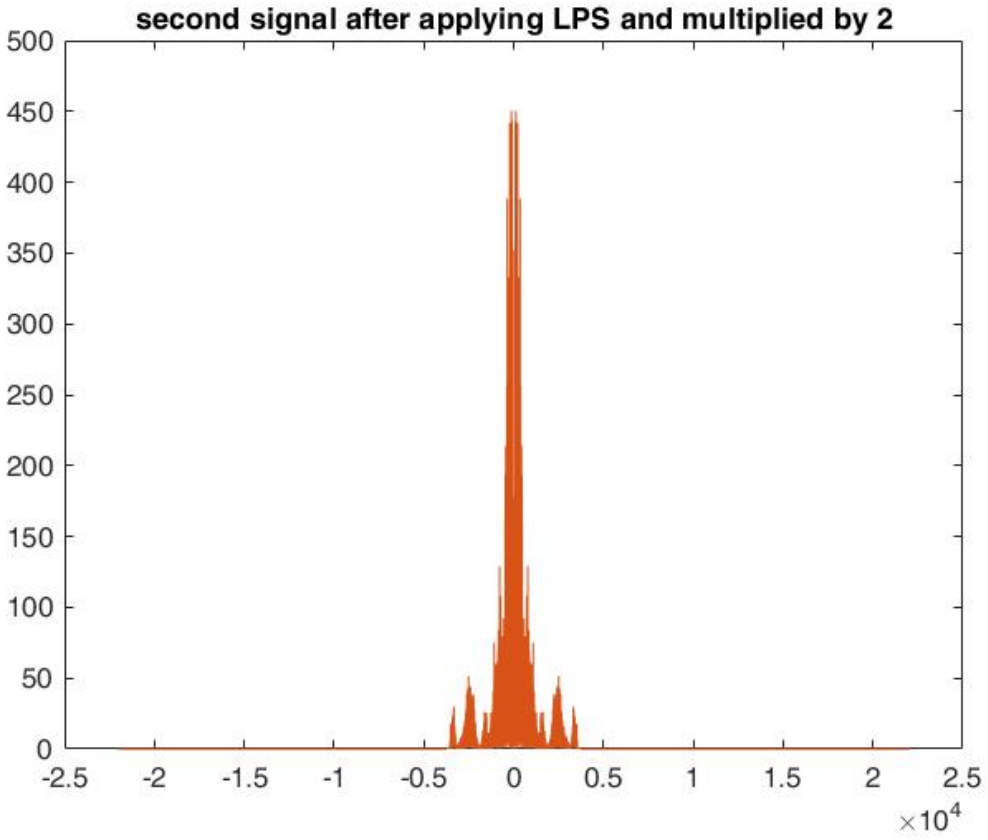


Figure 19: the second restored signal

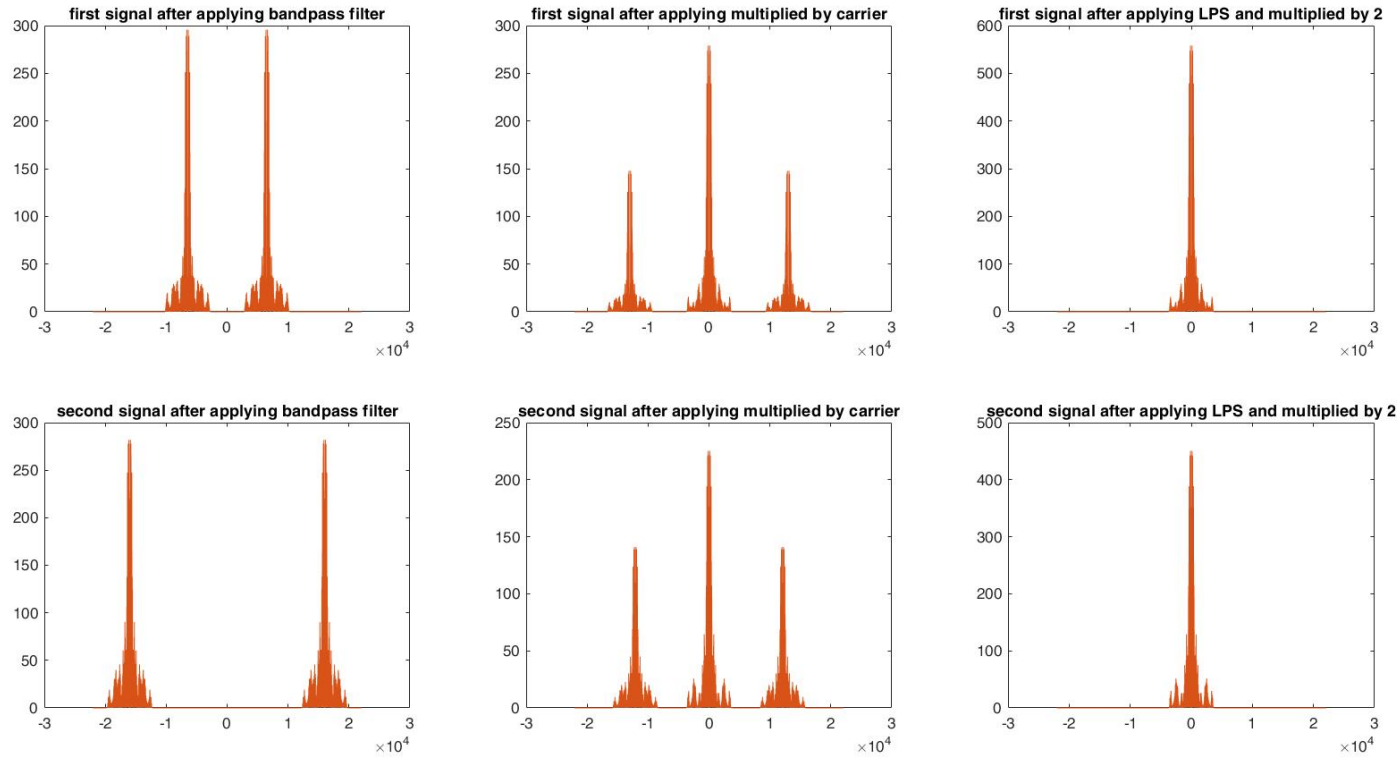


Figure 20: the whole process steps

The block diagram of transmitter and receiver :

