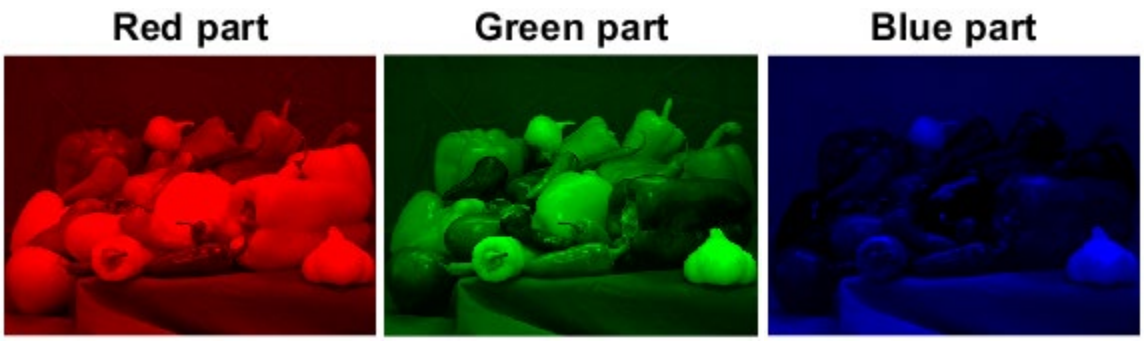
Name	Sec	BN
Youssef Samir Zidan	4	9220988
Yassin Essam Yassin	4	9220965

Part 1

A.

```
im = imread('peppers.png');
redpart = im (:,:,1);
greenpart = im (:,:,2);
bluepart = im (:,:,3);
      rgbImage = cat(3, redpart, greenpart, bluepart);
imshow(rgbImage);
gray=rgb2gray(image);
imshow(gray);


redOnlyImage = cat(3, redpart, zeros(size(greenpart)), zeros(size(bluepart)));
greenOnlyImage = cat(3, zeros(size(redpart)), greenpart, zeros(size(bluepart)));
blueOnlyImage = cat(3, zeros(size(redpart)), zeros(size(greenpart)), bluepart);


figure;
subplot(1, 3, 1), imshow(redOnlyImage), title('Red part');
subplot(1, 3, 2), imshow(greenOnlyImage), title('Green part');
subplot(1, 3, 3), imshow(blueOnlyImage), title('Blue part');
```



B

1.

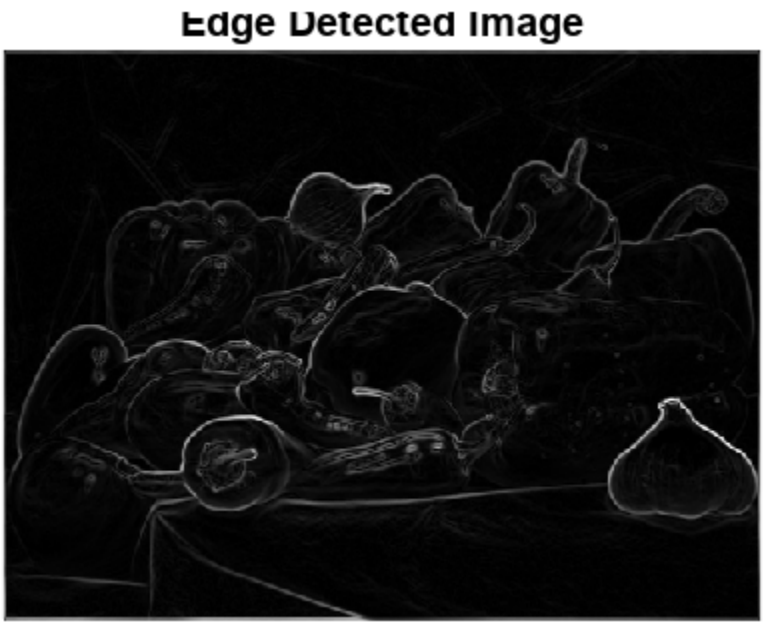
Edge detection kernal

```
Ex = [-1 0 1; -2 0 2; -1 0 1]; % Horizontal edge detection
Ey = [-1 -2 -1; 0 0 0; 1 2 1]; % Vertical edge detection
```

Why we choose it ?

we choose edge detection kernel as it has Ex which gets horizontal edges and Ey which gets vertical edges in both Ex and Ey the central number in central row is 0 and the summation of total numbers equal 0 , to compare the central pixel with neighbours pixels , if there is no change or small change it will have number close to zero and it will be black , if there is fast change in intensity the pixel will have a value and edge will be detected

Results:




```
%  Edge detection
% Define Sobel kernels
Ex = [-1 0 1; -2 0 2; -1 0 1]; % Horizontal edge detection
Ey = [-1 -2 -1; 0 0 0; 1 2 1]; % Vertical edge detection


% Apply Sobel kernels using conv2
edge_x = conv2(double(gray), Gx);
edge_y = conv2(double(gray), Gy);
```

```
% Combine the horizontal and vertical edges
edge_img = sqrt(edge_x.^2 + edge_y.^2);


figure, imshow(mat2gray(edge_img)), title('Edge Detected Image');
```

