Cairo University
Faculty of Engineering
Dept. of Electronics and Electrical Communications
Second Year
Analysis of Continuous-Time Signals
ELC 2030

Analysis of Continuous-Time Signals
Image filtering and restoration & Communication system simulation project

Student Name	Section	B.N.
يوسف اشرف محمد ابوطالب	4	9220972
يوسف خالد عمر محمود	4	9220984

Presented by

Instructors: Dr. Michael Melek Abdel-Sayed

Image filtering and restoration

Task a:

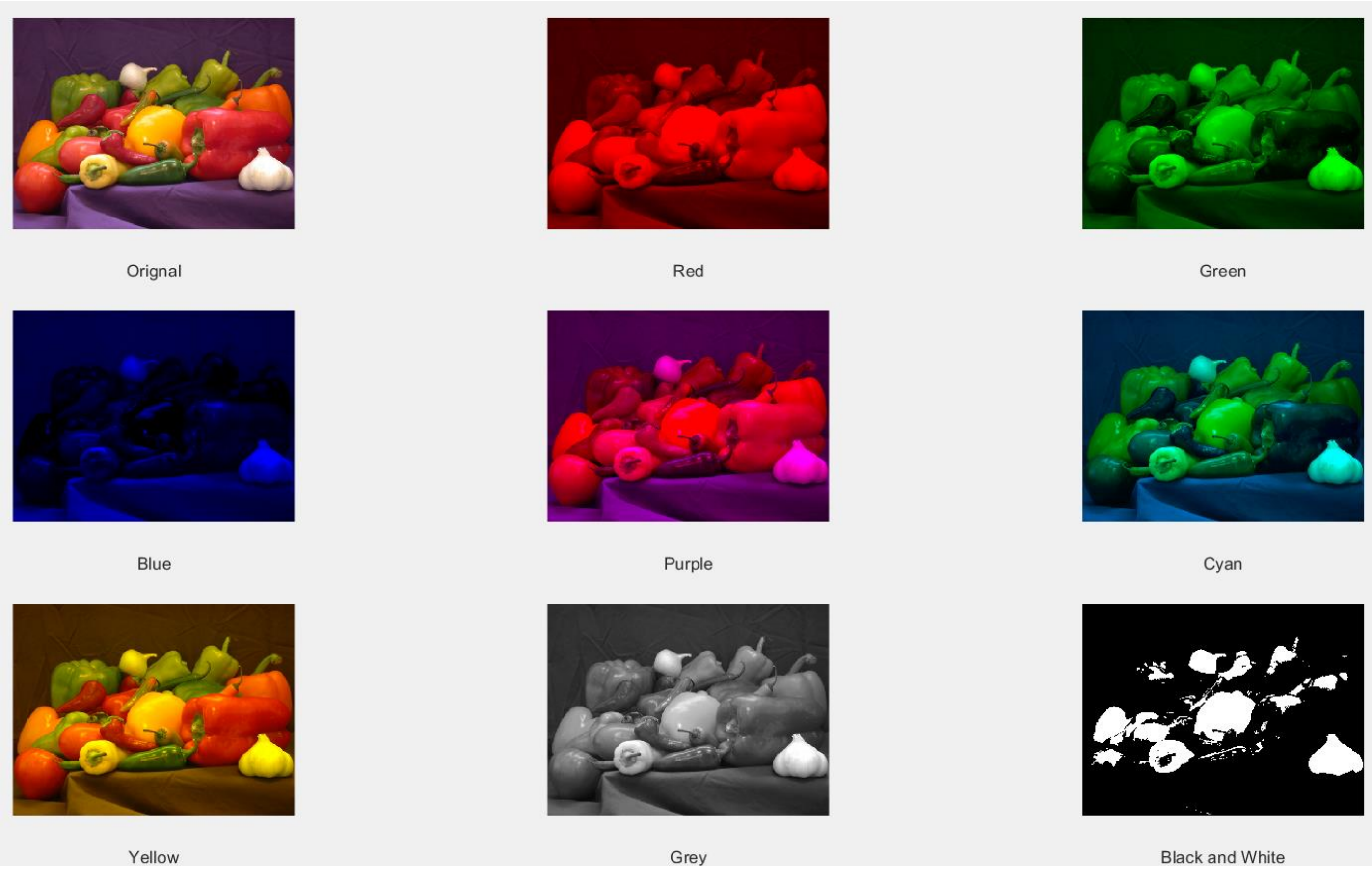
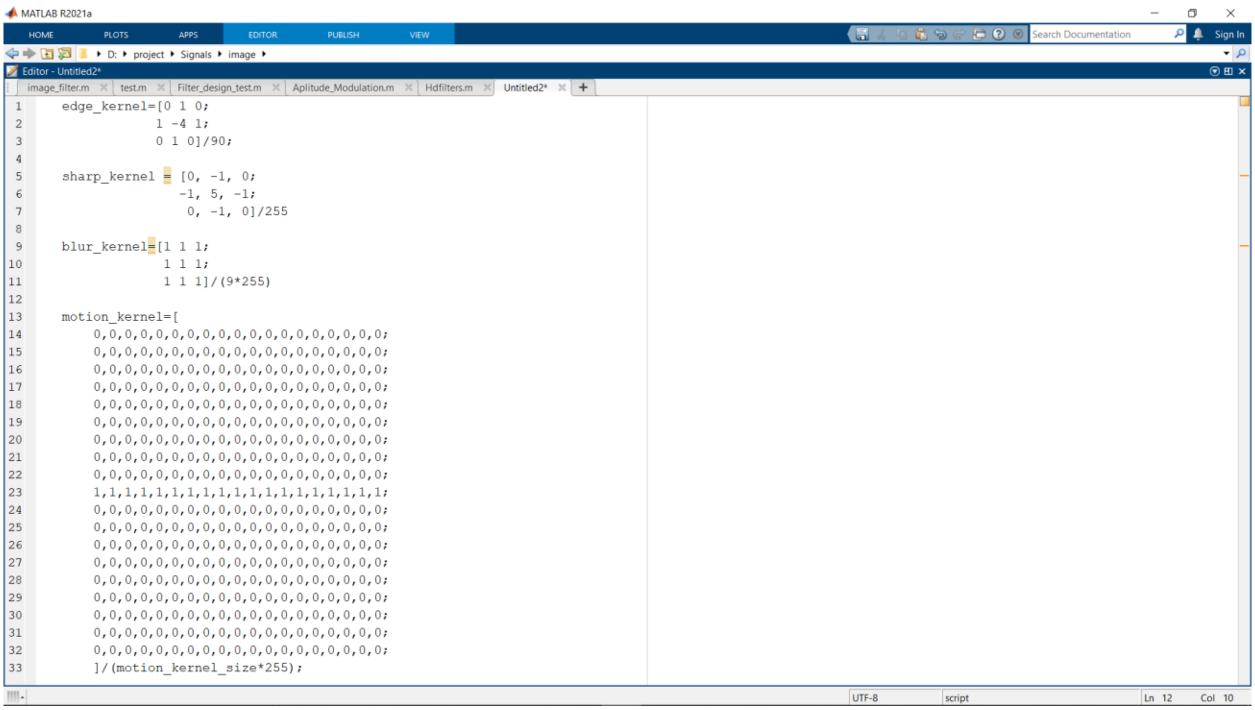


Figure 1 Colors of the picture

I read the picture “peppers” in section A of the image code, then extracted the red,green and blue part of it and Showing the 3 Comboniation (Purple, Yellow and Cyan) of them and getting gray picture to use it in edge detection and the last picture is black and white as shown in *Figure 1*.

Task b



```
edge_kernel=[0 1 0;
             1 -4 1;
             0 1 0]/90;

sharp_kernel = [0, -1, 0;
               -1, 5, -1;
                0, -1, 0]/255

blur_kernel=[1 1 1;
             1 1 1;
             1 1 1]/(9*255)

motion_kernel=[
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    ]/(motion_kernel_size*255);
```

Figure 2 Kernels

As shown in *figure 2*, We used for each process a certain kernal and we will discuss each process with its kernel.[1]

Edge:

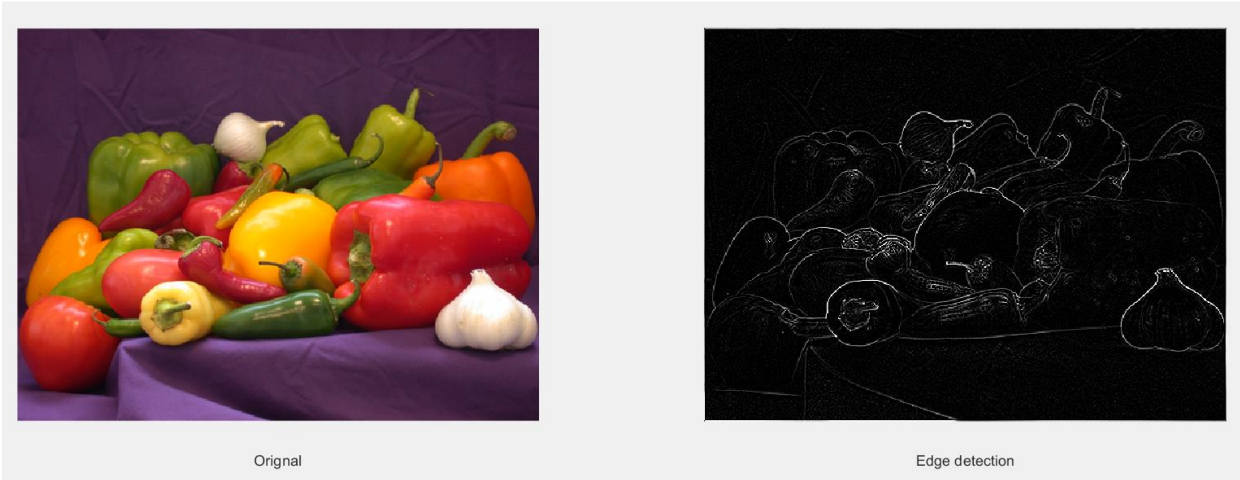


Figure 3 Edge picture

$Edge = \nabla\hat{}2\, pic$ *Equation 1*

As shown in *Equation 1* and *figure3*, the edge is the Laplacian of the picture so we made the kernel, shown in *figure 2*, that represent the Laplacian operator numerically and used it in section B of the image code.[2]

Sharp:

Figure 4 Sharp Picture

$Sharp = pic + \nabla\hat{}2\, pic$ *Equation 2*

As shown in *Equation 2* and *figure4*, the sharp is the sum of the Laplacian of the picture and the picture itself, so we made the kernel, shown in *figure 2*, that represent the Laplacian operator added with the orignal input numerically and used it in section C of the image code.[2]

Blur:

Figure 5 blur picture

$$Blur = avg(picture)$$ *Equation 3*

As shown in *Equation 3* and *figure5*, the blur is the average of the picture, so we made the kernel, shown in *figure 2*, that represent the average operator of the orignal input and used it in section D of the image code.[3]

Motion blur:

Figure 6 motion blur picture

$$Motion\ blur = shift\ horizontal(picture)$$ *Equation 4*

As shown in *Equation 4* and *figure6*, the motion blur is the average of the picture, so we made the kernel, shown in *figure 2*, that represent the average operator of the orignal input and used it in section E of the image code.[4]

Task c

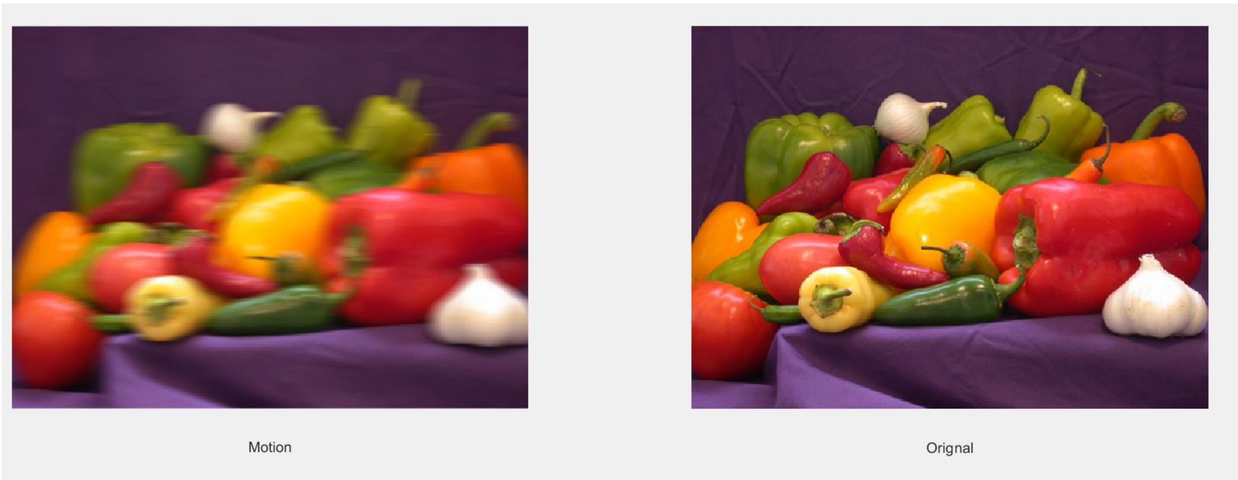


Figure 7 Restored picture

We know that:

$$Motion\ picture(t) = Picture(t) * motion\ kernel(t) \quad \textit{Equation 5}$$

But in frequency domain *Equation 5 will be:*

$$Motion\ picture(w) = Picture(w)\ \ X\ motion\ kernel(w) \quad \textit{Equation 6}$$

So to restore the orignal picture we will divide according to *Equation 6*, and to calculate the restored picture we will use:

$$picture(w) = motion\ picture(w)\ \ /\ motion\ kernel(w) \quad \textit{Equation 7}$$

So we first padded the motion kernel in section F of the image code then we fourier transformed both Motion picture and padded kernel to restore the picture according to *Equation 7.*

Then we inversed fourier transform the restored picture in frequency domain to become time domain.

When we display both motion and restored image, we crop them due to motion kernel effect.

Communication system simulation:

Explanation of my work :

First, we use matlab to record the voice using the record code. then we design a low bass filters with different cut-off frequency. Then in the filter code we chose the voice at cut-off frequency (20000 Hz &4000 Hz) to know what happened at this cut-off frequency, then we plot the input and the output record in different domain (time and frequency) and this is the result of two signal (records)[5].

Why we chose this sampling frequency and bit depth?

- We chose sampling frequency (44100) just to improve the qulity of voice, we know we can use less than this number[6].

- We chose the bit depth (16-bit) because that is the ideal number of bits use to store the voice.