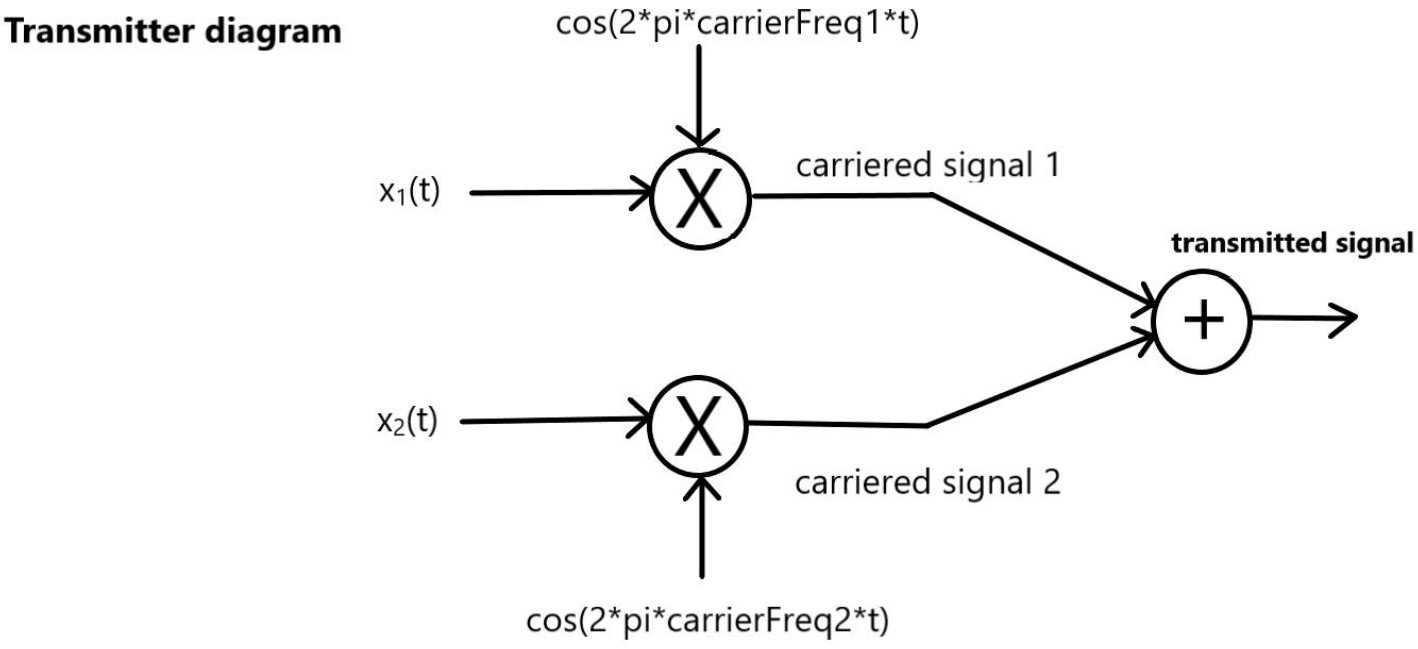


Figure 21: block diagram of the transmitter

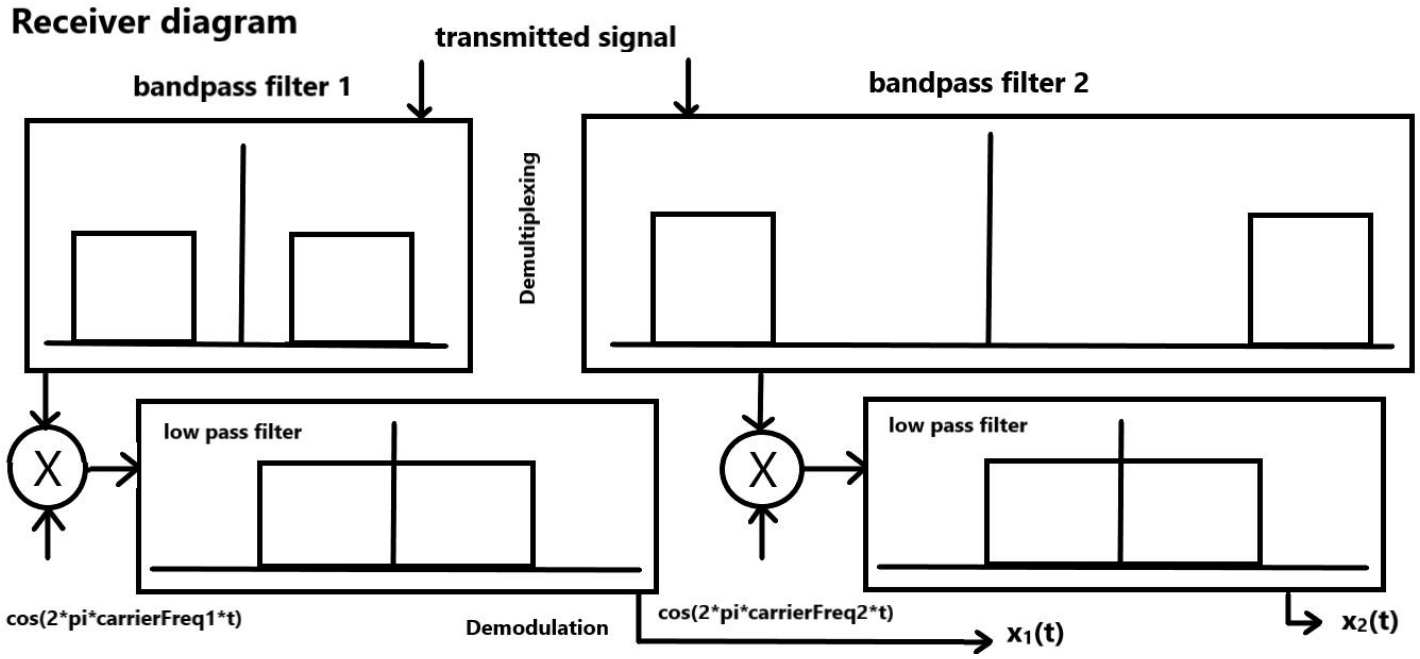


Figure 22:block diagram of the receiver

- Then we saved the restored audio (demodulated signals) using the function audiowrite() and named them as required.

Q: Explain the operation of the receiver with equations both in time domain and in frequency domain.

Ans:
In time domain:
Carrier : $p_1(t) = \cos(2\pi f_{c1} t)$, $p_2(t) = \cos(2\pi f_{c2} t)$, $\omega_{c1} = 2\pi f_{c1} t$, $\omega_{c2} = 2\pi f_{c2} t$
The transmitted signals : $s_1(t)$, $s_2(t)$
After applying the band pass filter on the signals we get The demultiplexed signals : $w_1(t)$, $w_2(t)$.

Then the restored signals : $x_1(t) = 2 * cos(2\pi f_{c1}) * w_1(t)$

$$x_2(t) = 2 * cos(2\pi f_{c2}) * w_2(t)$$

In frequency domain:

$$X_1(\omega) = \frac{1}{2\pi} * 2 * (W_1(\omega) \text{ conv } P_1(\omega))$$

$$X_2(\omega) = \frac{1}{2\pi} * 2 * (W_2(\omega) \text{ conv } P_2(\omega))$$

Or by shifting property:

$$X_1(\omega) = (W_1(\omega - \omega_{c1}) + W_1(\omega + \omega_{c1}))$$

$$X_2(\omega) = (W_2(\omega - \omega_{c2}) + W_2(\omega + \omega_{c2}))$$

Matlab code for Image part:

```
% READ THE IMAGE

image = imread('C:\Users\altyseer\Desktop\signals\peppers.png');
% ____________________________________________________________________________________

% gray preprocessing

grayImage = rgb2gray(image);
grayImage = im2double(grayImage);

% ____________________________________________________________________________________

% RGB preprocessing

redimg = image(:,:,1);
greenimg = image(:,:,2);
blueimg = image(:,:,3);
blackImg = zeros(size(blueimg,1),size(blueimg,2));
redcomp = cat(3,redimg,blackImg,blackImg);
greencomp = cat(3,blackImg,greenimg,blackImg);
bluecomp = cat(3,blackImg,blackImg,blueimg);

% ____________________________________________________________________________________

% Display THE ORIGINAL IMAGA & ITS 3 CHANNELS.

figure;
subplot(4,5,3)
imshow(image)
title("the original component")
subplot(4,5,8)
imshow(redcomp)
title("the red component")
subplot(4,5,13)
imshow(greencomp)
title("the green component")
subplot(4,5,18)
imshow(bluecomp)
title("the blue component")

redimg = im2double(redimg);
greenimg = im2double(greenimg);
blueimg = im2double(blueimg);
% ____________________________________________________________________________________

% edge detection KERNALS

vertical_kernel = [-1 0 1; -1 0 1; -1 0 1];
horizontal_kernrl = [-1 -1 -1;0 0 0;1 1 1]
% ____________________________________________________________________________________
```

```
% VERTICAL EDGE DETECTION

y_gray_edges = conv2(grayImage, vertical_kernel);
y_red_edges = conv2(redimg, vertical_kernel);
y_green_edges = conv2(greenimg, vertical_kernel);
y_blue_edges = conv2(blueimg, vertical_kernel);

% ____________________________________________________________________________________

% HORIZONTAL EDGE DETEXTCTION

x_gray_edges = conv2(grayImage, horizontal_kernrl);
x_red_edges = conv2(redimg, horizontal_kernrl);
x_green_edges = conv2(greenimg, horizontal_kernrl);
x_blue_edges = conv2(blueimg, horizontal_kernrl);
% ____________________________________________________________________________________

Y_edge_RGB= cat(3,y_red_edges,y_green_edges,y_blue_edges);
x_edge_RGB= cat(3,x_red_edges,x_green_edges,x_blue_edges);
gray_edges = sqrt(x_gray_edges.^2 + y_gray_edges.^2);
figure
imshow(gray_edges)
title("GRAY Edge detection")

% ____________________________________________________________________________________

RGB_edges = sqrt(x_edge_RGB.^2 + Y_edge_RGB.^2);
figure
imshow(RGB_edges)
title("RGB Edge detection")

% ____________________________________________________________________________________

% SHARPING

sharp_kernel = [-1 -1 0;-1 7 -1; -1 -1 0]
sharp_red = conv2(redimg, sharp_kernel);
sharp_green = conv2(greenimg, sharp_kernel);
sharp_blue = conv2(blueimg, sharp_kernel);
sharp_image= cat(3,sharp_red,sharp_green,sharp_blue);

figure
imshow(sharp_image)
title("Image sharpening")

% ____________________________________________________________________________________
% BLURING

blurr_kernel = [1 1 1;1 1 1; 1 1 1]/9
blurr_red = conv2(redimg, blurr_kernel);
blurr_green = conv2(greenimg, blurr_kernel);
blurr_blue = conv2(blueimg, blurr_kernel);
blurr_image= cat(3,blurr_red,blurr_green,blurr_blue);
figure
imshow(blurr_image)
title("Blurring")
```

```
% motion_blure

mov_blurr_kernel =zeros(15,15) ;
mov_blurr_kernel(8,:)= 1/15;
mov_blurr_red = conv2(redimg, mov_blurr_kernel);
mov_blurr_green = conv2(greenimg, mov_blurr_kernel);
mov_blurr_blue = conv2(blueimg, mov_blurr_kernel);
mov_blurr_image= cat(3,mov_blurr_red,mov_blurr_green,mov_blurr_blue);
figure
imshow(mov_blurr_image)
title("Motion Blurring")
% ____________________________________________________________________________________

%Restore the orignal image

size_out = size(mov_blurr_red);
f_kernel = zeros(size_out(1),size_out(2));
f_kernel(1:15,1:15) = mov_blurr_kernel;
f_kernel = fft2(f_kernel);
f_red = fft2(mov_blurr_red);
f_green = fft2(mov_blurr_green);
f_blue = fft2(mov_blurr_blue);
% ____________________________________________________________________________________


f_red_original= f_red./f_kernel;
f_green_original= f_green./f_kernel;
f_blue_original= f_blue ./f_kernel;


red_original = ifft2(f_red_original);
green_original = ifft2(f_green_original);
blue_original = ifft2(f_blue_original);

% ____________________________________________________________________________________
```