2. Image sharpening

Image sharpening kernal

```
sharpenKernel = [0 -10 0; -10 10000 -10; 0 -10 0];
```

why we choose it ?

we choose the sharpening kernel as we detected the edges berofe now central pixels multiplied by 10000 and surrounding pixels are multiplied by -10 so it will be sharped because of high frequency components in the edges and the pixels at edges will have higher contrast making the edge appears sharp


```
%  Image sharpening
sharpenKernel = [0 -10 0; -10 10000 -10; 0 -10 0];
sharpenedImage1 = conv2((redpart), sharpenKernel, 'same');
sharpenedImage2 = conv2((greenpart), sharpenKernel, 'same');
sharpenedImage3 = conv2((bluepart), sharpenKernel, 'same');
      sharpenedImagecat = cat(3, sharpenedImage1, sharpenedImage2,
sharpenedImage3);
figure, imshow(mat2gray(sharpenedImagecat)), title('Image Sharpening');
```

3. Blurring (averaging)

Kernal used

```
blurKernel = ones(5, 5) / 25;
```

```
blurKernel = 5×5
    0.0400    0.0400    0.0400    0.0400    0.0400
    0.0400    0.0400    0.0400    0.0400    0.0400
    0.0400    0.0400    0.0400    0.0400    0.0400
    0.0400    0.0400    0.0400    0.0400    0.0400
    0.0400    0.0400    0.0400    0.0400    0.0400
```

Why we choose it ?

we choose the blurring kernel as it of size 5*5 and all value are 1/25 to create average , so central pixel value is the average value of it and the 24 surrounding pixels , also by taking the average the high frequency details are reduced so by increasing the size of kernel more than 5*5 more averaging and blurring but it cause loss of details in image

Results :


```
%  Blurring (averaging)
blurKernel = ones(5, 5) / 25;
blurredImage1 = conv2(redpart, blurKernel, 'same');
blurredImage2 = conv2(greenpart, blurKernel, 'same');
blurredImage3 = conv2(bluepart, blurKernel, 'same');
      blurredImagecat = cat(3, blurredImage1, blurredImage2, blurredImage3);
```

```
figure, imshow(mat2gray(blurredImagecat)), title('Blurring (Averaging)');
```

4. Motion blurring

Kernal used

```
motionBlurKernel = zeros(1, 20);
motionBlurKernel(:) = 1 / 20;
```

```
motionBlurKernel = 1×20
    0.0500    0.0500    0.0500    0.0500    0.0500 ···
```

Why we choose it ?

we choose motion blurring kernel also depends on average but average in the horizontal direction as it has length of 20 and all elements are 1/20 to create average on length of 20 , also it depends in length of kernel so longer length will result in more motion blurring

Results :


```
%  Motion blurring
motionBlurKernel = zeros(1, 20);
motionBlurKernel(:) = 1 / 20;
rdash=conv2(redpart, motionBlurKernel);
gdash=conv2(greenpart, motionBlurKernel);
```

```
bdash=conv2(bluepart, motionBlurKernel );
      MblurredImagecat = cat(3, rdash, gdash, bdash);
figure, imshow(mat2gray(MblurredImagecat)), title('Motion Blurring');
```

C


```
% Call the restoreimage function to restore the motion-blurred image
restored_red = restoreimage(rdash, motionBlurKernel);
restored_green = restoreimage(gdash, motionBlurKernel);
restored_blue = restoreimage(bdash, motionBlurKernel);
% Crop the restored images to match the original dimensions
[Oheight, Owidth]=size(redpart);
restored_red = restored_red(1:Oheight, 1:Owidth);
restored_green = restored_green(1:Oheight, 1:Owidth);
restored_blue = restored_blue(1:Oheight, 1:Owidth);
```

```
      Restored = cat(3, restored_red, restored_green, restored_blue);

figure, imshow(mat2gray(Restored)), title('Restored');
```

```
function restored_img = restoreimage(image, motionBlurKernel)
    % Compute the Fourier Transform of the blurred image
    F_blurred = fft2(image);


    % Padding the kernel to the size of the image
    kernel_padded = zeros(size(image, 1), size(image, 2));
    kernel_padded(1:size(motionBlurKernel, 1), 1:size(motionBlurKernel, 2)) =
motionBlurKernel;
    % FT of the kernel
    F_kernel = fft2(kernel_padded);


    % Inverse filter
    inverse_filter = 1 ./ F_kernel;


    % Set a threshold to avoid division by very small values
    threshold = 1e-20;
    inverse_filter(abs(F_kernel) < threshold) = 0;


    F_restored = F_blurred .* inverse_filter;
    restored_img = ifft2(F_restored);
   end
```