First signal:

the result at (20000 Hz):

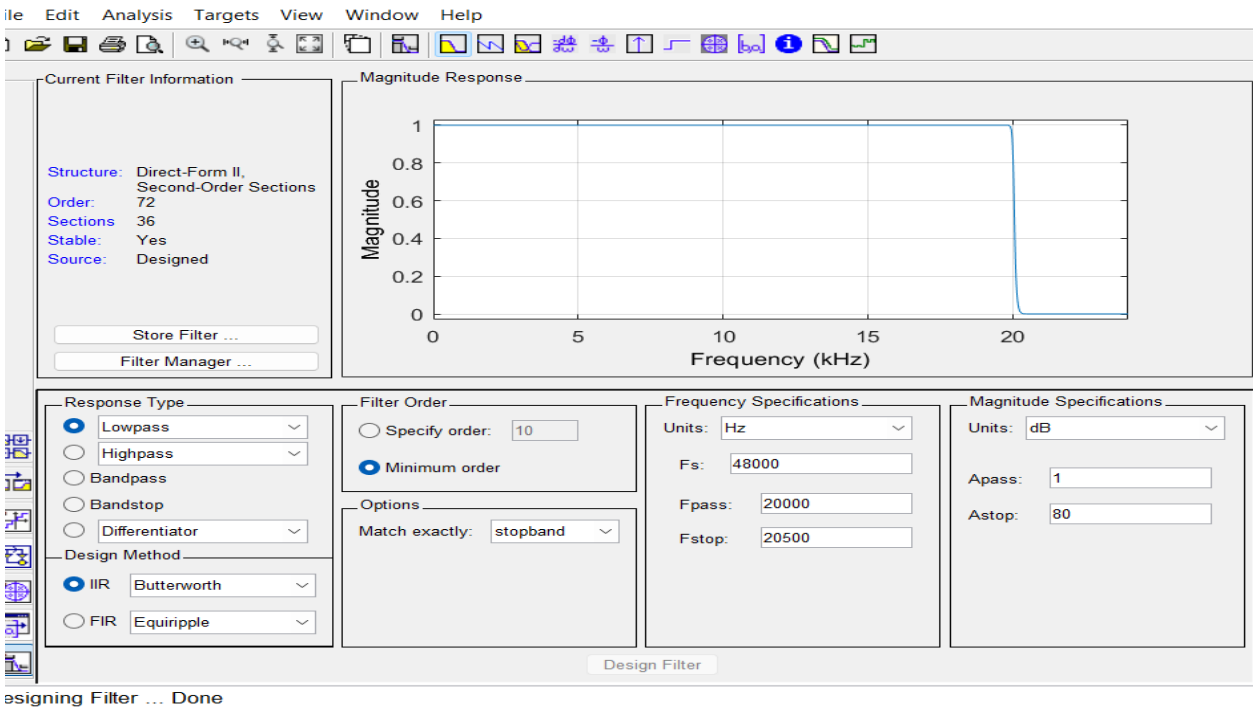


Figure 8.the frequency response of the filter

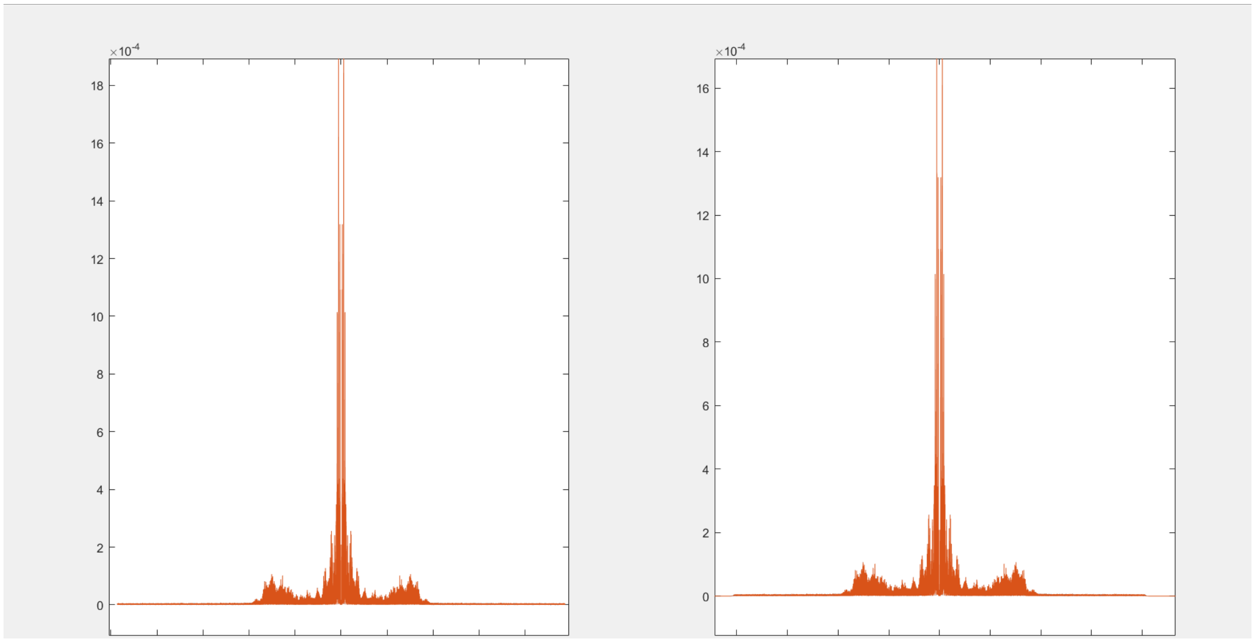


Figure 10.the magnitude spectrum of the first signal @20000 Hz.(input and output) in frequency response

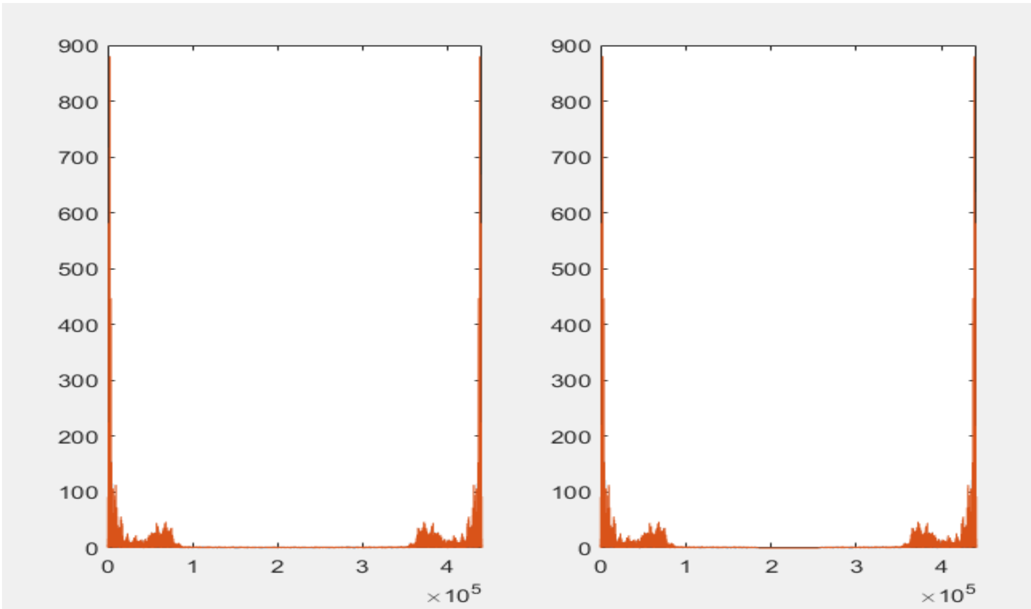


Figure 9.the magnitude spectrum of the first signal @2000 Hz.(input and output) (k)

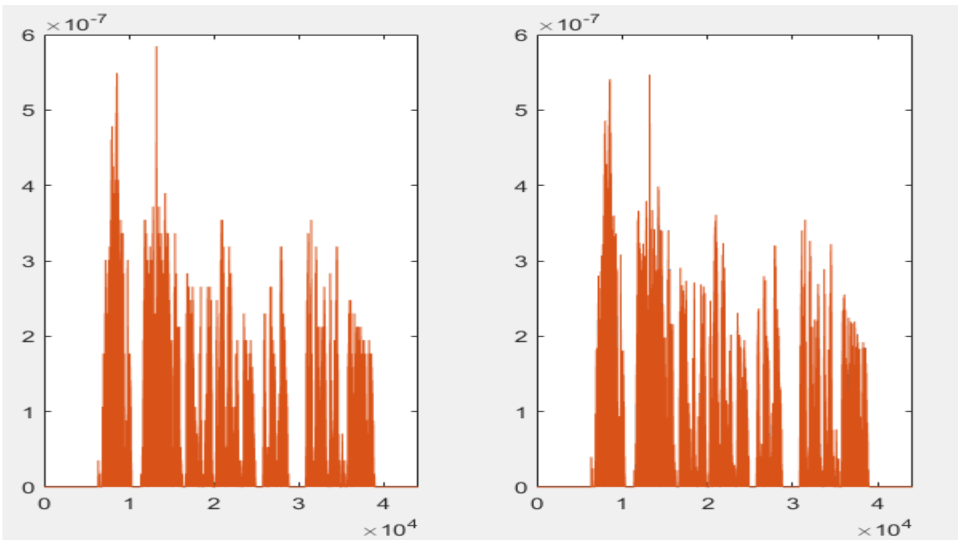


Figure 11 the magnitude spectrum of the first sgnal @20000 Hz.(input and output) (Hz)

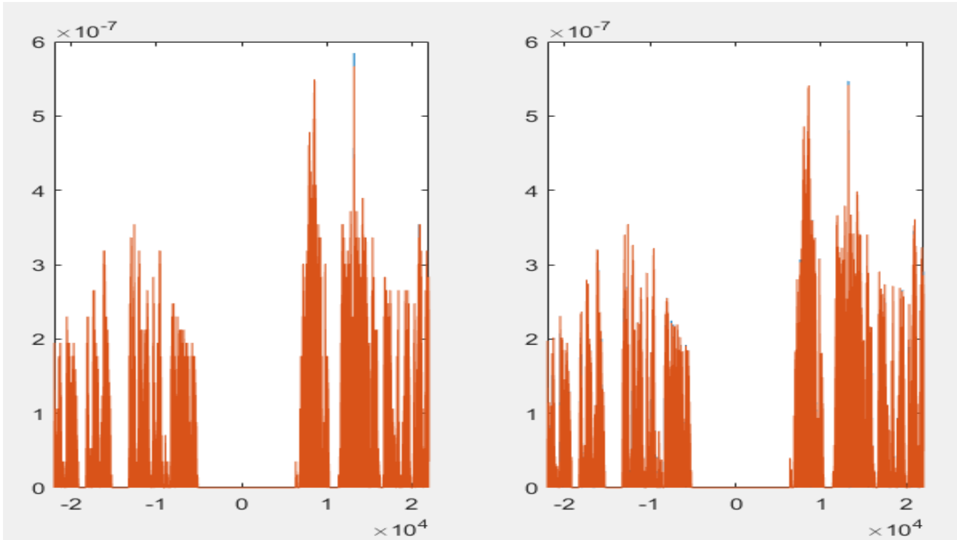


Figure 12.the magnitude spectrum of the first signal @20000 Hz.(input and output) (k) using ffishift'

The results at (4000 Hz):

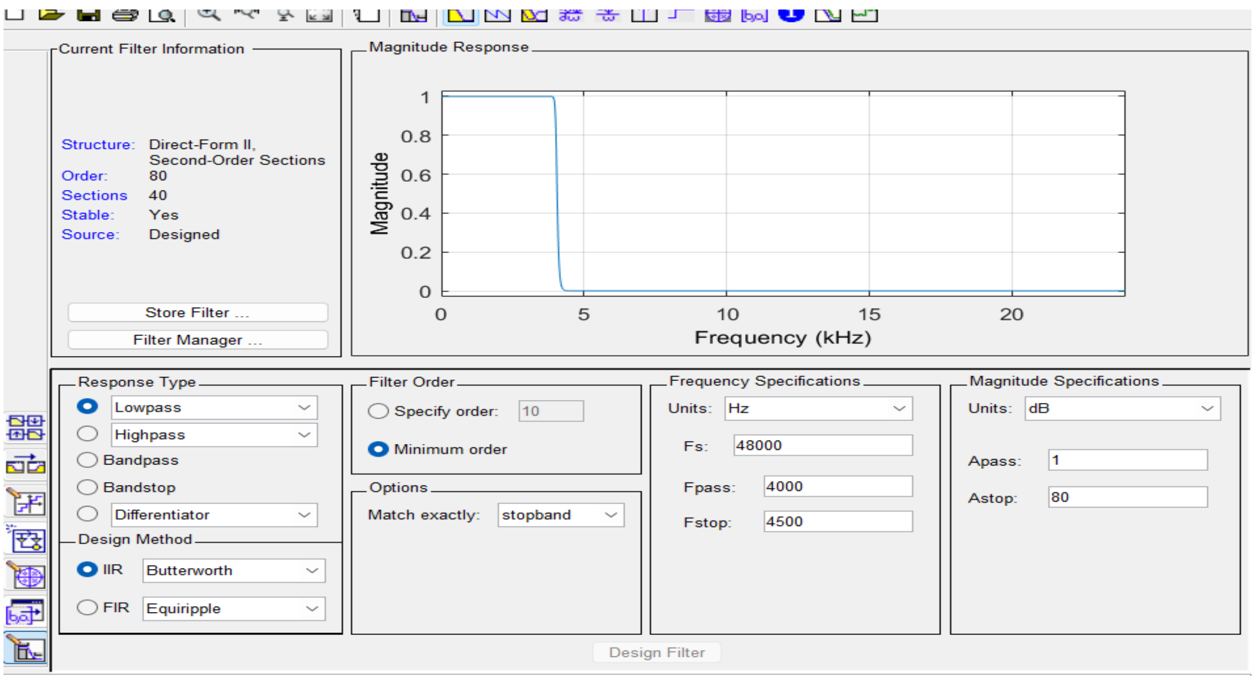


Figure 13.the frequency response of the filter(4000 hz)

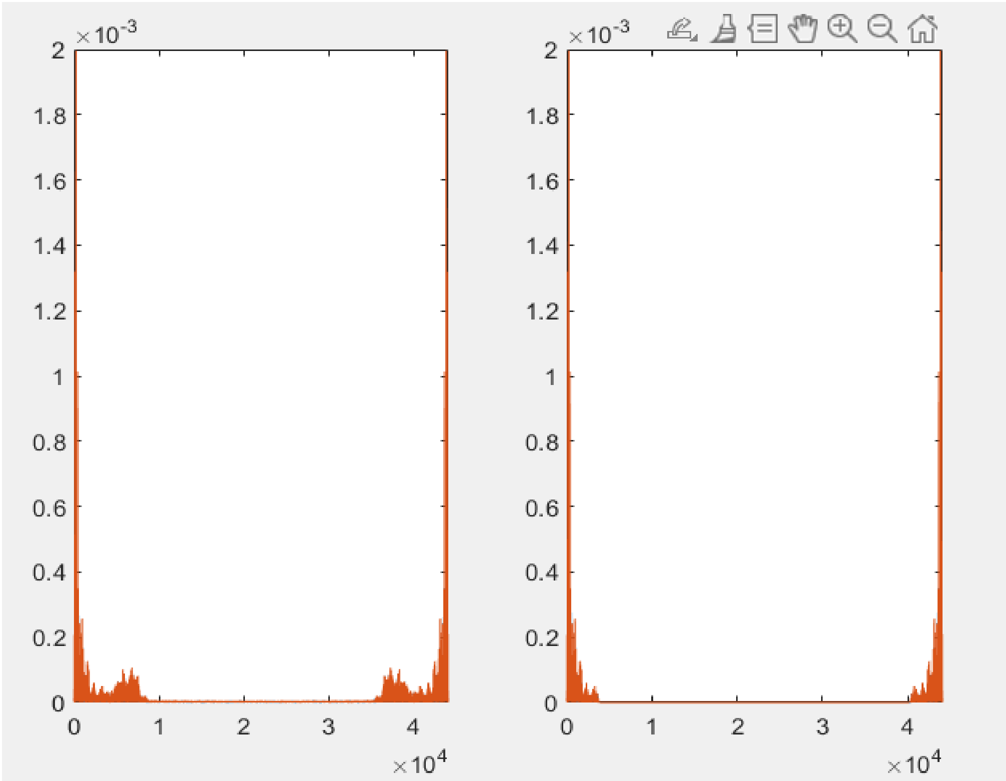


Figure 14..the magnitude spectrum of the first signal @4000 Hz.(input and output) in frequency response

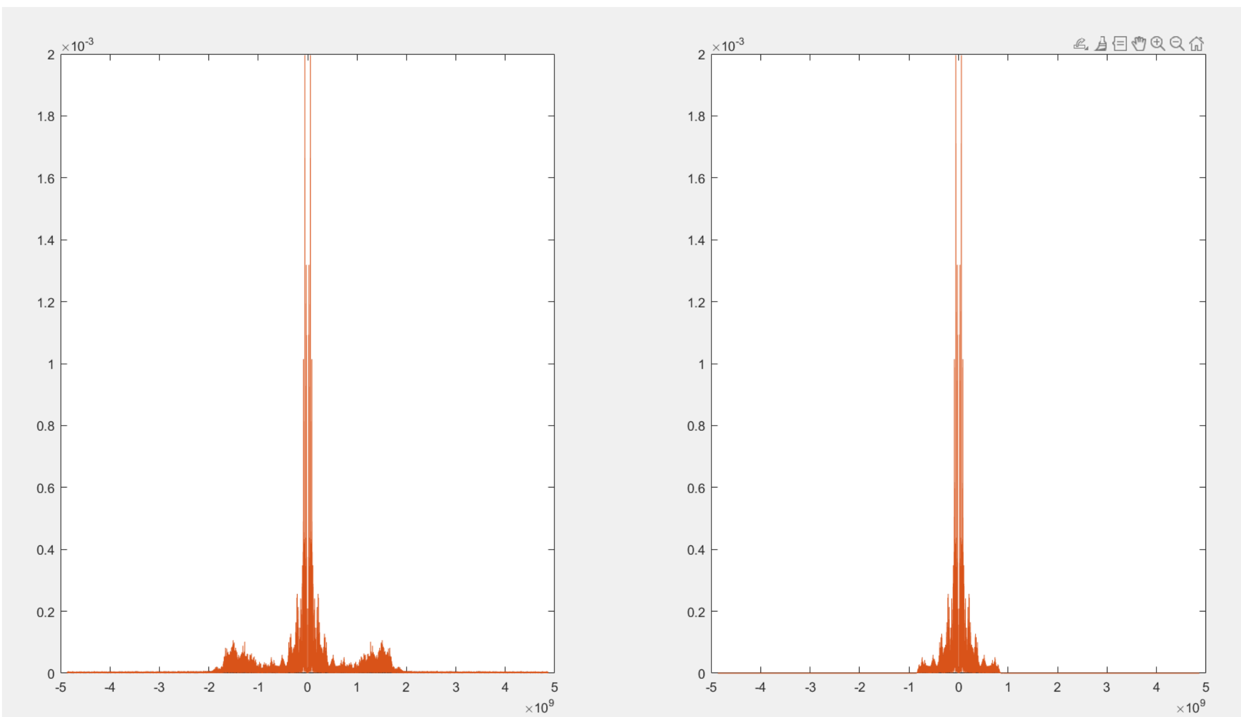


Figure 15.the magnitude spectrum of the first signal @4000 Hz.(input and output) (k)