Matlab code for Audio part:

```
clear
clc
close all
load bandpass2.mat
load bandpass1.mat
load Hd.mat
load LPS1.mat

%-------------------------------------------------------------------------

% task A:


%%
% recording and saving the first audio file
audioFile1 = audiorecorder (44100, 16, 2);
disp('start recording input 1');
recordblocking(audioFile1, 10);
disp('end of recording input 1');
input1 = getaudiodata (audioFile1, "double");
audiowrite('input1.wav', input1, 44100);
sound(input1, 44100);
%%
% recording and saving the second audio file
audioFile2 = audiorecorder (44100, 16, 2);
disp('start recording input 2');
recordblocking(audioFile2, 10);
disp('end of recording input 12');
input1 = getaudiodata (audioFile2, "double");
audiowrite('input2.wav', input1, 44100);
sound(input1, 44100);
%%
bit_depth = 16;
fs = 44100;

[x1,fs] = audioread ('input1.wav');

[x2,fs] = audioread ('input2.wav');

% playing audio

sound(x1, fs);
sound(x2, fs);



% task B:

N = length(x1);
y1 = fft(x1, N); % get fast fourir transform of signal x1
% ----------------------
y2 = fft(x2, N); % get fast fourir transform of signal x1
% ----------------------
```

```
% for ploting against frequency

f1 = (0 : N - 1) * fs / N; % frequency used in normal case
f2 = (-N/2 : N/2 - 1) * fs / N; % frequency used in shifting

% filtering the signals with low pass filter Hd

filteredSignal1 = filter(Hd, x1);
filteredSignal2 = filter(Hd, x2);

% ------------------------------
% listening to both signals to determine the suitable Fpass and Fstop
sound(filteredSignal1, fs);
sound(filteredSignal2, fs);

% convering the filtered signals by fft

filteredSignal_fft1 = fft(filteredSignal1, N);
filteredSignal_fft2 = fft(filteredSignal2, N);

%--------------------------------------------------------------------------

% task C:

% ploting signals before (unshifted and shifted) before and after filtering
% divide the screen into eight sections
figure
subplot(4, 2, 1);
plot(f1,abs(y1)/ N);
title('signal 1 against frequency before filtering')

% shifting zero to the center of the spectrum
subplot(4, 2, 2);
plot(f2, abs(fftshift(y1)) / N);
title('signal 1 after shifting before filtering');

subplot(4, 2, 3);
plot(f1,abs(y2)/ N);
title('signal 2 against frequency before filtering');

% shifting zero to the center of the spectrum
subplot(4, 2, 4);
plot(f2, abs(fftshift(y2)) / N);
title('signal 2 after shifting before filtering');
% ----------------------
subplot(4, 2, 5);
plot(f1,abs(filteredSignal_fft1)/ N);
title('signal 1 against frequency after filtering');

% shifting zero to the center of the spectrum
subplot(4, 2, 6);
plot(f2, abs(fftshift(filteredSignal_fft1)) / N);
title('signal 1 after filtering and shifting');

subplot(4, 2, 7);
plot(f2,abs(filteredSignal_fft2)/ N);
```

```
title('signal 2 against frequency after filtering');

% shifting zero to the center of the spectrum
subplot(4, 2, 8);
plot(f2, abs(fftshift(filteredSignal_fft2)) / N);