Full code

```
im = imread('peppers.png');
redpart = im(:,:,1);
greenpart = im(:,:,2);
bluepart = im(:,:,3);
      rgbImage = cat(3, redpart, greenpart, bluepart);
imshow(rgbImage);
gray=rgb2gray(im);
imshow(gray);
```

```
redOnlyImage = cat(3, redpart, zeros(size(greenpart)), zeros(size(bluepart)));
greenOnlyImage = cat(3, zeros(size(redpart)), greenpart, zeros(size(bluepart)));
blueOnlyImage = cat(3, zeros(size(redpart)), zeros(size(greenpart)), bluepart);


figure;
subplot(1, 3, 1), imshow(redOnlyImage), title('Red part');
subplot(1, 3, 2), imshow(greenOnlyImage), title('Green part');
subplot(1, 3, 3), imshow(blueOnlyImage), title('Blue part');




%  Edge detection
Ex = [-1 0 1; -2 0 2; -1 0 1]; % Horizontal edge detection
Ey = [-1 -2 -1; 0 0 0; 1 2 1]; % Vertical edge detection


% Apply Sobel kernels using conv2
edge_x = conv2(double(gray), Ex);
edge_y = conv2(double(gray), Ey);


% Combine the horizontal and vertical edges
edge_img = sqrt(edge_x.^2 + edge_y.^2);


figure, imshow(mat2gray(edge_img)), title('Edge Detected Image');


%  Image sharpening
sharpenKernel = [0 -10 0; -10 10000 -10; 0 -10 0];
sharpenedImage1 = conv2((redpart), sharpenKernel, 'same');
sharpenedImage2 = conv2((greenpart), sharpenKernel, 'same');
sharpenedImage3 = conv2((bluepart), sharpenKernel, 'same');
      sharpenedImagecat = cat(3, sharpenedImage1, sharpenedImage2,
sharpenedImage3);
figure, imshow(mat2gray(sharpenedImagecat)), title('Image Sharpening');


%  Blurring (averaging)
blurKernel = ones(5, 5) / 25;
blurredImage1 = conv2(redpart, blurKernel, 'same');
blurredImage2 = conv2(greenpart, blurKernel, 'same');
blurredImage3 = conv2(bluepart, blurKernel, 'same');
      blurredImagecat = cat(3, blurredImage1, blurredImage2, blurredImage3);
figure, imshow(mat2gray(blurredImagecat)), title('Blurring (Averaging)');
```

```
%  Motion blurring
motionBlurKernel = zeros(1, 20);
motionBlurKernel(:) = 1 / 20;
rdash=conv2(redpart, motionBlurKernel);
gdash=conv2(greenpart, motionBlurKernel);
bdash=conv2(bluepart, motionBlurKernel );
      MblurredImagecat = cat(3, rdash, gdash, bdash);
figure, imshow(mat2gray(MblurredImagecat)), title('Motion Blurring');


% Call the restoreimage function to restore the motion-blurred image
restored_red = restoreimage(rdash, motionBlurKernel);
restored_green = restoreimage(gdash, motionBlurKernel);
restored_blue = restoreimage(bdash, motionBlurKernel);
% Crop the restored images to match the original dimensions
[Oheight, Owidth]=size(redpart);
restored_red = restored_red(1:Oheight, 1:Owidth);
restored_green = restored_green(1:Oheight, 1:Owidth);
restored_blue = restored_blue(1:Oheight, 1:Owidth);


      Restored = cat(3, restored_red, restored_green, restored_blue);

figure, imshow(mat2gray(Restored)), title('Restored');
```

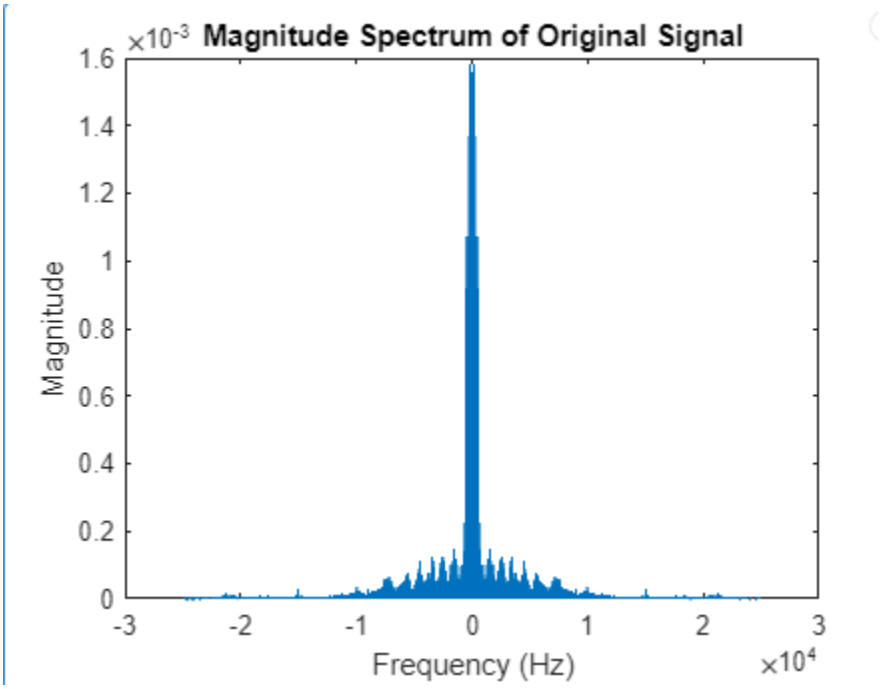
```
function restored_img = restoreimage(image, motionBlurKernel)
    % Compute the Fourier Transform of the blurred image
    F_blurred = fft2(image);


    % Padding the kernel to the size of the image
    kernel_padded = zeros(size(image, 1), size(image, 2));
    kernel_padded(1:size(motionBlurKernel, 1), 1:size(motionBlurKernel, 2)) =
motionBlurKernel;
    % FT of the kernel
    F_kernel = fft2(kernel_padded);


    % Inverse filter
    inverse_filter = 1 ./ F_kernel;


    % Set a threshold to avoid division by very small values
    threshold = 1e-20;
```

