**Signals**

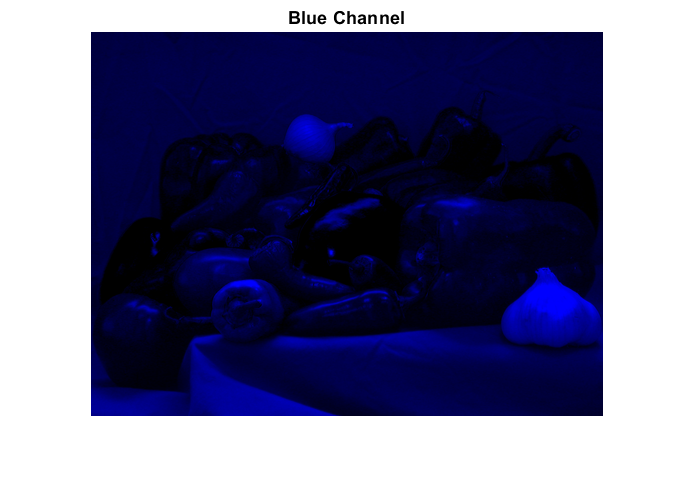
|  |  |  |
| --- | --- | --- |
| BN | **ID** | **Name** |
| 15 | 9220091 | أحمد محمد عرفات |
| 21 | 9220121 | أحمد وهبة شعبان |

1. **Image filtering and restoration**

**Task A :**

I used “imread” function to read the image and then used then isolated every colour channel by setting the other channels to zero then used the “imshow” function to display the channels.





Task B:

Edge detection:

I used sobel kernel to do the edge detection, and then defined “sobelkernelx” and “sobelkernely” for horizontal and vertical edge detection, the kernels are 3x3 matrices, The Sobel operator is a convolution-based technique used for edge detection in image processing. It works by computing the gradient of the image intensity at each pixel, highlighting regions of rapid intensity change that correspond to edges.

Then I used the “rgb2gray” function to make the image in grayscale then converted it to double by using the function “double “, then applied the convolution on the grayscale image using the “conv2” function.

Then calculated the gradient magnitude by combining the horizontal and vertical gradients, and the displayed the edge detected image.

**Sharpening:**

First, I separated the colour channels from the original RGB image, and then converted the colour channels into double using the “double” function and then normalized them by dividing by 255.0.

Then I defined the sharpening kernel “which is a combination of a Laplacian operator and an identity matrix” as a 3x3 matrix, I used this kernel because it enhances edges and details in the image, and the Laplacian operator is known for highlighting regions of rapid intensity change, such as edges, while the identity matrix preserves the original pixel values. Combining these two components creates an effect that enhances edges and fine details.

Then I applied the convolution on each colour channel separately using the “conv2” function, and then combined the colour channels by using the “cat” function, and then adjusted the values in the sharpened image to ensure they fall within the valid range [0, 1] using the “min” and “max” functions.

Then I displayed the results in a figure with 2 subplots, one for the original image and the other for the sharpened image.



**Blurring:**

First, I separated the colour channels from the original RGB image, and then converted the colour channels into double using the “double” function and then normalized them by dividing by 255.0.

Then I defined the gaussian kernel based on a specified standard deviation and size, The Gaussian kernel achieves blurring through the process of convolution. When convolving an image with a Gaussian kernel, each pixel in the output image is computed as a weighted sum of its neighbouring pixels in the input image, with the weights determined by the Gaussian distribution. The convolution operation effectively smoothes the image by averaging the pixel values, giving more weight to the central pixel and decreasing weights as you move away from the centre.

Then I applied the convolution on each colour channel separately using the “conv2” function, and then combined the colour channels by using the “cat” function, and then adjusted the values in the sharpened image to ensure they fall within the valid range [0, 1] using the “min” and “max” functions.

Then I displayed the results in a figure with 2 subplots, one for the original image and the other for the blurred image.



**Motion blurring:**

First, I separated the colour channels from the original RGB image, and then converted the colour channels into double using the “double” function and then normalized them by dividing by 255.0.

then I defined the motion blur kernel as a horizontal row vector of ones, representing a uniform motion blur in the horizontal direction.