title('signal 2 after filtering and shifting');

%--------------------------------------------------------------------------

% task D:

% time vecor
t = 0 : 1/fs : (N - 1)/fs;

carrierFreq1 = 6500; %carrier frequency for the first signal
carrierFreq2 = 16000; %carrier frequency for the first signal


carrier1 = cos(2*pi*(carrierFreq1)*t);
carrier2 = cos(2*pi*(carrierFreq2)*t);

% make transpose for both carriers, to performe multiplication correctly

carrier1 = carrier1.';
carrier2 = carrier2.';

modulatedSignal1 = filteredSignal1.*carrier1; % .* dot product
modulatedSignal2 = filteredSignal2.*carrier2;
sumOfModulatedSignals = modulatedSignal1 + modulatedSignal2; % transmitted signal
fftmodulatedSignal1 = fft(modulatedSignal1);
fftmodulatedSignal2 = fft(modulatedSignal2);
fftsumOfModulatedSignals = fft (sumOfModulatedSignals);

figure;
subplot(3, 1, 1);
plot(f2, abs(fftshift(fftmodulatedSignal1)) / N);
title('modulatedSignal 1');

subplot(3, 1, 2);
plot(f2, abs(fftshift(fftmodulatedSignal2)) / N);
title('modulatedSignal 2');

subplot(3, 1, 3);
plot(f2, abs(fftshift(fftsumOfModulatedSignals)) / N);
title('sumOfModulatedSignals');

%--------------------------------------------------------------------------

% task e:
% for the first signal

demultiplixedSignal1 = filter(bandpass1, sumOfModulatedSignals); % applying bandpass
filter
carrieredDemultiplixedsignal1 = demultiplixedSignal1.*carrier1;
% CDS --> Carriered Demultiplixed Signal
filteredCDS1 = filter(LPS1, carrieredDemultiplixedsignal1).*2; % applying low pass filter
```

```
fftDMS1 = fft(demultiplixedSignal1); % DMS DeMultiplixed Signal
fftCDMS1 = fft(carrieredDemultiplixedsignal1); % CDMS --> Carriered DeMultiplixed Signal
fftfCDS1 = fft(filteredCDS1);

figure;
subplot(2, 3, 1)
plot(f2, abs(fftshift(fftDMS1)));
title('first signal after applying bandpass filter');
subplot(2, 3, 2)
plot(f2, abs(fftshift(fftCDMS1)));
title('first signal after applying multiplied by carrier');
subplot(2, 3, 3)
plot(f2, abs(fftshift(fftfCDS1)));
title('first signal after applying LPS and multiplied by 2');

% for the second signal

demultiplixedSignal2 = filter(bandpass2, sumOfModulatedSignals); % applying bandpass
filter
carrieredDemultiplixedsignal2 = demultiplixedSignal2.*carrier2;
% CDS --> Carriered Demultiplixed Signal
filteredCDS2 = filter(LPS1, carrieredDemultiplixedsignal2).*2; % applying low pass filter

fftDMS2 = fft(demultiplixedSignal2);
fftCDMS2 = fft(carrieredDemultiplixedsignal2);
fftfCDS2 = fft(filteredCDS2);


subplot(2, 3, 4)
plot(f2, abs(fftshift(fftDMS2)));
title('second signal after applying bandpass filter');
subplot(2, 3, 5)
plot(f2, abs(fftshift(fftCDMS2)));
title('second signal after applying multiplied by carrier');
subplot(2, 3, 6)
plot(f2, abs(fftshift(fftfCDS2)));
title('second signal after applying LPS and multiplied by 2');

%export demodulated audio files into PC
audiowrite('output1.wav' , filteredCDS1 , fs);
audiowrite('output2.wav' , filteredCDS2 , fs);
```