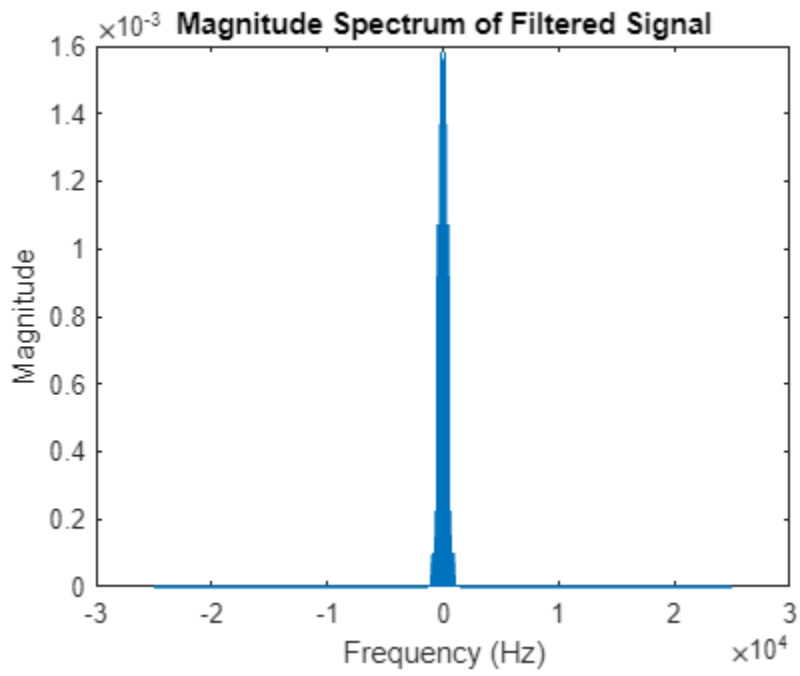
```
    inverse_filter(abs(F_kernel) < threshold) = 0;


    F_restored = F_blurred .* inverse_filter;
    restored_img = ifft2(F_restored);
   end
```

Part 2

A-As we discussed in lecture the sampling frequency is how man samples we take per second and it must be greater than 2fm and for digital audio fm can be up to around 20khz but fs must be oversampled to avoid overlapping so fs is around 44.khz - 48 khz as in lecture if we increase more the sampling frequency the audio will be more finer and clear so we increased it to 50khz that will very fine and clear , for the bit depth it its how many bits taken per sample and it affects the bit rate which equal to number of bit depth multiplied by fs , also higher bit depth leads to higher resolution audio and more clear , so we choose 16 bits

b.




Magnitude Spectrum of Filtered Signal
×10-3
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Magnitude
-3
-2
-1
0
1
2
3
Frequency (Hz)
×10^4