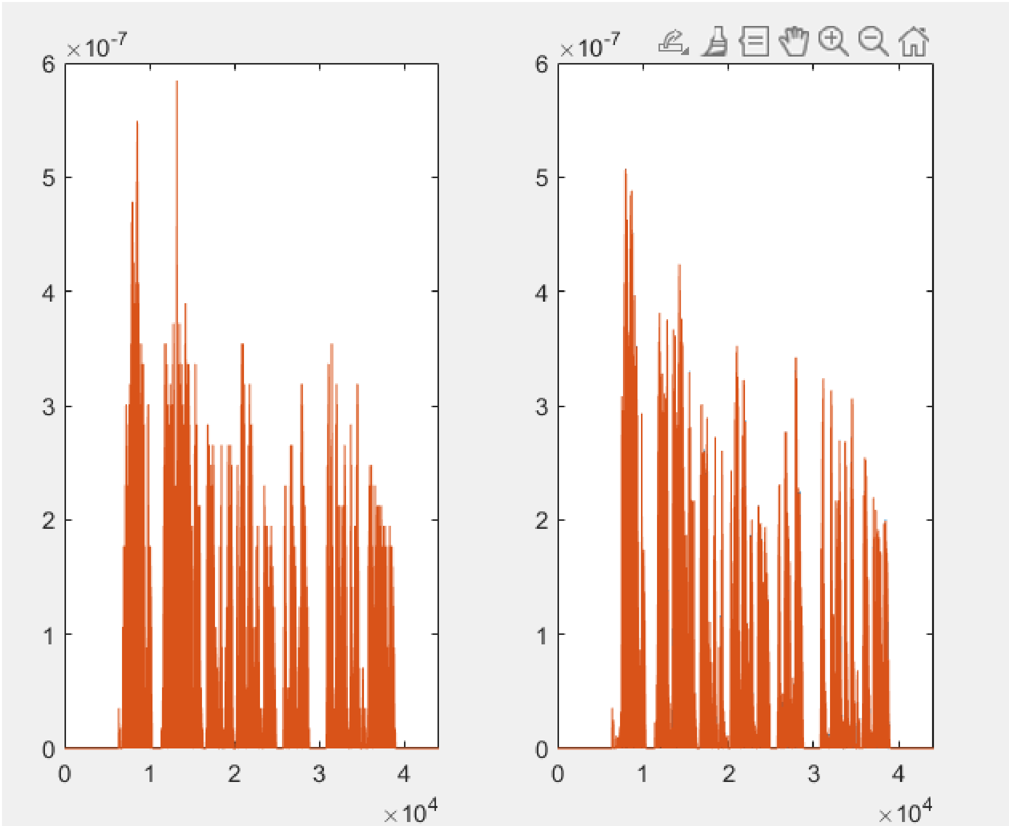


Figure 16.the magnitude spectrum of the first signal @4000 Hz.(input and output) (Hz)

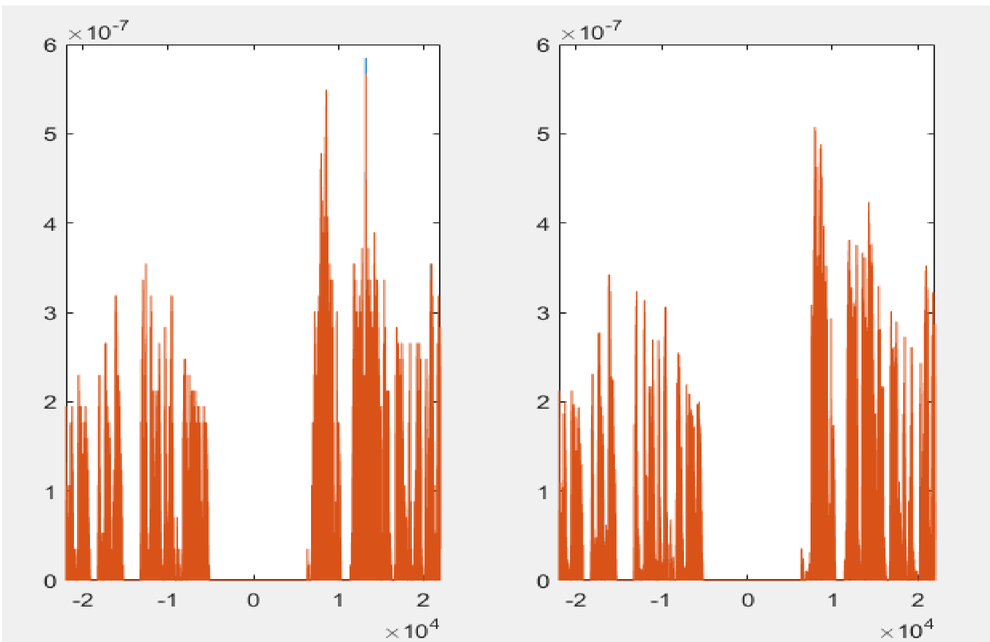


Figure 17 the magnitude spectrum of the first signal @4000 Hz.(input and output) (k) using fftshift'

The second signal:

the result at (20000 Hz):

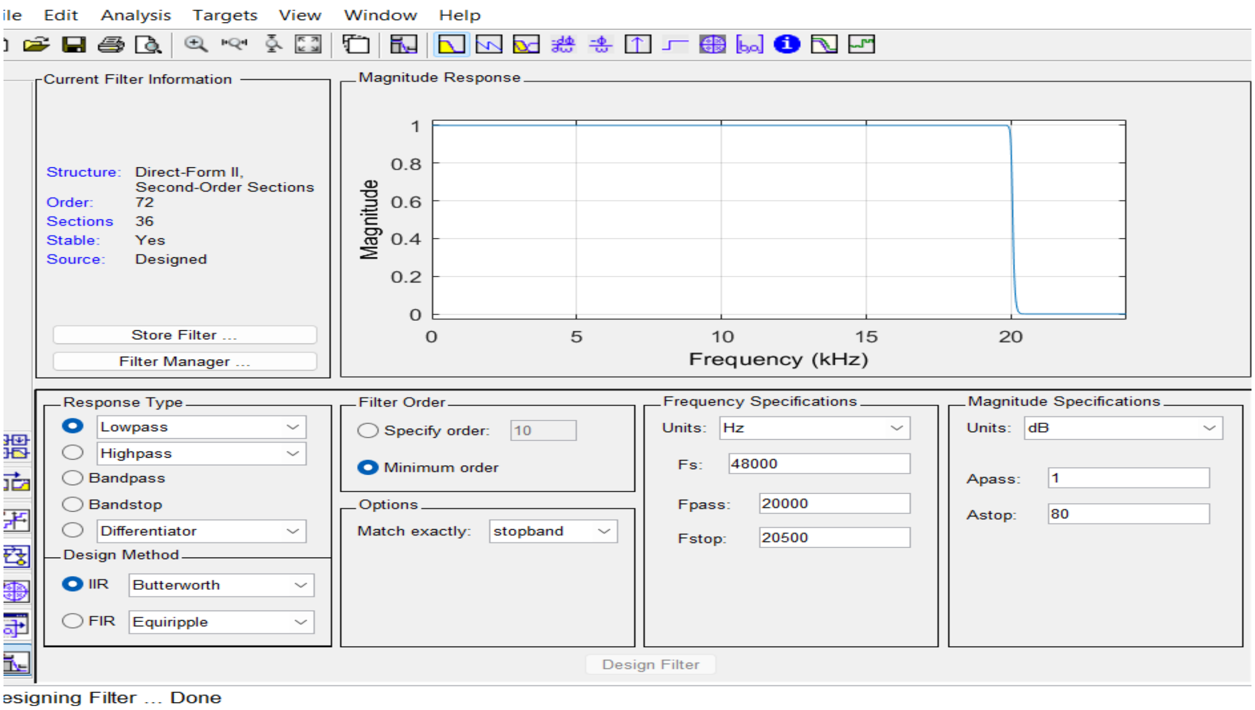


Figure 18.the frequency response of the filter

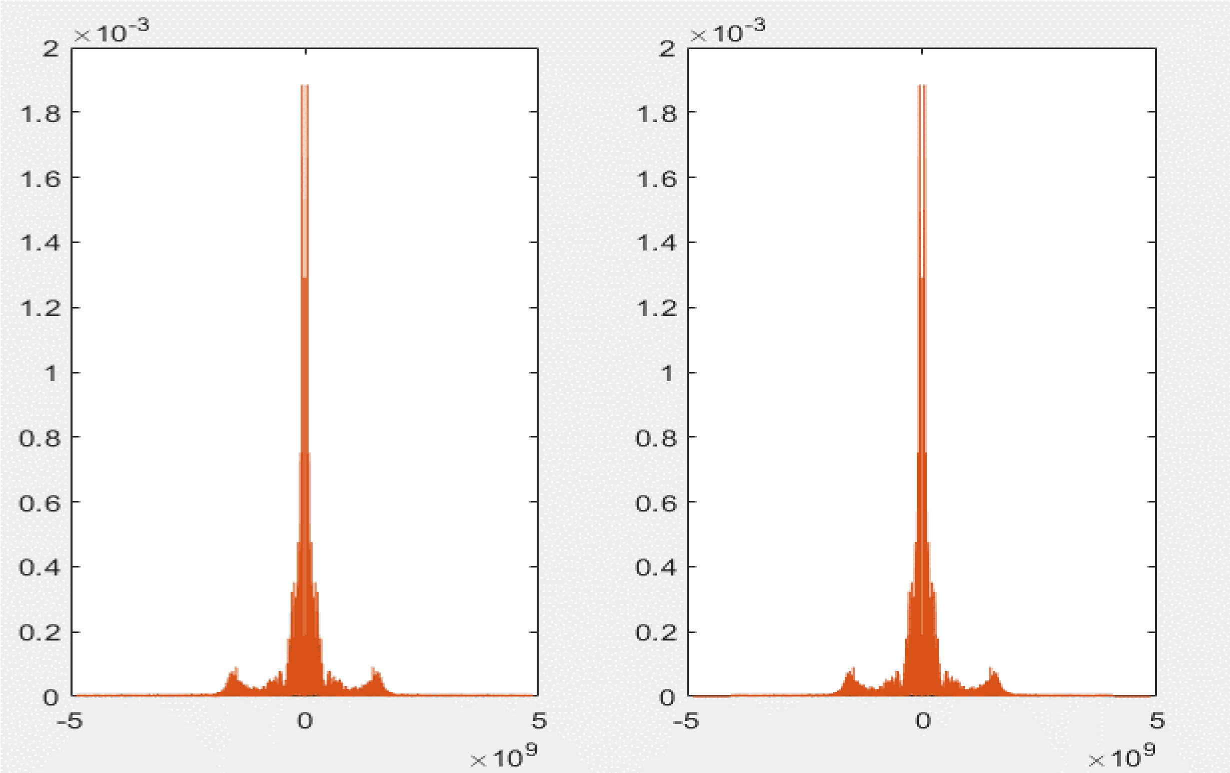


Figure 19.the magnitude spectrum of the second signal @20000 Hz.(input and output) in frequency response

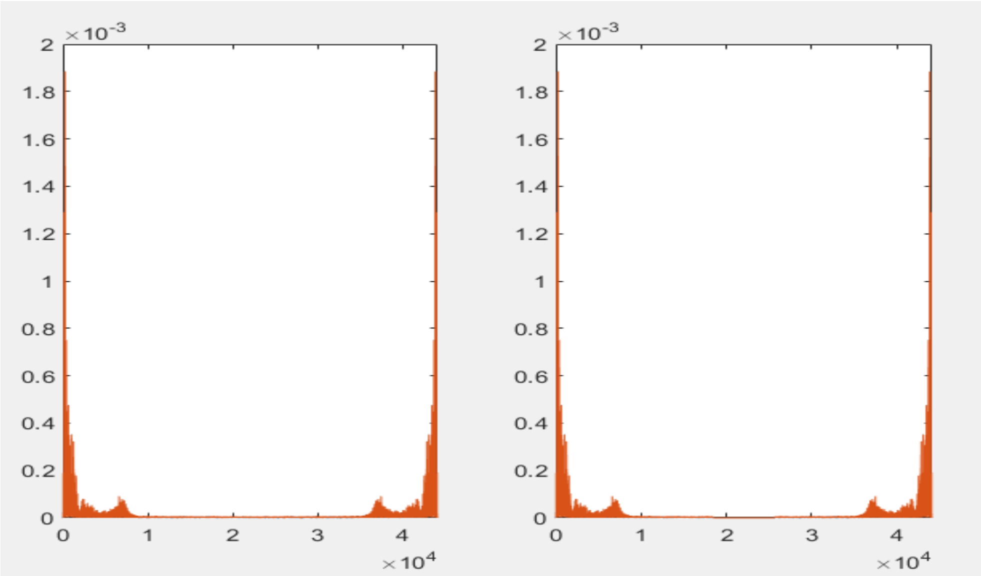


Figure 20.the magnitude spectrum of the second signal @20000 Hz.(input and output) (k)

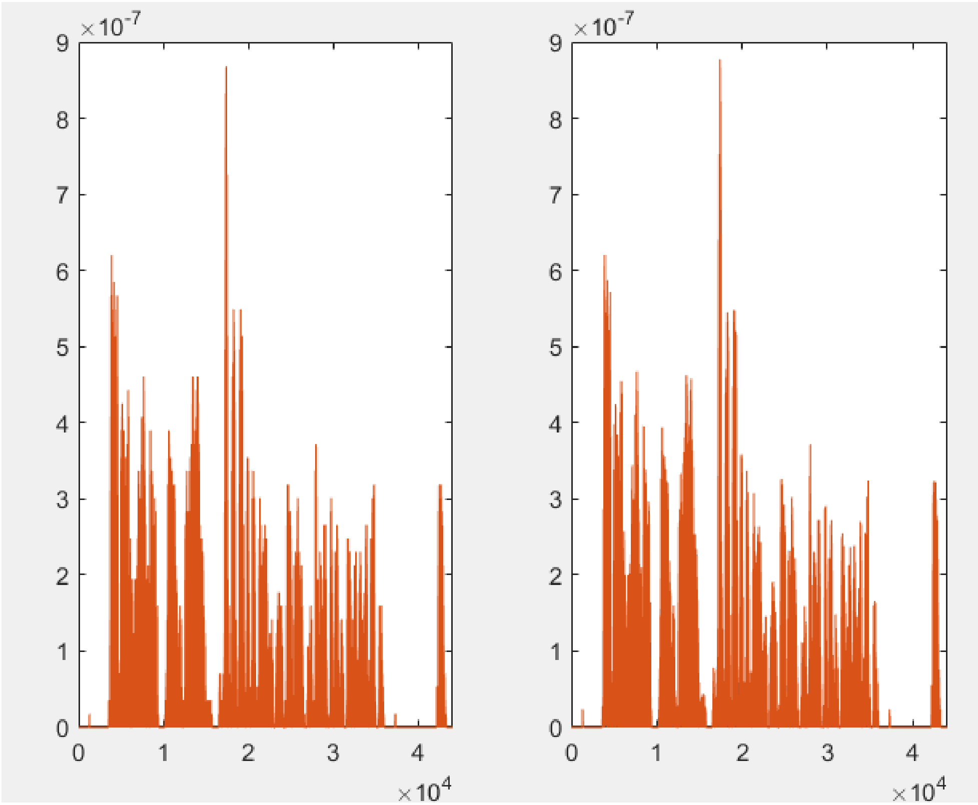


Figure 21.the magnitude spectrum of the second signal @20000 Hz.(input and output) (Hz)