References:

1. https://youtu.be/KuXjwB4LzSA?si=_KU-nrCaqM_GToDT
2. https://www.tutorialspoint.com/edge-detection-using-prewitt-scharr-and-sobel-operator#:~:text=Technically%2C%20the%20Prewitt%20operator%20utilizes,of%20objects%20within%20the%20image
3. https://www.geeksforgeeks.org/image-edge-detection-operators-in-digital-image-processing/
4. https://medium.com/@itberrios6/how-to-apply-motion-blur-to-images-75b745e3ef17
5. https://www.geeksforgeeks.org/image-sharpening-using-laplacian-filter-and-high-boost-filtering-in-matlab/
6. https://en.m.wikipedia.org/wiki/Box_blur
7. https://www.sciencedirect.com/topics/engineering/laplacian-filter#:~:text=2D%20Gaussian%20function.-,The%20Laplacian%20filter%20detects%20sudden%20intensity%20transitions%20in%20the%20image,that%20determines%20the%20edge%20pixels
8. https://www.mathworks.com/help/matlab/ref/im2double.html
9. https://www.mathworks.com/help/matlab/ref/double.cat.html
10. https://www.mathworks.com/help/matlab/ref/transpose.html
11. https://www.mathworks.com/help/matlab/ref/subplot.html
12. https://www.mathworks.com/matlabcentral/answers/464783-finding-the-dot-product-of-two-vectors