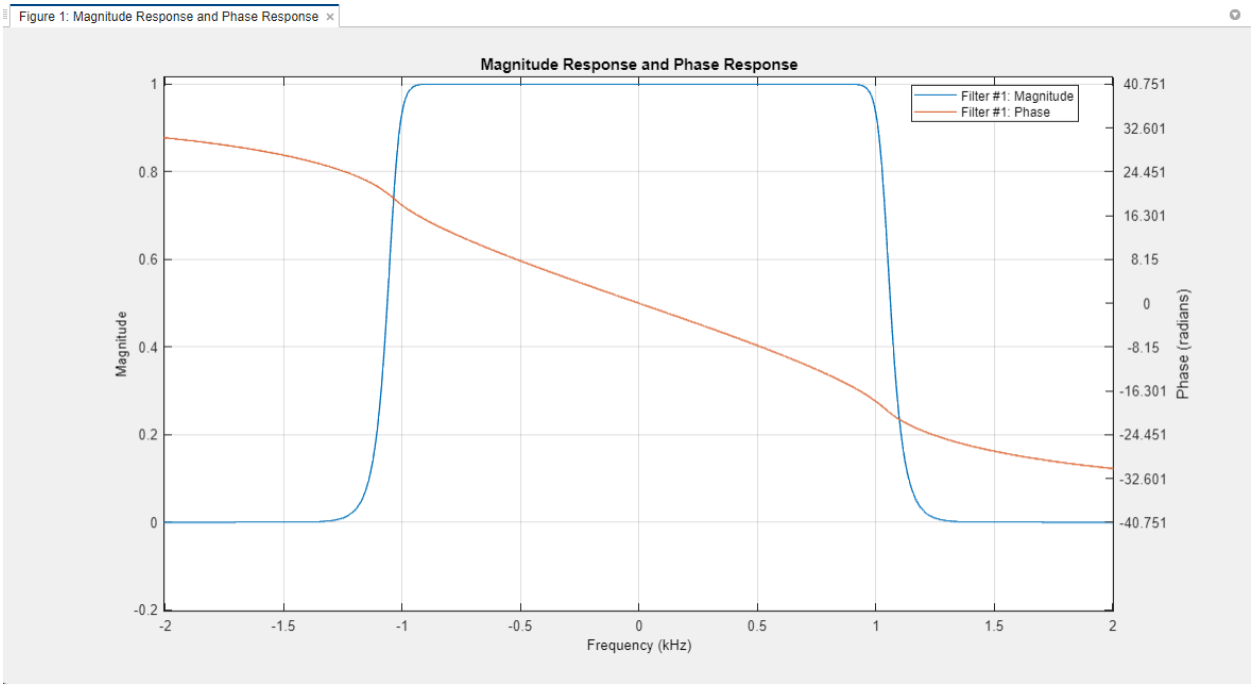
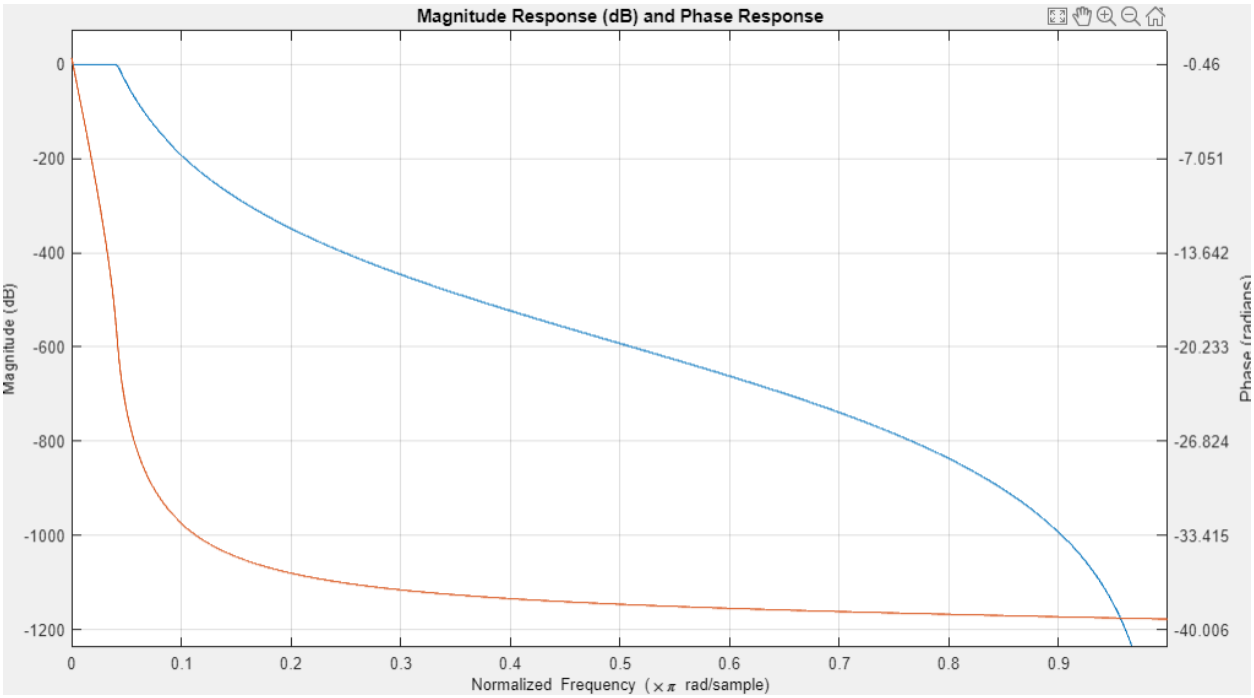
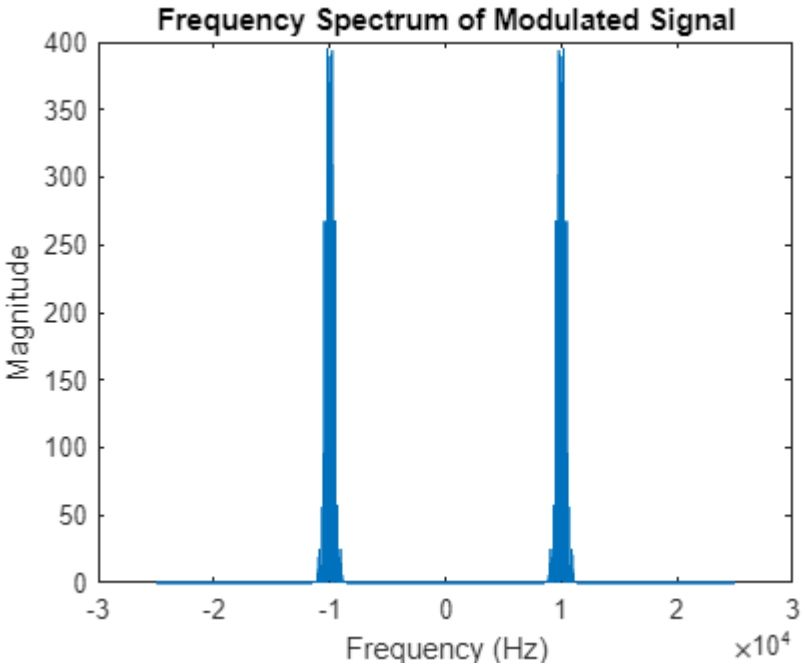
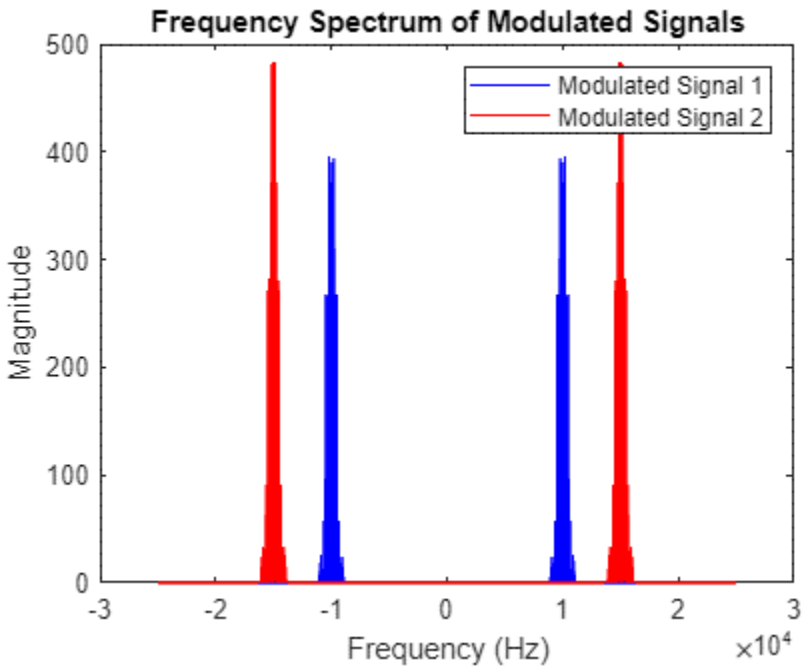