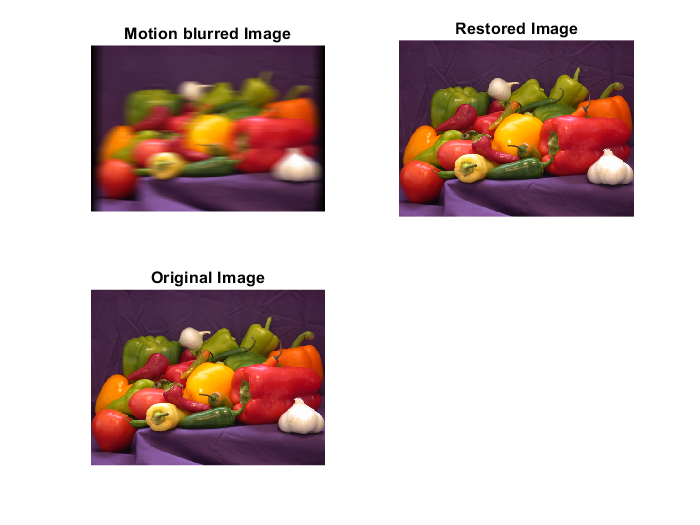
Then I applied the convolution on each colour channel separately using the “conv2” function, and then combined the colour channels by using the “cat” function, and then adjusted the values in the sharpened image to ensure they fall within the valid range [0, 1] using the “min” and “max” functions.

Then I displayed the results in a figure with 2 subplots, one for the original image and the other for the motion-blurred image.



**Task c:**

I computed the Fourier transform for the image and the kernel by using the “fft2” function, the I tried to restore the original image from the motion-blurred image in the frequency domain by dividing the Fourier transform of the motion-blurred image by the Fourier transform of the motion kernel, with the addition of a small constant to prevent division by zero, then I applied the inverse Fourier transform by using the “ifft2” function to obtain the restored image in the spatial domain.

The I displayed the results in a figure of 3 subplots, one for the original image and one for the blurred image and the last one for the restored image.

**The code:**

% task a:

% Reading the image

originalImage = imread('peppers.png');

grayImage = rgb2gray(originalImage);

figure;

imshow(originalImage);

title('Original');

% Red channel

redChannel = originalImage;

redChannel(:,:,2:3) = 0;

figure;

imshow(redChannel);

title('Red Channel');

% Blue channel

blueChannel = originalImage;

blueChannel(:,:,1:2) = 0;

figure;

imshow(blueChannel);

title('Blue Channel');

% Green channel

greenChannel = originalImage;

greenChannel(:,:,1) = 0;

greenChannel(:,:,3) = 0;

figure;

imshow(greenChannel);

title('Green Channel');

% taskk b:

% edge detection

% Sobel kernel for edge detection

sobelKernelX = [-1, 0, 1; -2, 0, 2; -1, 0, 1];

sobelKernelY = sobelKernelX';

% Convolution using conv2

edgesX = conv2(double(grayImage), sobelKernelX, 'same');

edgesY = conv2(double(grayImage), sobelKernelY, 'same');

% Calculate gradient magnitude

gradientMagnitude = sqrt(edgesX.^2 + edgesY.^2);

% Display the results

figure;

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

subplot(2, 2, 2); imshow(grayImage, []); title('Grayscale Image');

subplot(2, 2, 3); imshow(edgesX, []); title('Sobel X');

subplot(2, 2, 4); imshow(edgesY, []); title('Sobel Y');

figure;

imshow(gradientMagnitude, []); title('Edge Detected');

%sharpening

% Separate color channels

redChannel = originalImage(:, :, 1);

greenChannel = originalImage(:, :, 2);

blueChannel = originalImage(:, :, 3);

% Convert to double for convolution

redChannel = double(redChannel) / 255.0;

greenChannel = double(greenChannel) / 255.0;

blueChannel = double(blueChannel) / 255.0;

% Sharpening kernel (combination of Laplacian and identity)

sharpeningKernel = [0, -1, 0; -1, 5, -1; 0, -1, 0];

% Convolution using conv2 for each channel

sharpenedRed = conv2(redChannel, sharpeningKernel, 'same');

sharpenedGreen = conv2(greenChannel, sharpeningKernel, 'same');

sharpenedBlue = conv2(blueChannel, sharpeningKernel, 'same');

% Combine the sharpened channels

sharpenedImage = cat(3, sharpenedRed, sharpenedGreen, sharpenedBlue);

% Adjust the sharpened image to ensure values are in the valid range [0, 1]

sharpenedImage = min(max(sharpenedImage, 0), 1);

% Display the results

figure;

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

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

% blurring

% Separate color channels

redChannel = originalImage(:, :, 1);

greenChannel = originalImage(:, :, 2);

blueChannel = originalImage(:, :, 3);

% Convert to double for convolution

redChannel = double(redChannel) / 255.0;

greenChannel = double(greenChannel) / 255.0;

blueChannel = double(blueChannel) / 255.0;