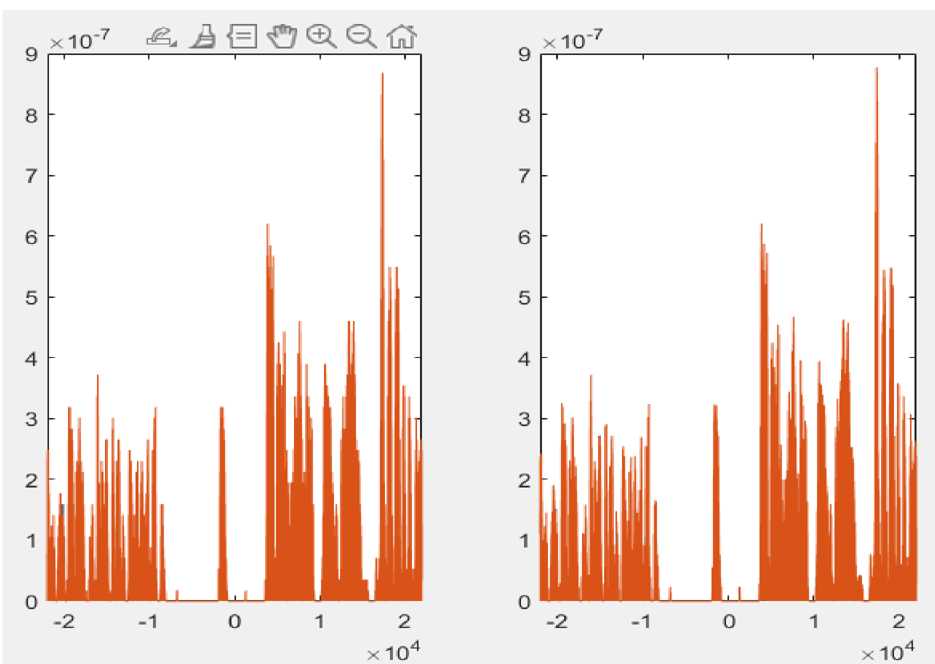


Figure 22.the magnitude spectrum of the second sgnal @20000 Hz.(input and output) using fftshift'

the results at (4000 Hz):

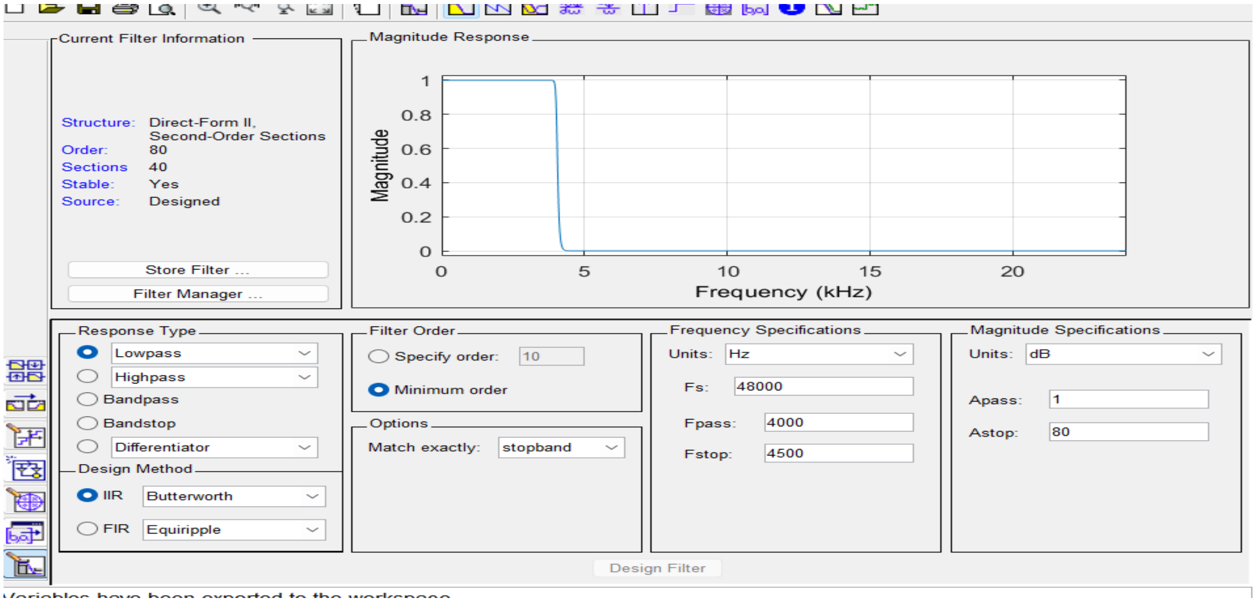


Figure 23.the frequency response of the filter(4000 hz)

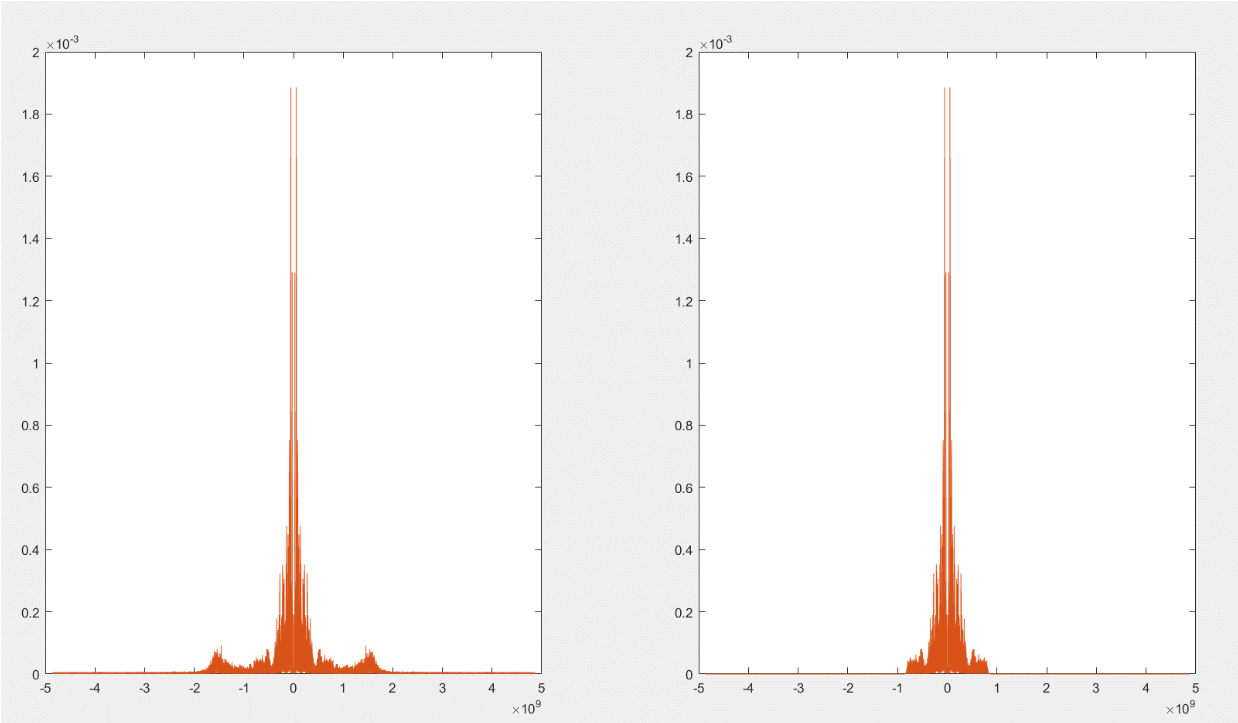


Figure 24.the magnitude spectrum of the second signal @4000 Hz.(input and output) in frequency response

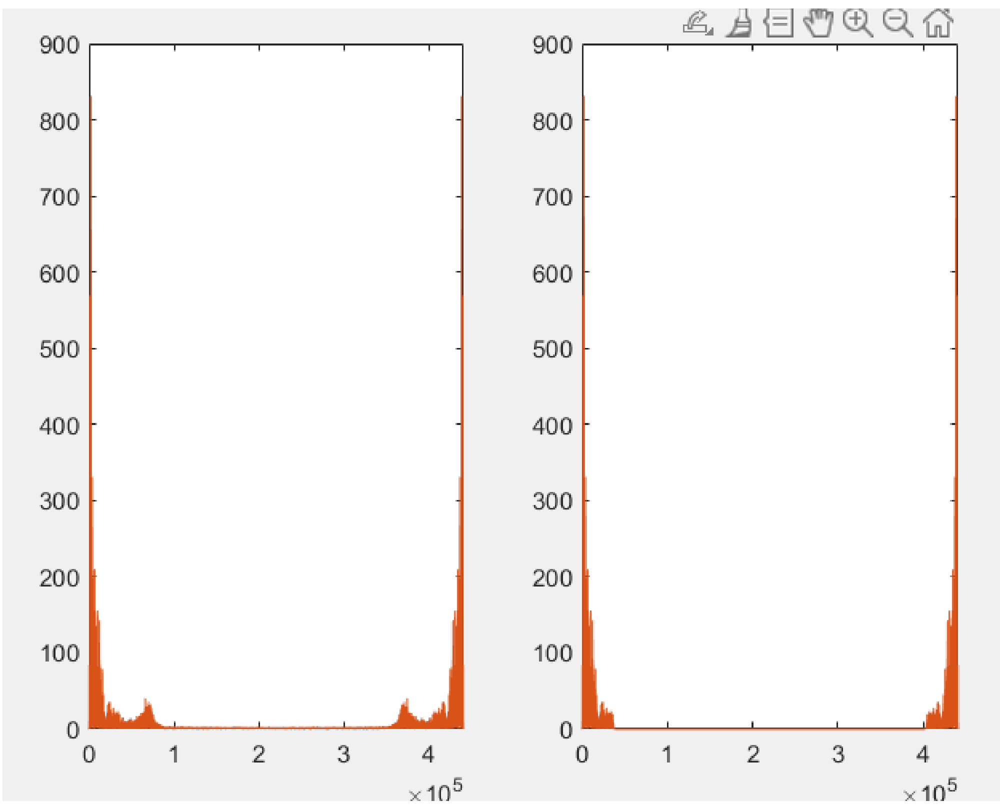


Figure 25.the magnitude spectrum of the second signal @4000 Hz(input and output) (k)

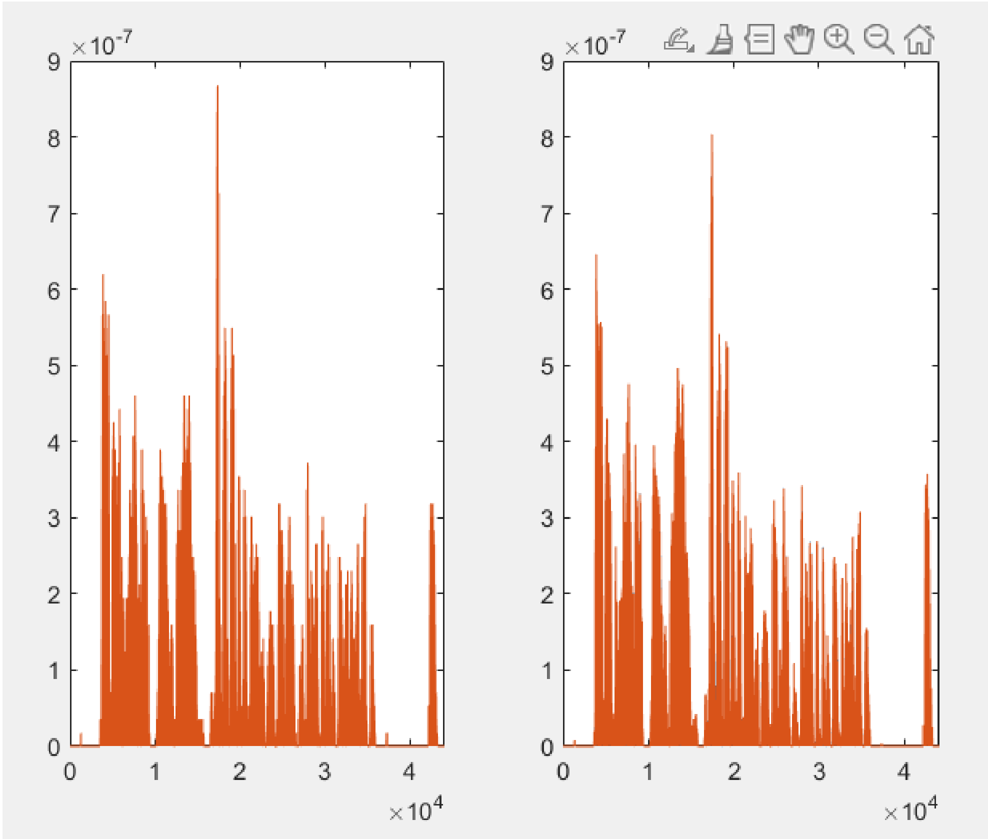


Figure 26 the magnitude spectrum of the second sgnal @4000 Hz.(input and output)

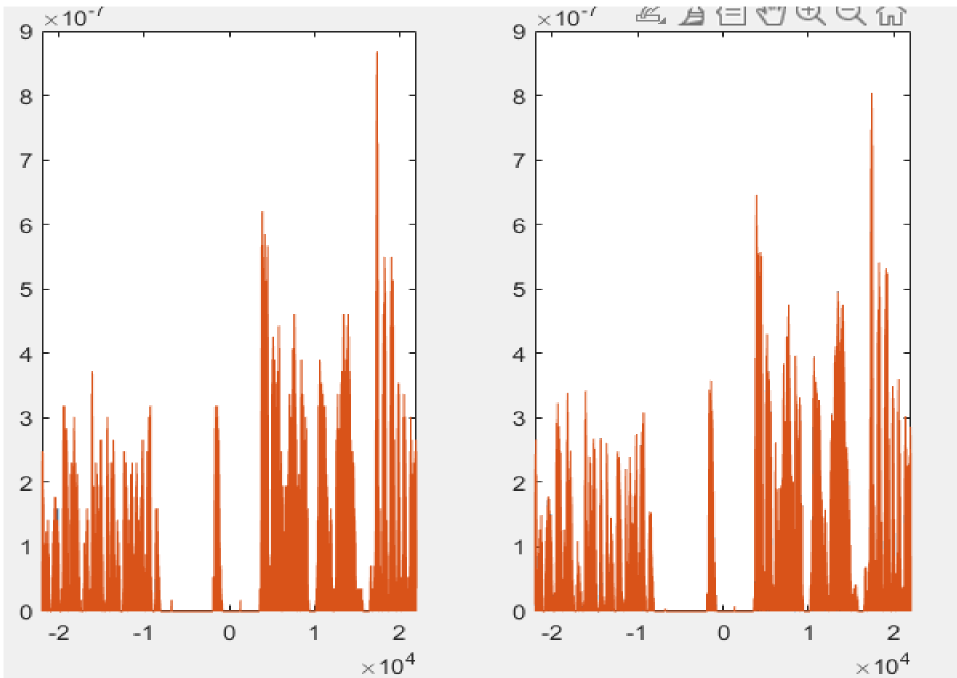


Figure 27 the magnitude spectrum of the first signal @4000 Hz.(input and output) (k) using fftshift'

For modulation (D&E):

$p\ (t)\ =\ cos\ (\omega 0t)$ *Equation 8*

$\boldsymbol{r\ (t)\ =\ s(t)\ p(t)}$ *Equation 9*

Where p(t) is the carrier signals(t) the signal and the r(t)is the modulated signal in time domain.

Explanation of my work:

First, we need to get the carrier signal which is (cos) as shown in *Equation 8* using section B of Amplitude modulation code, the we must make sure that the last frequency in the input signal as shown in section A of Amplitude modulation code (filtered signal with cut-off frequency 4000 Hz) does not interact with the carrier as shown in *Equation 10 and 11*.

$$\boldsymbol{R1(\omega) = \frac{1}{2} * S1(\omega - \omega o1) + \frac{1}{2} * S1(\omega + \omega o1)} \quad \textit{Equation 10}$$

$$\boldsymbol{R2(\omega) = \frac{1}{2} * S2(\omega - \omega o2) + \frac{1}{2} * S2(\omega + \omega o2)} \quad \textit{Equation 11}$$

$$\boldsymbol{Rtot(\omega) = R1(w) + R2(w)} \quad \textit{Equation 12}$$

Where S(w), P(w)and R(w) in frequency domain and ωo is the frequency of carrier and ω is the frequency of the signal.