Figure 1: Magnitude Response and Phase Response
Magnitude Response and Phase Response
Filter #1: Magnitude
Filter #1: Phase
Magnitude
Phase (radians)
Frequency (kHz)
1
0.8
0.6
0.4
0.2
0
-0.2
40.751
32.601
24.451
16.301
8.15
0
-8.15
-16.301
-24.451
-32.601
-40.751
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

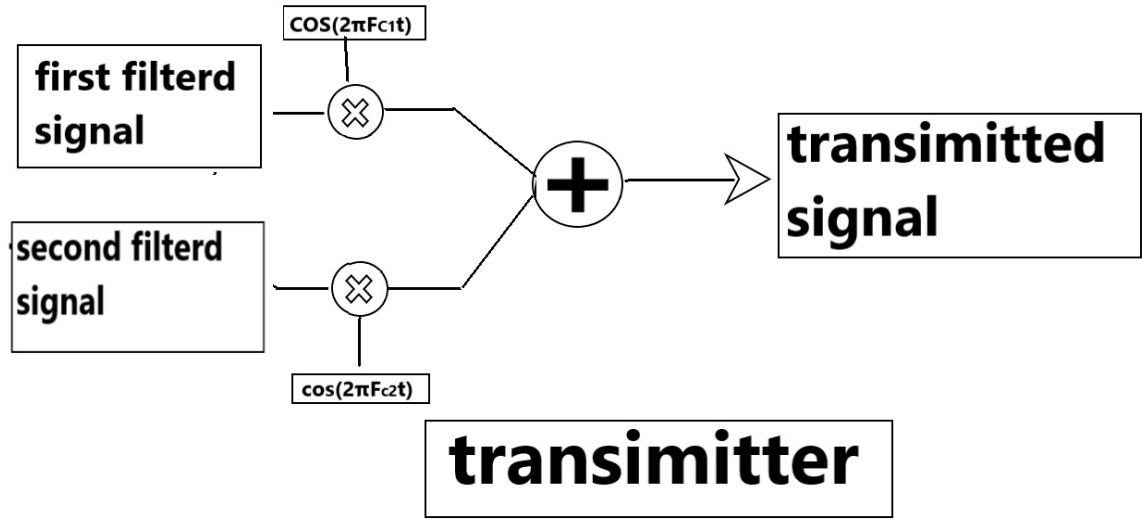

Magnitude Response (dB) and Phase Response
Magnitude (dB)
Phase (radians)
Normalized Frequency (×π rad/sample)
0
-200
-400
-600
-800
-1000
-1200
-0.46
-7.051
-13.642
-20.233
-26.824
-33.415
-40.006
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