% Define the Gaussian blur kernel

sigma = 1.5; % Standard deviation of the Gaussian kernel

size = 6; % Size of the kernel (should be even)

[X, Y] = meshgrid(-(size-1)/2:(size-1)/2, -(size-1)/2:(size-1)/2);

gaussianKernel = exp(-(X.^2 + Y.^2) / (2 \* sigma^2));

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

% Convolution using conv2 for each channel

blurredRed = conv2(redChannel, gaussianKernel, 'same');

blurredGreen = conv2(greenChannel, gaussianKernel, 'same');

blurredBlue = conv2(blueChannel, gaussianKernel, 'same');

% Combine the blurred channels

blurredImage = cat(3, blurredRed, blurredGreen, blurredBlue);

% Adjust the blurred image to ensure values are in the valid range [0, 1]

blurredImage = min(max(blurredImage, 0), 1);

% Display the results

figure;

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

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

% motion blurring

% Separate color channels

redChannel = originalImage(:, :, 1);

greenChannel = originalImage(:, :, 2);

blueChannel = originalImage(:, :, 3);

% Convert to double for convolution

redChannel = double(redChannel) / 255.0;

greenChannel = double(greenChannel) / 255.0;

blueChannel = double(blueChannel) / 255.0;

% Define the motion blur kernel

motionBlurLength = 15; % Length of the motion blur

motionBlurKernel = ones(1, motionBlurLength) / motionBlurLength;

% Convolution using conv2 for each channel

blurredRed = conv2(redChannel, motionBlurKernel);

blurredGreen = conv2(greenChannel, motionBlurKernel);

blurredBlue = conv2(blueChannel, motionBlurKernel);

% Combine the blurred channels

blurredImage = cat(3, blurredRed, blurredGreen, blurredBlue);

% Adjust the blurred image to ensure values are in the valid range [0, 1]

blurredImage = min(max(blurredImage, 0), 1);

% Display the results

figure;

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

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

% Task c:

original\_image=imread('peppers.png');

motion\_kernel= ones(1,30)/30;

motional\_image(:,:,1) = (conv2(original\_image(:,:,1),motion\_kernel));

motional\_image(:,:,2) = (conv2(original\_image(:,:,2),motion\_kernel));

motional\_image(:,:,3) = (conv2(original\_image(:,:,3),motion\_kernel));

motion\_kernel\_ft=fft2(motion\_kernel,size(motional\_image,1),size(motional\_image,2));

motional\_image\_ft=fft2(motional\_image);

epsilon=.00001;

image\_restored\_ft =motional\_image\_ft ./(motion\_kernel\_ft+epsilon);

restored\_image = abs(ifft2(image\_restored\_ft));

restored\_image =restored\_image(1:size(original\_image,1),1:size(original\_image,2),1:size(original\_image,3));

% Assuming 'original\_image', 'motional\_image', and 'restored\_image' are defined

figure;

% Subplot 1: Display motional\_image

subplot(2, 2, 1);

imshow(uint8(motional\_image));

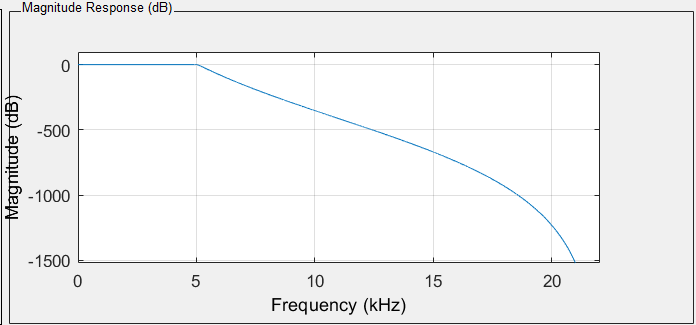
title('Motion blurred Image');