Then we multiply the signals with different frequency and cosine as shown in *Equation 9*, then add each signal to get the modulation of two signal as shown in *Equation 12* using section C of Amplitude modulation code.

We must make sure ω0>ω1 to avoid any interaction between the modulated signal and the filtered signal.

Then we make the code tell the user which signal do you want to here (like the radio) as shown in section D of Amplitude modulation code, then after we chose the signal we design a band bass with right (Fpass & Fstop) to do the Demultiplexing of the selected signal as shown in *figure 28 and 29* using section E of Amplitude modulation code.

The we multiply the demultiplexed signal with carrier signal as shown in section F of Amplitude modulation code which have frequency we chose as shown in *figure 31*.

The last step is to design a low pass filter with gain 2 as shown in section H of Amplitude modulation code to get the output signal as shown in *figure 30*.

And this is the results.

Then, We play the signals to listen to them as shown in section Q of Amplitude modulation code

The frequency response of the bandpass filter (7500 Hz)

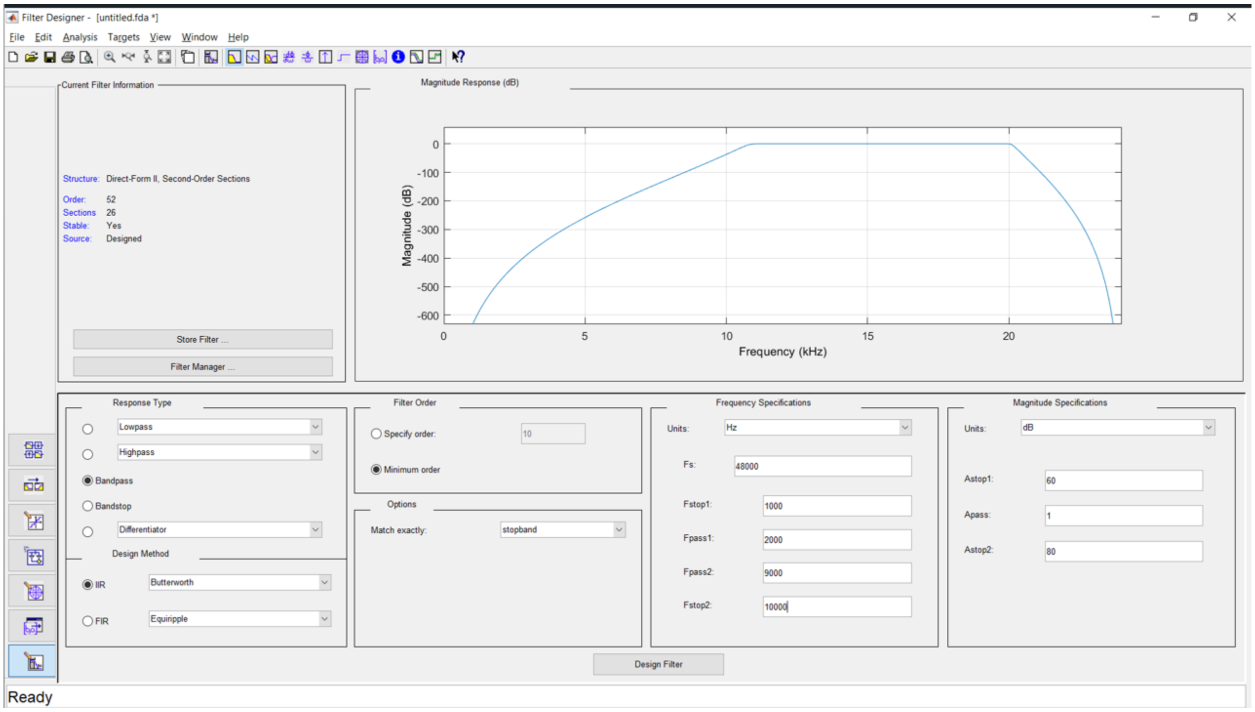


Figure 28 Band paas filter for channel 1

The frequency response of the bandpass filter (14500 Hz):

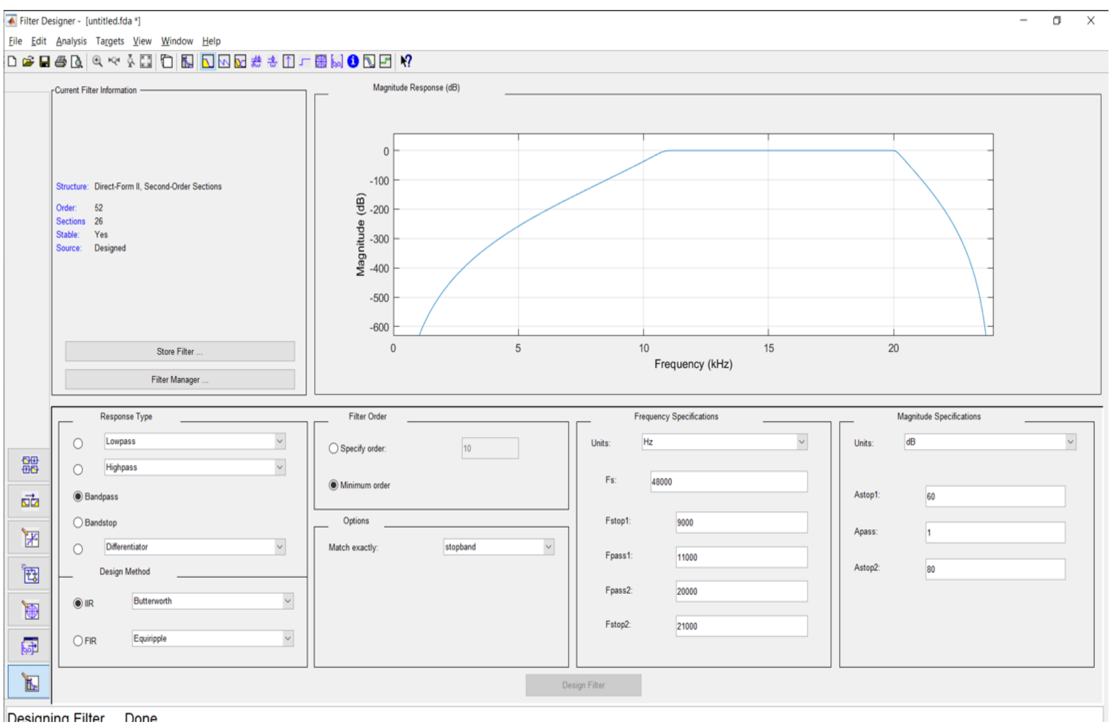


Figure 29 Band paas filter for channel 2

The frequency response of the lowpass filter (4000 hz):

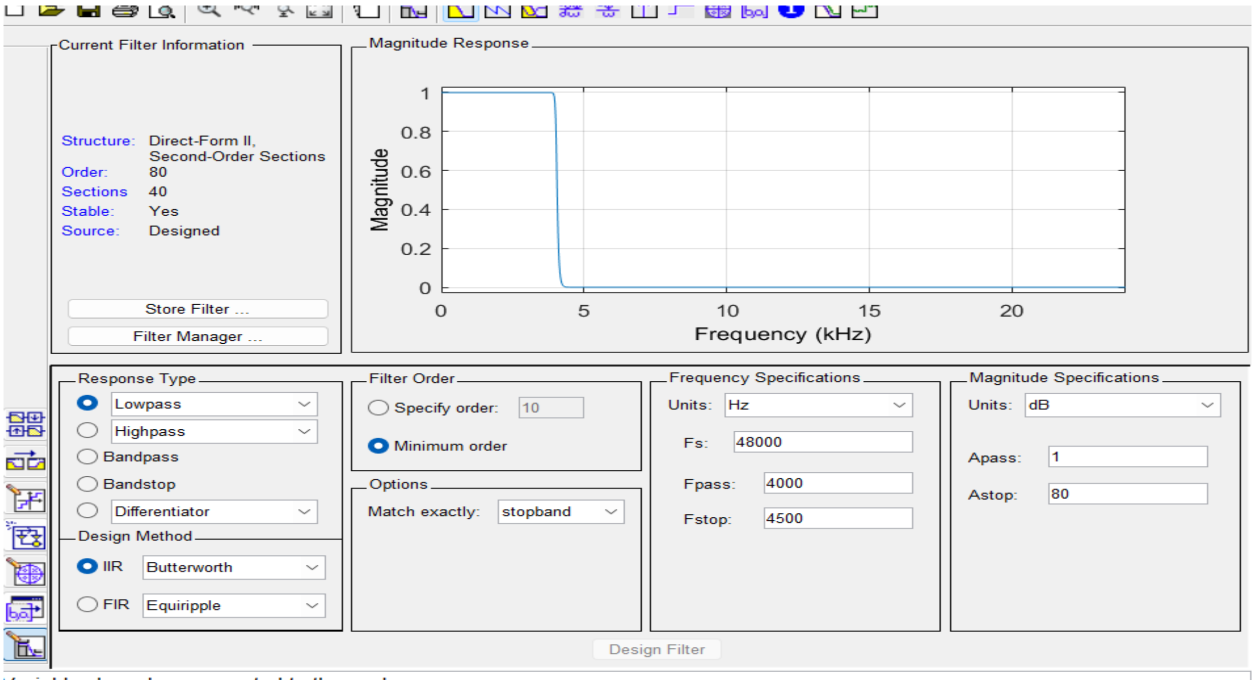


Figure 30 low paas filter for denoising

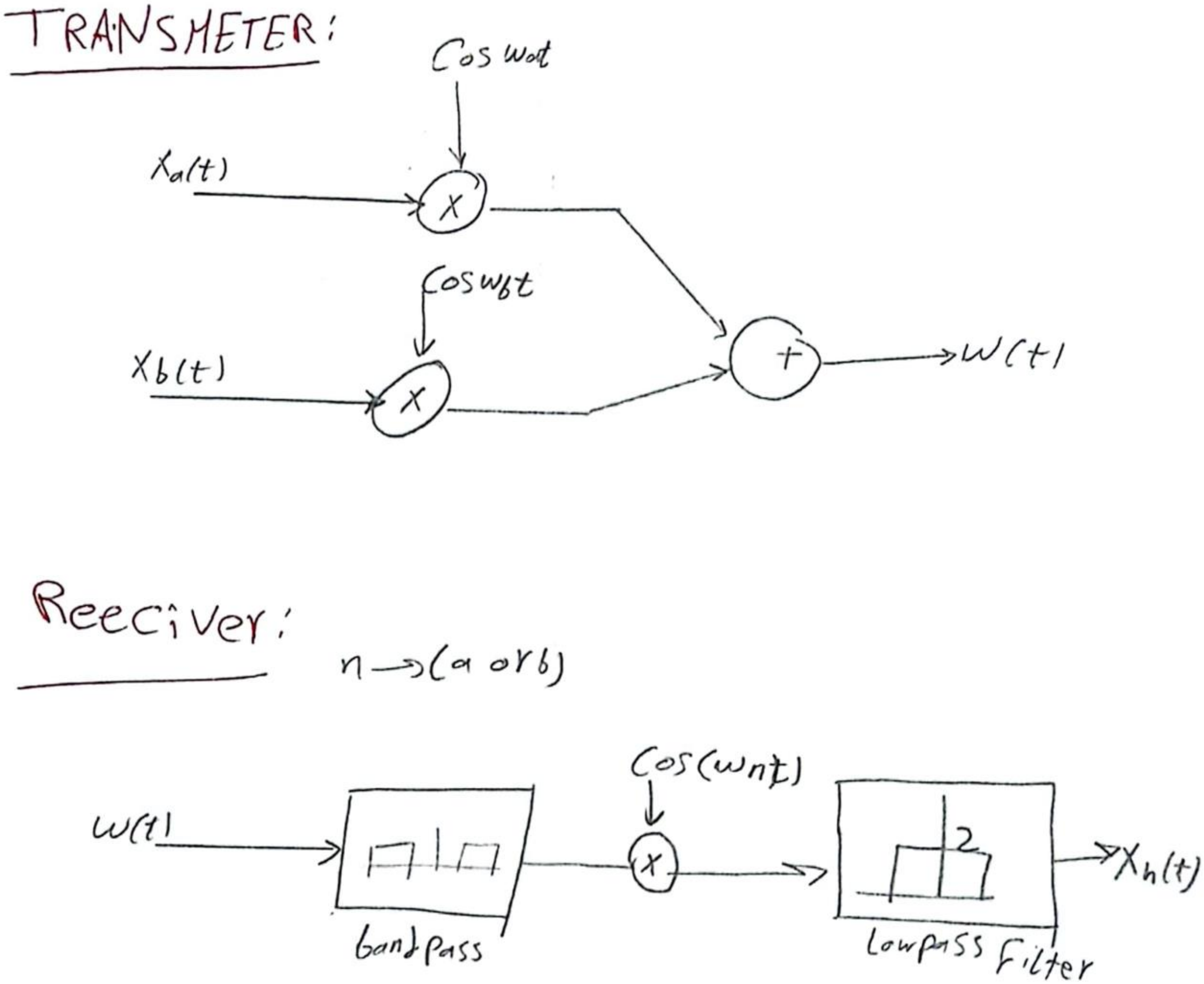
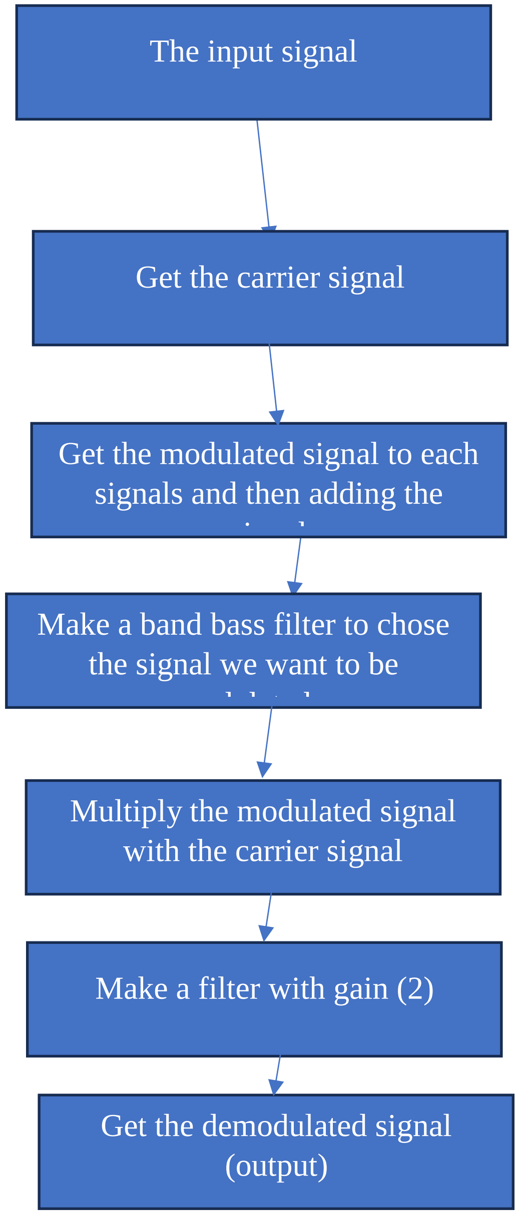
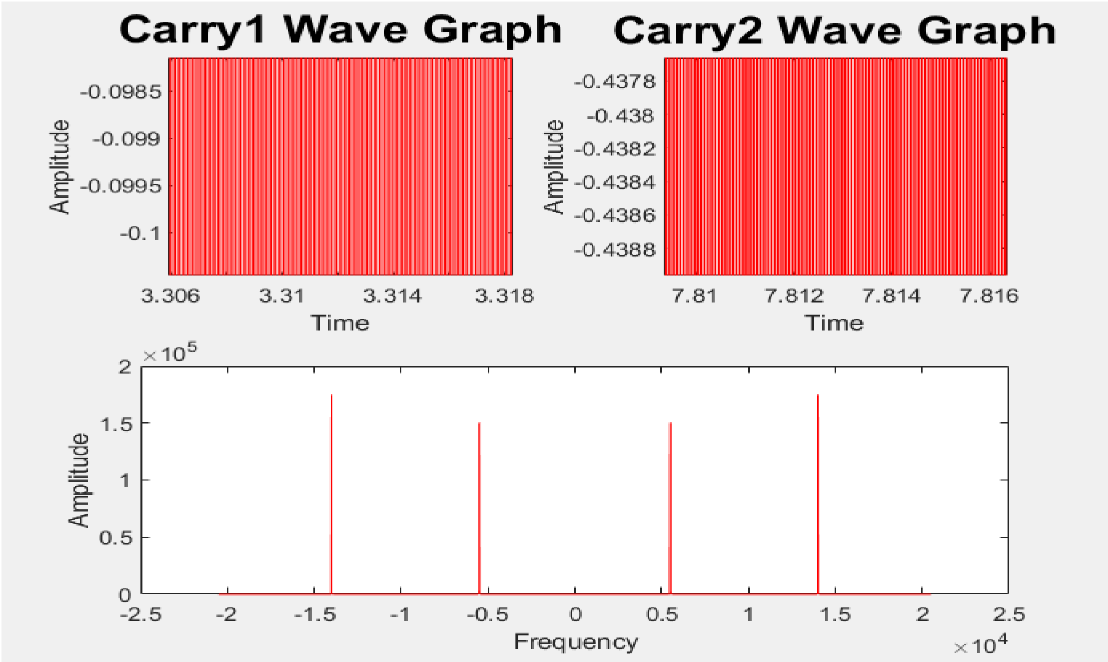