d

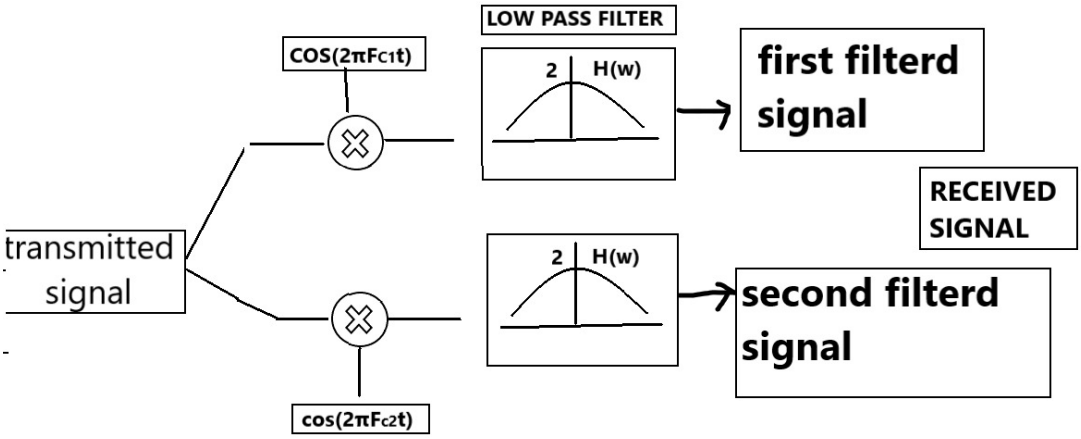

Frequency Spectrum of Modulated Signal
Magnitude
Frequency (Hz)
×10^4
400
350
300
250
200
150
100
50
0
-3
-2
-1
0
1
2
3



d-To avoid overlapping between the signal and itself fc-fpass>fpass-fc so for the first and second signal each must have frequency greater than fpass , fc1>fpass and fc2>fpass . and to avoid overlapping between the first signal and the second signal fc2-fpass>fc1+fpass so wich will be fc2-fc1>2fpass which means the diffrence between the first and the second frequency carrier must be greater than double of the lpf fpass



transimitter

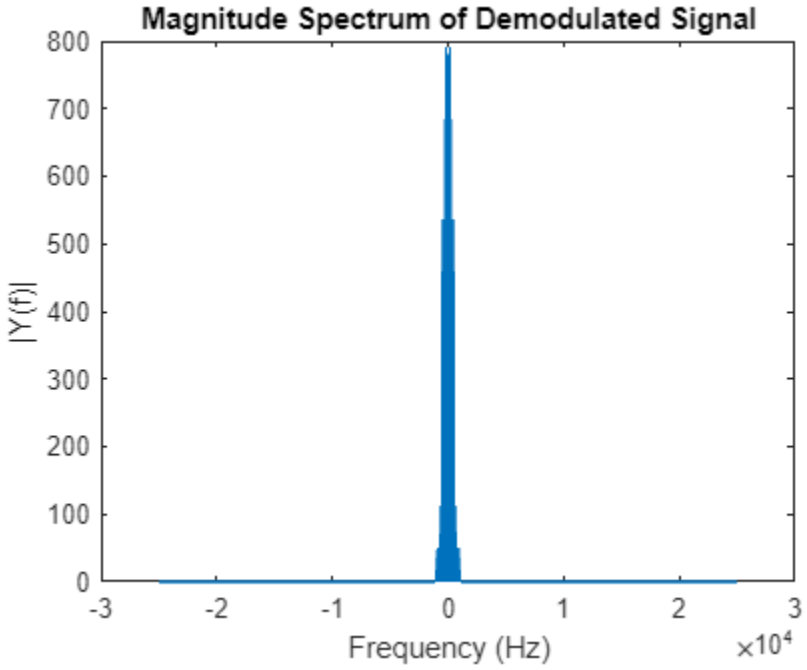


RECEIVER

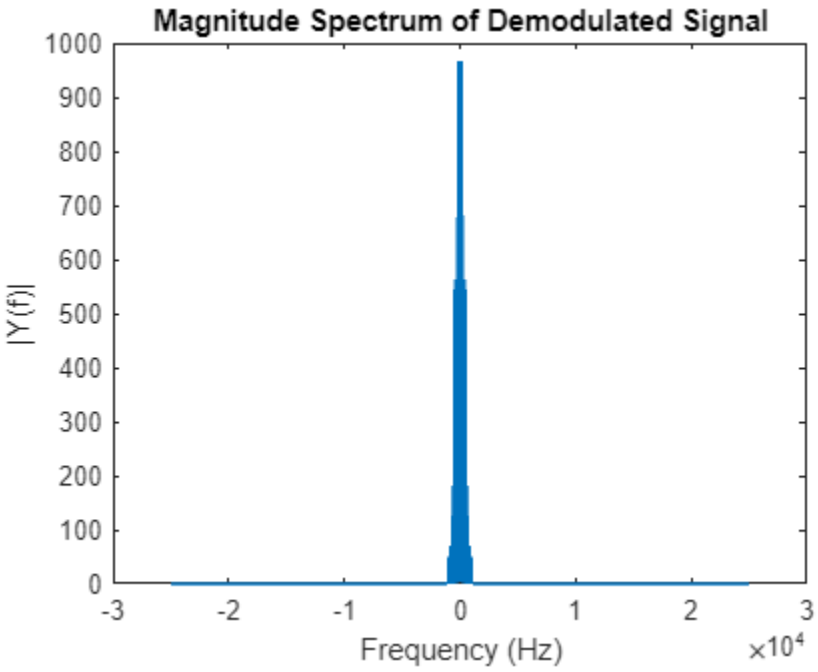
first equation is w(t)=transmitted signal(in time domain) x cos(2πft) , in time domain it is multiplication of transmitted signal by cos(2πft) the frequency of the cosine(carrier) must be the frequency of signal i want to demodulate , this equation in frequency domain will be convolution of transmitted signal(in frequency domain) with two impulses located at +fc and -fc then the modulated signal in transmitter in frequency domain i want to demodulate will be shifted to the zero frequency and with its original amplitude halved , and to 2fc and -2fc with amplitude divided by 4 .

second equation is w(t) convoluted with h(t) , in time domain it is convolution of the signal with h(t) , in frequency domain it is multiplication of signal with H(W) , where H(w) is non ideal low pass filter and multiplied with gain of 2 , that will pass the signal i want to demodulate which centred at zero frequency and cut the remaining bands of signals out of the bandwidth , also the other filtered signal will not overlap with signal i want to demodulate as the difference between the carrier frequency was big enough to not overlap

Input1.wav Demodulated



Input2.wav demodulated



```
[x1, fs1] = audioread('input1.wav');
[x2, fs2] = audioread('input2.wav');



% plotting for the first original signal
N1 = length(x1);
X1 = fft(x1, N1);
f1 = (-N1/2:N1/2-1)*fs1/N1;
figure;
plot(f1, abs(fftshift(X1))/N1);
title('Magnitude Spectrum of Original Signal 1');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


% Load the filter
load('lpf3.mat');


% Apply the loaded filter
filtered_voice1 = filter(lpf3, x1);
filtered_voice2 = filter(lpf3, x2);


% plotting for the filtered signal 1
N1_filtered = length(filtered_voice1);
X_filtered1 = fft(filtered_voice1, N1_filtered);
f_filtered1 = (-N1_filtered/2:N1_filtered/2-1)*fs1/N1_filtered;
figure;
plot(f_filtered1, abs(fftshift(X_filtered1))/N1_filtered);
title('Magnitude Spectrum of Filtered Signal 1');
xlabel('Frequency (Hz)');
ylabel('Magnitude');



% Modulation parameters
Fc_mod1 = 10000; % 10 kHz for first signal
Fc_mod2 = 15000; % 15 kHz for second signal


% Time vectors for modulation
t1 = (0:length(filtered_voice1)-1)/fs1;
t2 = (0:length(filtered_voice2)-1)/fs1;
```

```
% Modulate both signals (Sender
modsig1 = filtered_voice1 .* cos(2 * pi * Fc_mod1 * t1)';
modsig2 = filtered_voice2 .* cos(2 * pi * Fc_mod2 * t2)';
modulated_signal = modsig1+ modsig2;

% FFT for modulated signals
fft_modulated1 = fft(modsig1);
fft_modulated2 = fft(modsig2);


% FFT for the first modulated signal
N = length(modulated_signal1);
X_modulated1 = fft(modsig1, N);
f_modulated1 = (-N/2:N/2-1)*fs1/N;


% FFT for the second modulated signal
X_modulated2 = fft(modsig2, N);
f_modulated2 = (-N/2:N/2-1)*fs2/N;


% Plotting both spectrums on the same graph
figure;
plot(f_modulated1, abs(fftshift(X_modulated1))/N, 'b');  % First signal in blue
hold on;
plot(f_modulated2, abs(fftshift(X_modulated2))/N, 'r');  % Second signal in red
hold off;


% Adding titles and labels
title('Magnitude Spectrum of Both Modulated Signals');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
legend('Modulated Signal 1', 'Modulated Signal 2');


x = 1;
%mag spec of demodulated signals
N1_demodulated = length(demodulated_filtered_signal);
f_demodulated1 = (-N1_demodulated/2:N1_demodulated/2-1)*fs1/N1_demodulated;
figure;
plot(f_demodulated1, abs(fftshift(fft(demodulated_filtered_signal,
N1_demodulated)))/N1_demodulated);
title('Magnitude Spectrum of demodulated Signal 1');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
N2_demodulated = length(demodulated_filtered_signal);
f_demodulated2 = (-N2_demodulated/2:N2_demodulated/2-1)*fs2/N2_demodulated;
```

```
figure;
plot(f_demodulated2, abs(fftshift(fft(demodulated_filtered_signal,
N2_demodulated)))/N2_demodulated);
title('Magnitude Spectrum of demodulated Signal 2');
xlabel('Frequency (Hz)');
ylabel('Magnitude');


% Demodulate (Receiver)

while x ~= 0
    x = input('Choose signal to demodulate: 1 or 2 (0 to exit): ');


    switch x
        case 1
            Fc_mod = Fc_mod1;
            t = t1;
            modulated_signal = modulated_signal1;
        case 2
            Fc_mod = Fc_mod2;
            t = t2;
            modulated_signal = modulated_signal2;
        otherwise
            continue;
    end


    if x == 1 || x == 2
        demodulated_signal = modulated_signal .* cos(2 * pi * Fc_mod * t)';

        demodulated_filtered_signal = 2 * filter(lpf3, demodulated_signal);

        % Play the demodulated signal
        sound(demodulated_filtered_signal, fs1);
    end
end
```

Reference

1- Digital Image processing Gonalez 3rd edition

2- *Digital Signal processing Proakis 4th edition*

3- *Digital Image Processing, K. Pratt,*