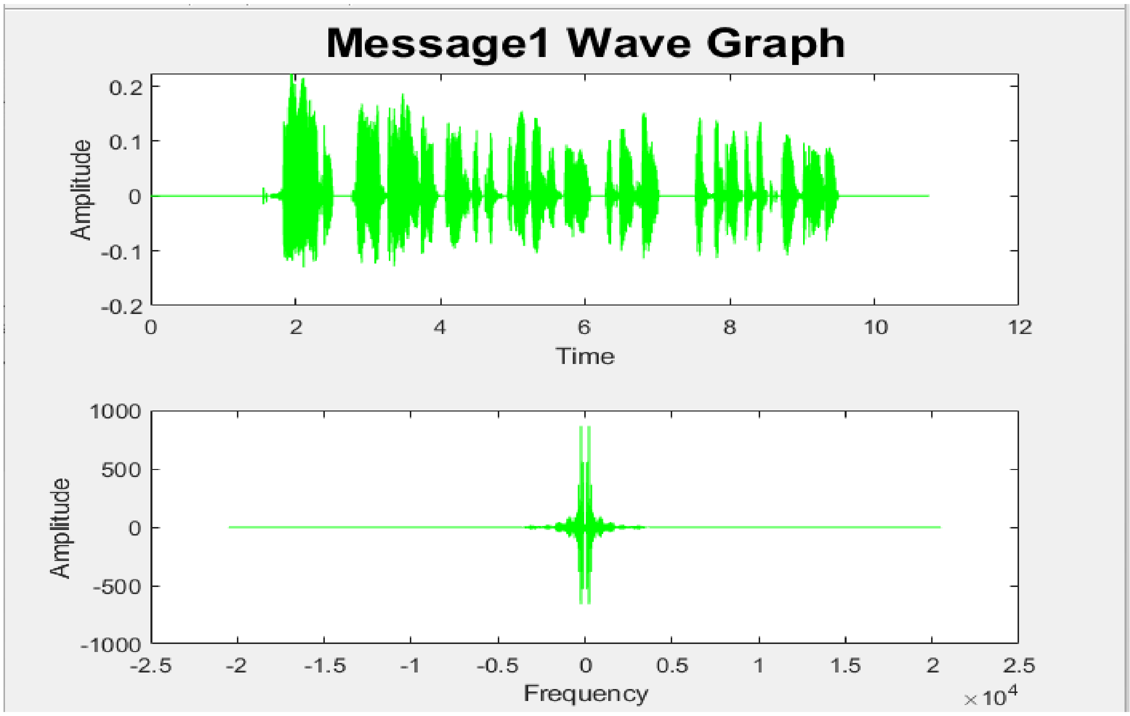
Figure 31 Transmitter and Receiver design

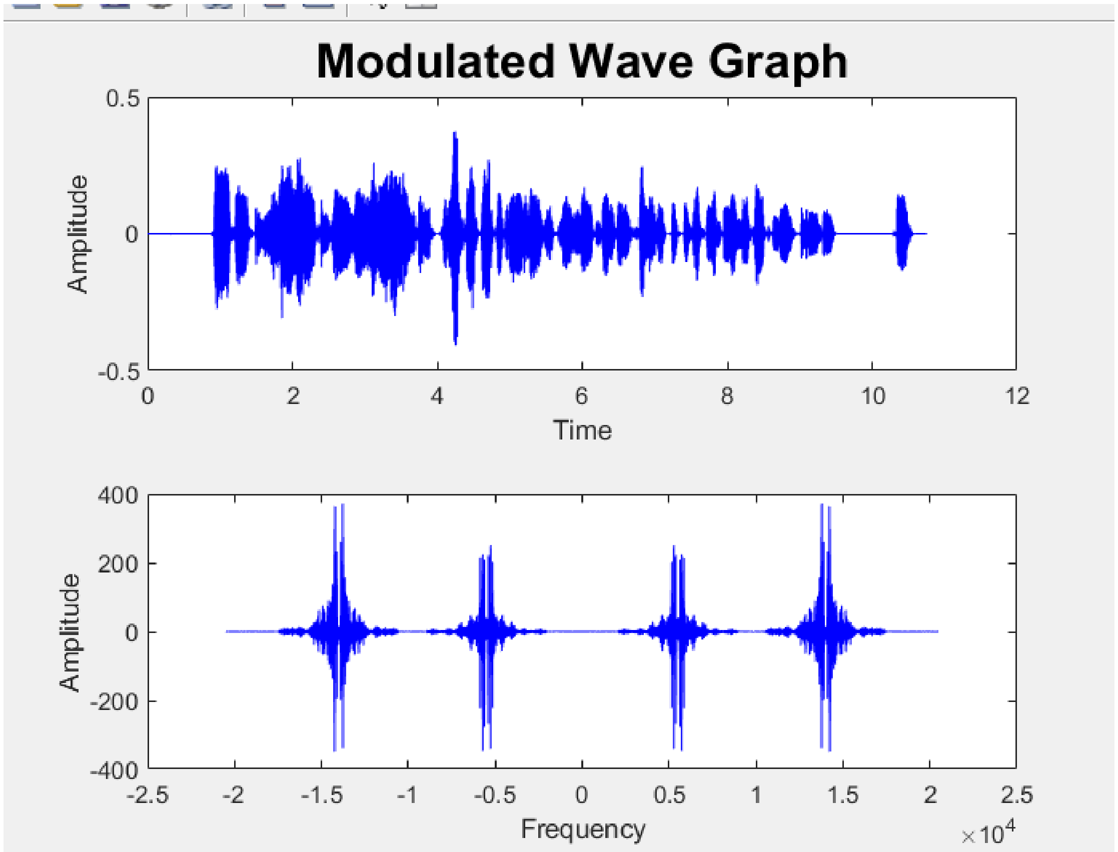
Explanation of the steps of work flow chart:

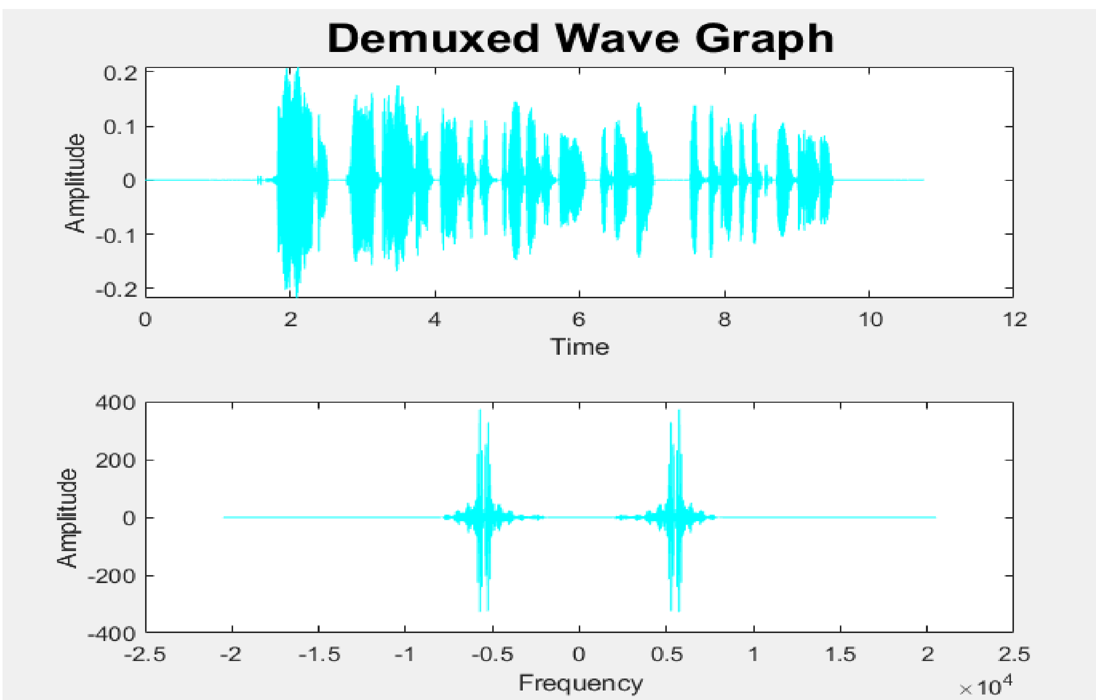


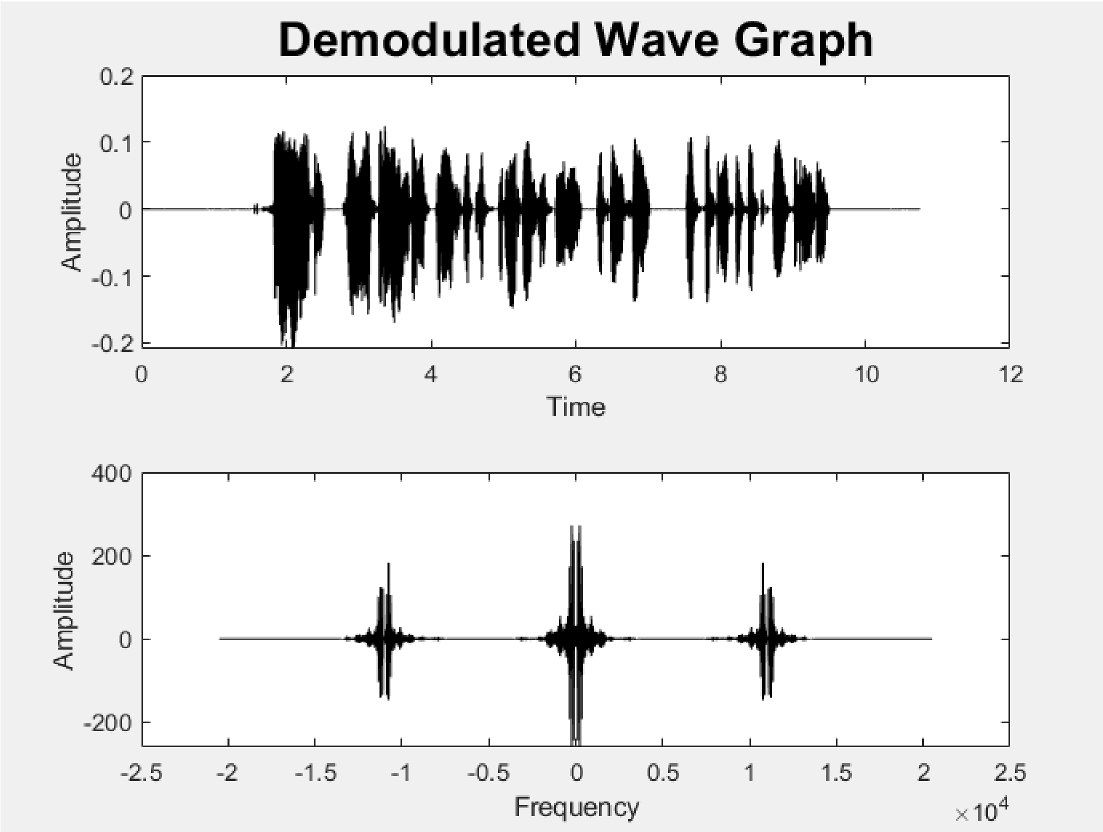
For channel 1:

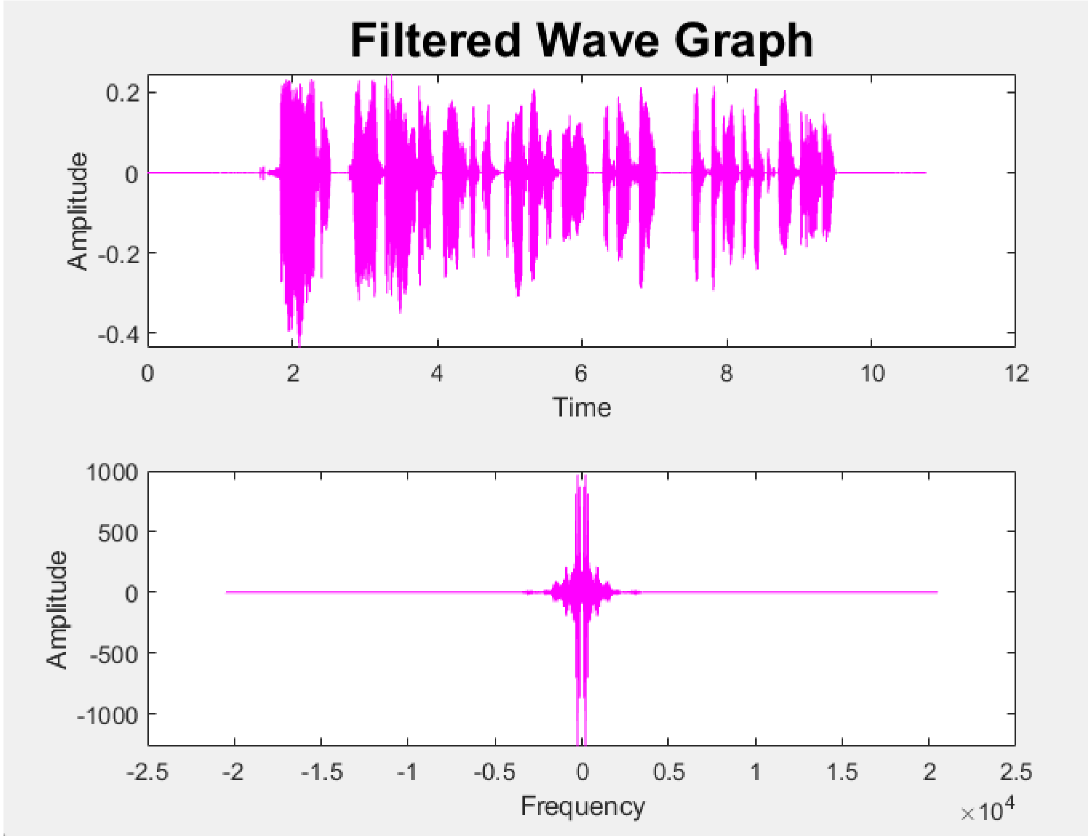




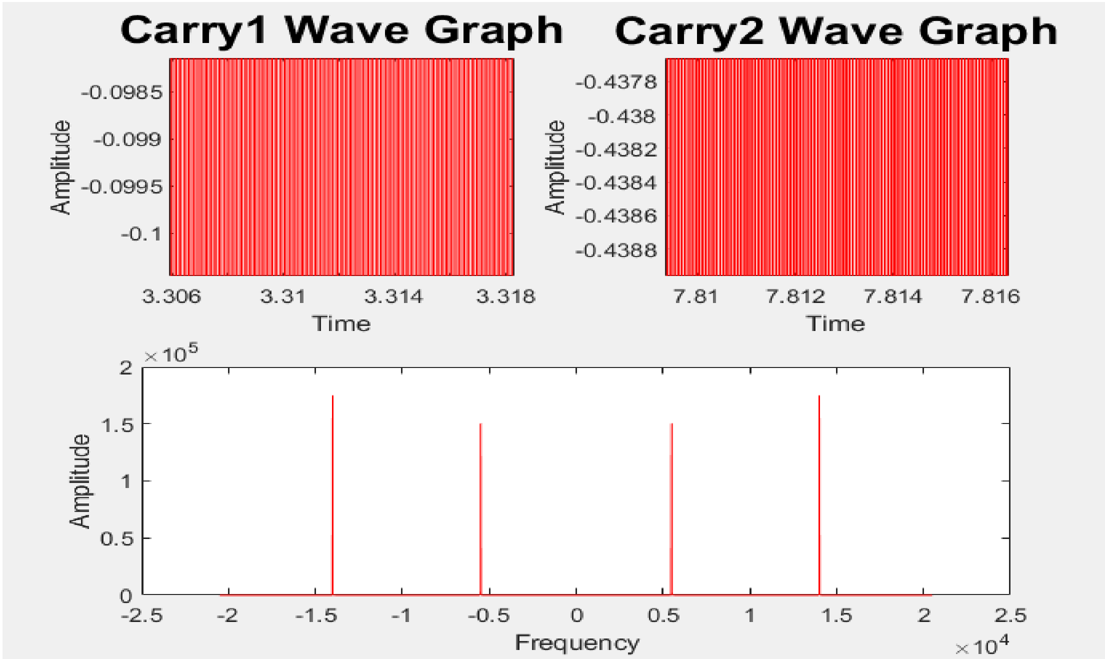

Modulated Wave Graph
Amplitude
Time
Amplitude
Frequency
×10^4


Demuxed Wave Graph
Amplitude
Time
Amplitude
Frequency
×10^4

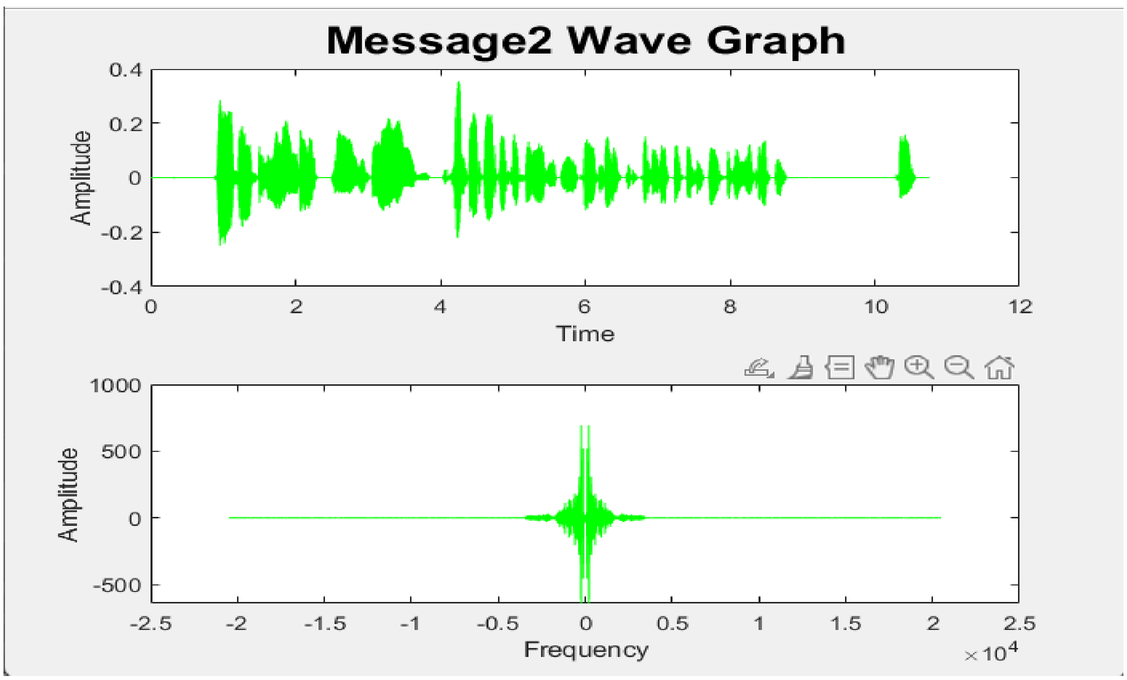




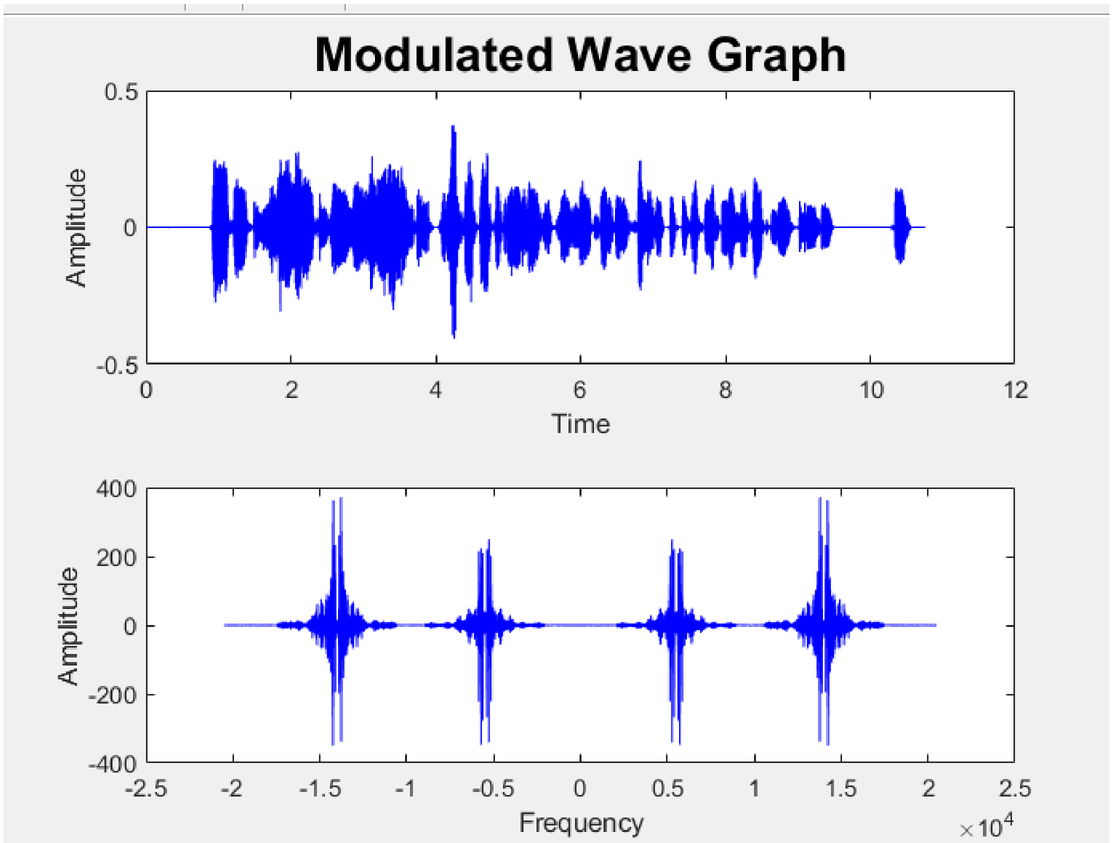
For channel 2:

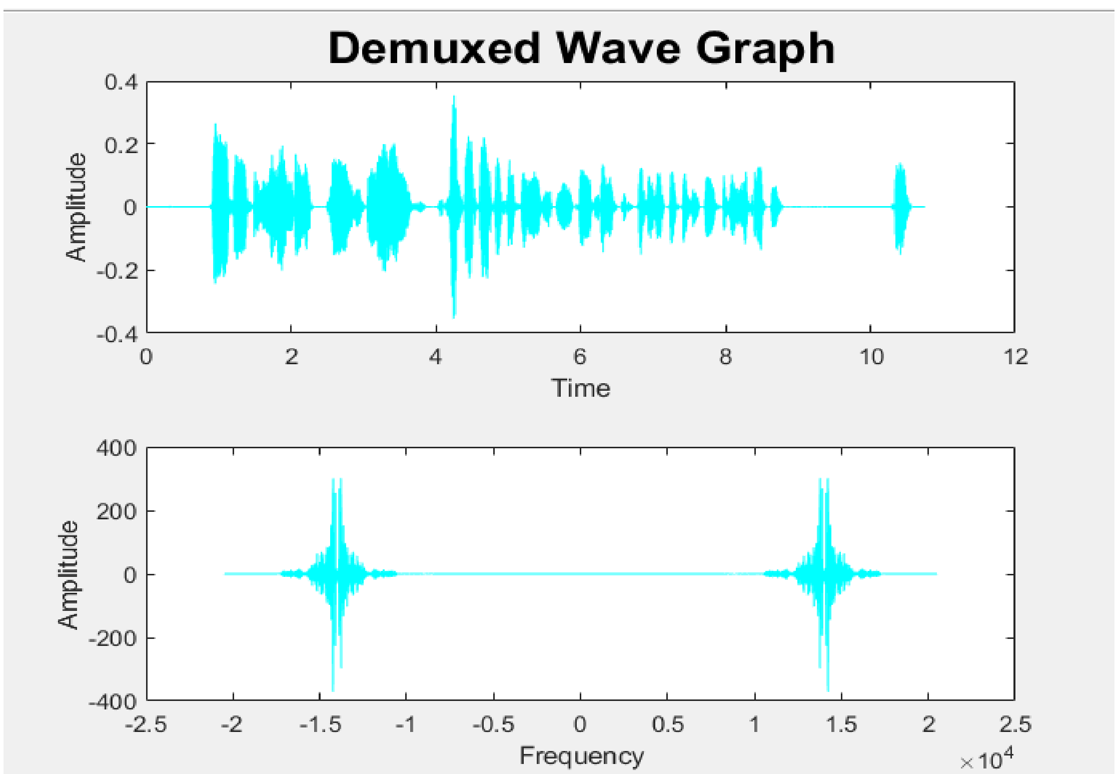


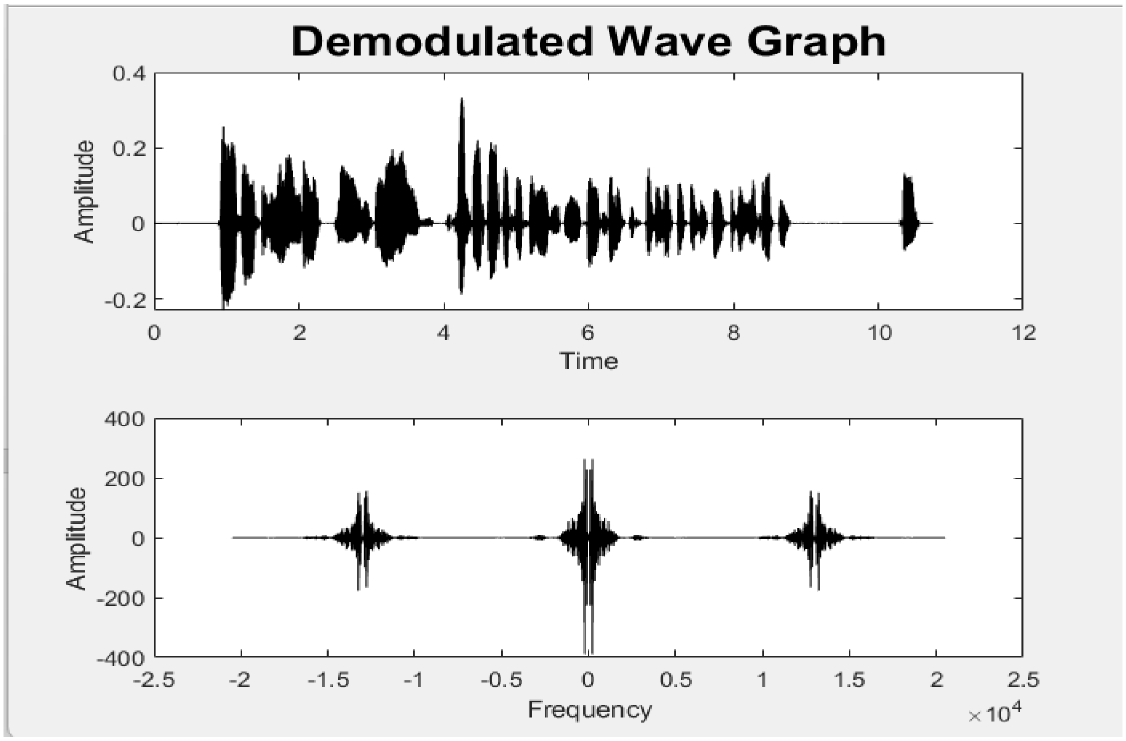
Carry1 Wave Graph
Carry2 Wave Graph
Amplitude
-0.0985
-0.099
-0.0995
-0.1
3.306
3.31
3.314
3.318
Time
-0.4378
-0.438
-0.4382
-0.4384
-0.4386
-0.4388
7.81
7.812
7.814
7.816
Time
×10^5
2
1.5
1
0.5
0
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequency
×10^4

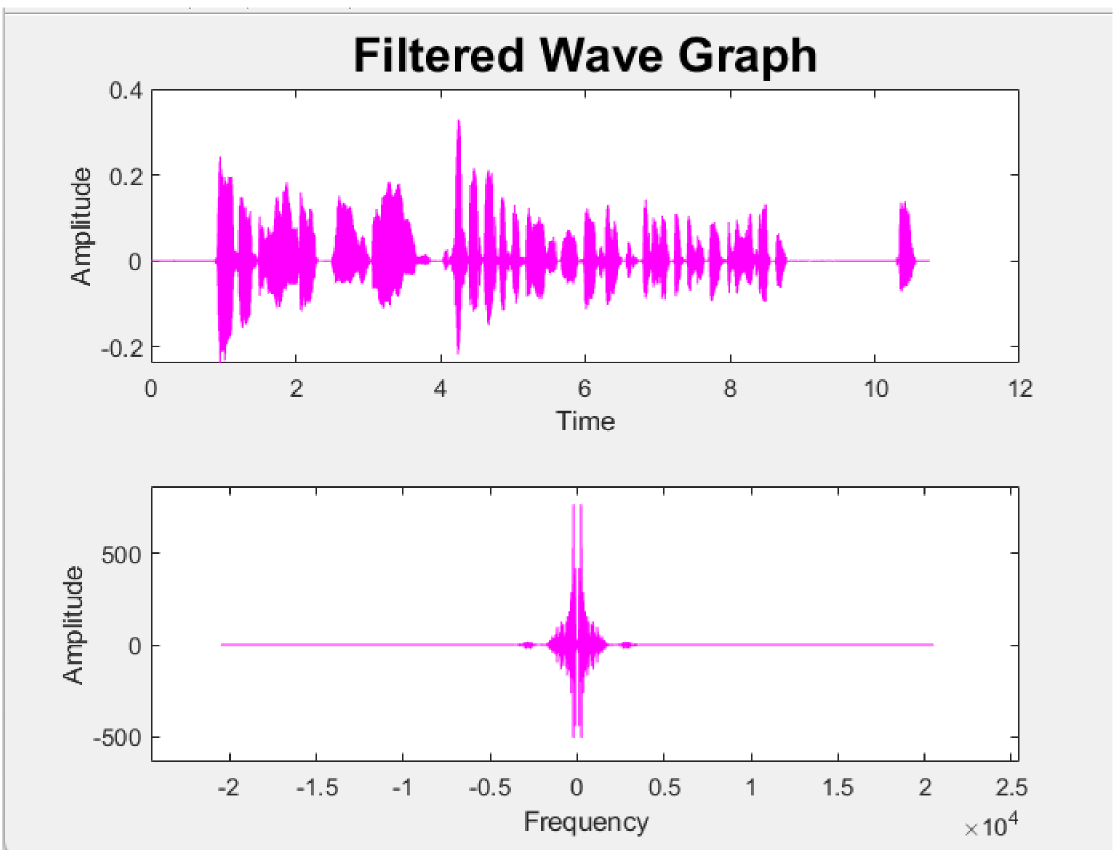


Message2 Wave Graph
Amplitude
0.4
0.2
0
-0.2
-0.4
0
2
4
6
8
10
12
Time
1000
500
0
-500
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5
Frequency
×10^4


Modulated Wave Graph
Amplitude
Time
Amplitude
Frequency
×10^4


Demuxed Wave Graph
Amplitude
Time
Amplitude
Frequency
×10^4





Why we chose the carrier frequency(7500 Hz)&(16000 Hz)?

- because we the last frequency in the input signal is equal to (4500 HZ).we use the signal the we get from filter function in low frequency (Fstop=4500 Hz).so we use (7500 Hz) or (14500) as a carrier frequency to avoid any interaction between the modulated signal and the filtered signal[7].

References:

[1] https://setosa.io/ev/image-kernels/

[2] https://www.youtube.com/watch?v=mbZeH9kdq3c&t=498s&ab_channel=ImageProcessing-JU

[3] https://en.wikipedia.org/wiki/Kernel_(image_processing)

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

[5] https://youtu.be/g3sfmjWkz5Q?si=Id_7uHqhuvtYGmvq

[6] https://youtube.com/shorts/9B8JbDSmFVE?si=eBBULLV_15JEGgXk

[7] https://youtu.be/IJuLmLyV-Sk?si=pFqW_1s3zZKZPjp-

Appendix

Image code:

```
clc          %clear the command window
clear        %clear the workspace to start
default =1; %if you want the default picture
%Section A
if default==1
    pic=imread('peppers.png');        %default
picture
else
    [file,path]=uigetfile('*.*');     %locate the
picture
    picp=strcat(path,file);           %picture path
= "path"+"file"
    pic=imread(picp);                 %read the
picture
end
show_colors(pic);                     %show all
colors

pause(3);                 %wait 3 seconds

%Section B
gray_pic=rgb2gray(pic);
%get the gray picture
edge_kernel=[0 1 0; 1 -4 1;0 1 0]/90
%edge kernel
edge_pic = conv2(gray_pic, edge_kernel);
%convelute the kernal with the picture
show_images(pic,edge_pic,"Orignal","Edge
detection");   %show the orignal and edge picture
imwrite(edge_pic,"image1.png");
%save the edge picture

pause(3);                 %wait 3 seconds
```

```
%Section C
sharp_kernel = [0, -1, 0; -1, 5, -1; 0, -1, 0]/255
%sharp kernel
sharp_pic(:,:,1) = conv2(pic(:,:,1), sharp_kernel);
%convelute the kernal with the red part of picture
sharp_pic(:,:,2) = conv2(pic(:,:,2), sharp_kernel);
%convelute the kernal with the green part of
picture
sharp_pic(:,:,3) = conv2(pic(:,:,3), sharp_kernel);
%convelute the kernal with the blue part of picture
show_images(pic,sharp_pic,"Orignal","Sharp");
%show the orignal and sharp picture
imwrite(sharp_pic,"image2.png");
%save the sharp picture

pause(3);                %wait 3 seconds

%Section D
blur_kernel=[1 2 1;2 4 2;1 2 1]/(16*255)
%blur kernel
blur_pic(:,:,1)=conv2(pic(:,:,1), blur_kernel);
%convelute the kernal with the red part of picture
blur_pic(:,:,2)=conv2(pic(:,:,2), blur_kernel);
%convelute the kernal with the green part of
picture
blur_pic(:,:,3)=conv2(pic(:,:,3), blur_kernel);
%convelute the kernal with the blue part of picture
show_images(pic,blur_pic,"Orignal","Blur");
%show the orignal and blur picture
imwrite(blur_pic,"image3.png");
%save the blur picture

pause(3);                %wait 3 seconds

%Section E
    motion_kernel_size=19;
    motion_kernel=[
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
```

```
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
    ]/(motion_kernel_size*255);
%motion kernel
motion_pic(:,:,1)=conv2(pic(:,:,1),motion_kernel);
%convelute the kernal with the red part of picture
motion_pic(:,:,2)=conv2(pic(:,:,2),motion_kernel);
%convelute the kernal with the green part of
picture
motion_pic(:,:,3)=conv2(pic(:,:,3),motion_kernel);
%convelute the kernal with the blue part of picture
[row,column] = size(motion_pic(:,:  ,1));
%take the pic size
%show the orignal and motion picture
show_images(pic,motion_pic(19:row-19,19:column-
19,:),"Orignal","Motion");
imwrite(motion_pic(19:row-19,19:column-
19,:),"image4.png");
%save the motion picture and we reduce the frame

pause(3);                    %wait 3 seconds

%Section F
pad_motion_kernel = zeros(row,column);
%intial the kernel
```

```
pad_motion_kernel(1: motion_kernel_size,1:
motion_kernel_size) = motion_kernel;
%making the pad

%Fourier transform
motion_kernel_fft(:,:,1)=fft2(pad_motion_kernel);
%fourier trasnform the motion lernel
motion_pic_fft(:,:,1)=fft2(motion_pic(:,:,1));
%fourier trasnform the red part
motion_pic_fft(:,:,2)=fft2(motion_pic(:,:,2));
%fourier trasnform the green part
motion_pic_fft(:,:,3)=fft2(motion_pic(:,:,3));
%fourier trasnform the blue part

RGB_scale=255*1e-19;
orignal_pic_fft(:,:,1) =motion_pic_fft(:,:,1)
./(motion_kernel_fft+RGB_scale);
%getting the orignal red part
orignal_pic_fft(:,:,2) =motion_pic_fft(:,:,2)
./(motion_kernel_fft+RGB_scale);
%getting the orignal green part
orignal_pic_fft(:,:,3) =motion_pic_fft(:,:,3)
./(motion_kernel_fft+RGB_scale);
%getting the orignal blue part

orignal_pic=255*ifft2(orignal_pic_fft);
%inverse fourier transform
show_images(motion_pic(19:row-19,19:column-
19,:),uint16(orignal_pic(19:row-19,19:column-
19,:)),"Motion","Orignal");
%show the motion and orignal picture
imwrite(uint16(orignal_pic(19:row-19,19:column-
19,:)),"image5.png");
%save the orignal picture and we reduce the frame
```

Communication code:

```
clear;
clc;
load filters;
%section A %%%%%%%
%get the first input
[file1,path1]=uigetfile('*.*');  %The file
location
mp1=strcat(path1,file1);         %The file path
[Ai1 ,fi1]=audioread(mp1);       %Read the file
om1=audioplayer(Ai1,fi1);    %make orignal audio
object

%get the seond input
[file2,path2]=uigetfile('*.*');  %The file
location
mp2=strcat(path2,file2);         %The file path
[Ai2 ,fi2]=audioread(mp2);       %Read the file
om2=audioplayer(Ai2,fi2);    %make orignal audio
object

Ai1=low_filter(Ai1,fi1,1,low_pass);
Ai2=low_filter(Ai2,fi2,2,low_pass);
fm1=audioplayer(Ai1,fi1);   %make filter audio
object
fm2=audioplayer(Ai2,fi2);   %make filter audio
object
disp("first input");
play(om1);                 %play orignal audio
pause(10);                %it plays it for 10 seconds
stop(om1);                 %stop orignal audio
disp("first input filtered");
play(fm1);                 %play orignal audio
pause(10);                %it plays it for 10 seconds
stop(fm1);                 %stop orignal audio
disp("second input");
```

```
play(om2);                 %play orignal audio
pause(10);                %it plays it for 10 seconds
stop(om2);                 %stop orignal audio
disp("second input filtered");
play(fm2);                 %play orignal audio
pause(10);                %it plays it for 10 seconds
stop(fm2);                 %stop orignal audio
audiowrite("output1filter.wav",Ai1,fi1); %save the
filtered file
audiowrite("output2filter.wav",Ai2,fi2); %save the
filtered file
%%%%%%%%%

%section Bs %%%%%%%%
% first we will get the carrier signal in frequincy
domain
Ai1=Ai1(:,1);                  %make it mono
Ai2=Ai2(:,1);                  %make it mono
A=1;                           %Carry amplitude
Fs=41000;                      %Sample frequency
fc1=5500;                      %Carry frequency
fc2=14000;                     %Carry frequency
Ts=1/Fs;                       %Sample period
N=length(Ai1);                 %Number of samples
t=[0:Ts:N*Ts-Ts];              %Time vector
f=[-Fs/2:Fs/N:Fs/2-Fs/N];      %Frequency vector
x1=A*cos(2*pi*fc1*t);          %Carry

%second carrier signal
x2=A*cos(2*pi*fc2*t);

%Carry graph
figure();                      %opens a figure
subplot(2,2,1);                %divide it into half
and plot the first half
plot(t,x1,'r');                %plot carry in time
domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Carry1 Wave Graph",'FontSize',18);
```

```
subplot(2,2,2);                     %divide it into half
and plot the first half
plot(t,x2,'r');                     %plot carry in time
domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Carry2 Wave Graph",'FontSize',18);

subplot(2,1,2);                          %second half
xft=fft(x1)+fft(x2);                     %Fourier transform
plot(f,fftshift(abs(xft)),'r');  %plot carry in
frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);

%Input1 graph
figure();                              %opens a figure
subplot(2,1,1);                        %divide it into half
and plot the first half
plot(t,Ai1,'g');                       %plot input in time
domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Message1 before filter Wave
Graph",'FontSize',18);
subplot(2,1,2);                        %second half
Ai1_fft=fft(Ai1);                      %Fourier transform
plot(f,fftshift(Ai1_fft),'g'); %plot input in
frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);

%Input2 graph
figure();                              %opens a figure
subplot(2,1,1);                        %divide it into half
and plot the first half
plot(t,Ai2,'g');                       %plot input in time
domain
xlabel("Time",'FontSize',10);
```

```
ylabel("Amplitude",'FontSize',10);
title("Message2 Wave before filter
Graph",'FontSize',18);
subplot(2,1,2);                    %second half
Ai2_fft=fft(Ai2);                  %Fourier transform
plot(f,fftshift(Ai2_fft),'g'); %plot input in
frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
%%%%%%%%%%%%%%%%%

%secion C %%%%%%%%%%%%%%%%%
%Modulate
Ym=transpose(x1).*Ai1+transpose(x2).*Ai2;
figure();                        %opens a figure
subplot(2,1,1);                  %divide it into half
and plot the first half
plot(t,Ym,'b');                  %plot modulated siganl
in time domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Modulated Wave Graph",'FontSize',18);
subplot(2,1,2);                  %second half
Ym_fft=fft(Ym);                  %Fourier transform
plot(f,fftshift(Ym_fft),'b');%plot modulated siganl
in frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
%%%%%%%%%%%%%%%%%

%section D %%%%%%%%%%%%%%%%%
%select which chanel
ch=0;
while ch ~=1 && ch~=2
ch=input("Which channel do you want?\n 1 or 2\n");
if ch ~=1 && ch~=2
disp("Invalid input");
end
end
```

```
%Channel selection
if ch==1
xs=x1;
As=Ai1;
fsel=fi1;
outname="1";
Hds=band_pass1;
filter_coef=4;
else if ch==2
        xs=x2;
        As=Ai2;
        fsel=fi2;
        outname="2";
        Hds=band_pass2;
        filter_coef=2;
    end
end
%%%%%%%%%%%%%%%%%

%section E %%%%%%%%%%%%%%%%%
%demultiplexing
Ydemux=filter(Hds,Ym);
figure();                             %opens a figure
subplot(2,1,1);                       %divide it into half
and plot the first half
plot(t,Ydemux,'c');                       %plot modulated
siganl in time domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Demuxed Wave Graph",'FontSize',18);
subplot(2,1,2);                       %second half
Ydemux_fft=fft(Ydemux);                       %Fourier
transform
plot(f,fftshift(Ydemux_fft),'c');%plot modulated
siganl in frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
%%%%%%%%%%%%%%%%%

%section F %%%%%%%%%%%%%%%%%
```

```
Yd=Ydemux.*transpose(xs);
figure();                               %opens a figure
subplot(2,1,1);                         %divide it into
half and plot the first half
plot(t,Yd,'k');                         %plot demodulated
siganl in time domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Demodulated Wave Graph",'FontSize',18);
subplot(2,1,2);                         %second half
Yd_fft=fft(Yd);                         %Fourier transform
plot(f,fftshift(Yd_fft),'k');   %plot demodulated
siganl in frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
%%%%%%%%%%%%%%%%%

%section H %%%%%%%%%%%%%%%%%
% filter with gain 2
Yf=filter_coef*filter(low_pass,Yd);
figure();                               %opens a figure
subplot(2,1,1);                         %divide it into
half and plot the first half
plot(t,Yf,'m');                         %plot filtered
siganl in time domain
xlabel("Time",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Filtered Wave Graph",'FontSize',18);
subplot(2,1,2);                         %second half
Yr_fft=fft(Yf);                         %Fourier transform
plot(f,fftshift(Yr_fft),'m');   %plot demodulated
siganl in frequency domain
xlabel("Frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
%%%%%%%%%%%%%%%%%

%section K %%%%%%%%%%%%%%%%%
%let everthing stereo
xs=transpose(xs);
xs(:,2)=xs(:,1);
```

```
As(:,2)=As(:,1);
Ym(:,2)=Ym(:,1);
Yd(:,2)=Yd(:,1);
Yf(:,2)=Yf(:,1);


%listen to eveything
cm=audioplayer(xs,fsel);
im=audioplayer(As,fsel);
mm=audioplayer(Ym,fsel);
dmux=audioplayer(Ydemux,fsel);
dm=audioplayer(Yd,fsel);
fm=audioplayer(Yf,fsel);
disp("Carry");
%play(cm); %Warning
%pause(11);
disp("Input");
play(im);
pause(11);
disp("Modulate");
play(mm);
pause(11);
disp("Demux");
play(dmux);
pause(11);
disp("Demodulate");
play(dm);
pause(11);
disp("Filtered");
play(fm);
pause(11);
%%%%%%%%%%%%%%%%%

%section Q %%%%%%%%%%%%%%%%%
%save the file
audiowrite("carry.wav",x1+x2,fsel);
audiowrite("modulate.wav",Ym,fsel);
audiowrite("demodulate.wav",Yd,fsel);
audiowrite("output"+outname+".wav",Yf,fsel);
%%%%%%%%%%%%%%%%%
```

```
function [Af] =
low_filter(A,f,input_num,low_pass_filter)
%LOW_FILTER Summary of this function goes here
%   Detailed explanation goes here
Af=filter(low_pass_filter,A);        %filter the
audio


figure();        %opens a figure
N=length(A);    %get the amplitude length
k=0:N-1;         %calculating k
AF=fft(Af,N);    %frequency shift the orignal
AI=fft(A,N);    %frequency shift the filter
subplot(1,2,1); %divide it into half and choose the
first half
plot(k,abs(AI));%draw the orignal audio
xlabel("K",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" before filter
Wave Graph",'FontSize',18);
subplot(1,2,2); %second half
plot(k,abs(AF));%draw the filter audio
xlabel("K",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" after filter
Wave Graph",'FontSize',18);
pause(3);        %wait 3 seconds

figure();            %opens a figure
F=(0:N-1)*f/N;      %calculate the frequency to plot
it
subplot(1,2,1);      %divide it into half and choose
the first half
plot(F,abs(A)/N);  %draw the orignal audio
xlabel("frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" before filter in
frequncy domain",'FontSize',18);
```

```
subplot(1,2,2);      %second half
plot(F,abs(Af)/N);   %draw the filter audio
xlabel("frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" after filter in
frequncy domain",'FontSize',18);
pause(3);            %wait 3 seconds


figure();                        %opens a figure
subplot(1,2,1);                  %divide it into half
and choose the first half
plot(F,abs(fftshift(A))/N);%draw the orignal audio
xlabel("frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" before filter
shifted frequency",'FontSize',18);
subplot(1,2,2);                  %second half
plot(F,abs(fftshift(Af))/N);%draw the filter audio
xlabel("frequency",'FontSize',10);
ylabel("Amplitude",'FontSize',10);
title("Input"+int2str(input_num)+" before filter
shifted frequency",'FontSize',18);


end

function record_myvoice(Fs,Nseconds,input_num)
%RECORD Summary of this function goes here
%   Detailed explanation goes here
warning off
ch=2;            %Number of channels--2 options--1
(mono) or 2 (stereo)
datatype='uint8';
nbits=16;        %8,16,or 24

                 % to record audio data from an
input device ...
...such as a microphone for processing in MATLAB
recorder=audiorecorder(Fs,nbits,ch);
disp('Start speaking..')
```

```
%Record audio to audiorecorder object,...
...hold control until recording completes
recordblocking(recorder,Nseconds);
disp('End of Recording.');
%Store recorded audio signal in numeric array
x=getaudiodata(recorder);
%Write audio file
audiowrite("input"+int2str(input_num)+".wav",x,Fs);
end
```