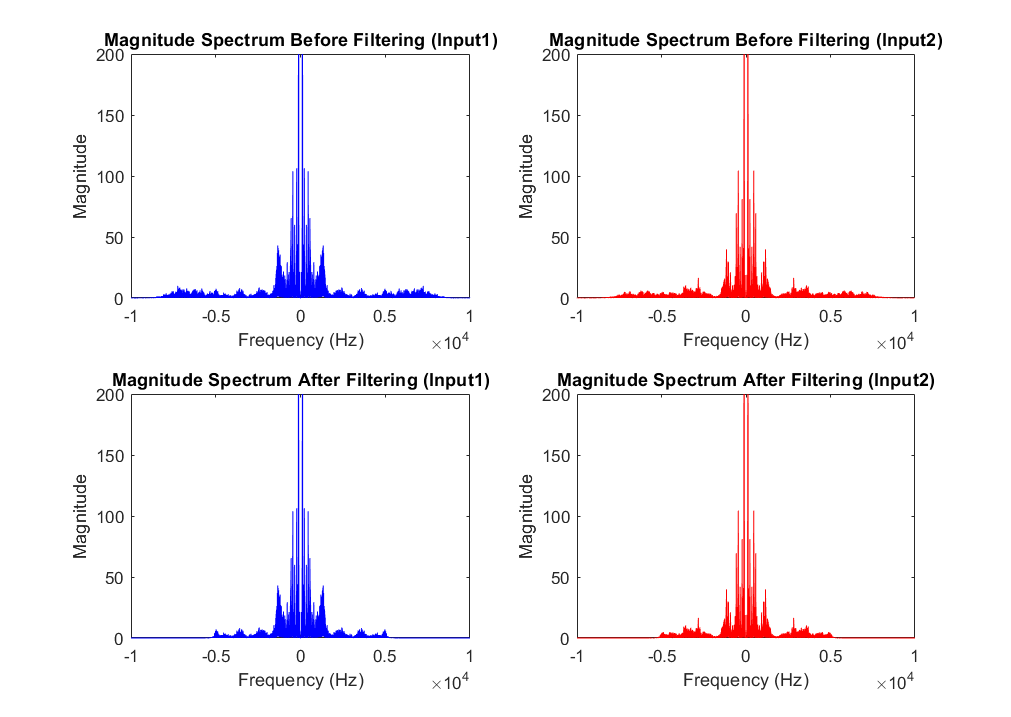
**II. Communication system simulation:**

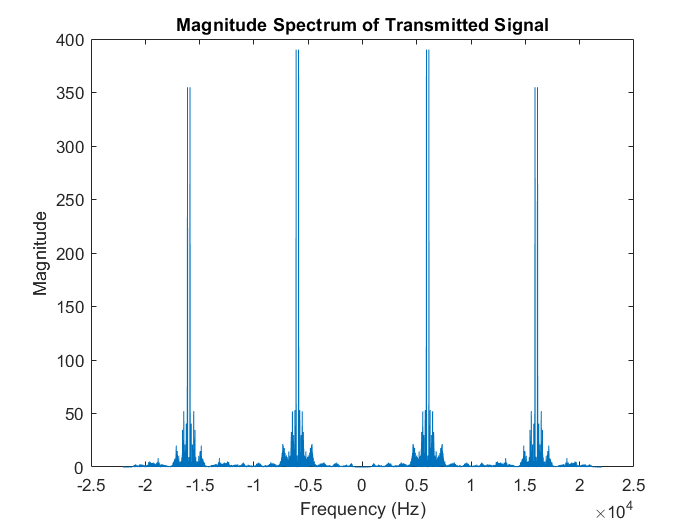
In task (A):

* (Sampling rate (frequency) = 44100) as the standard sampling rate for CDs. This frequency is more than double the highest frequency audible to the human ear (20 kHz)
* (bit Depth = 16) as with 16 bits, you can represent 2^16 (65,536) different amplitude levels, providing sufficient resolution for most audio applications. [1]

In task (B):

****I design LPF from command (filtrDesigner) (fpass =5000 Hz)

****In task (c):

****In task (d):

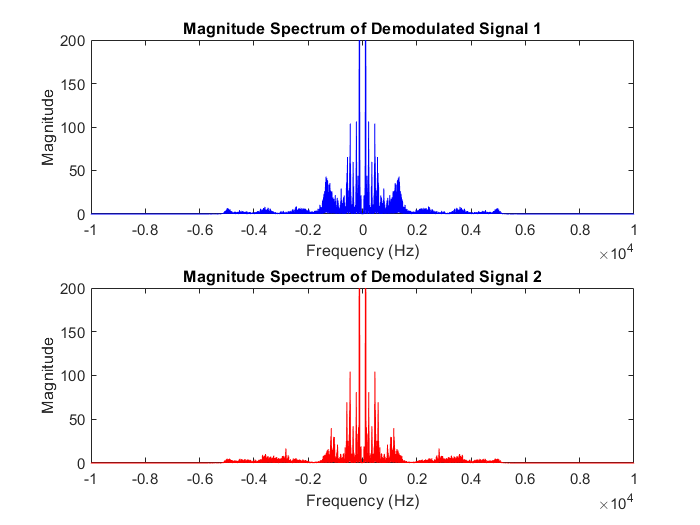
The choice of carrier

frequencies (6000 Hz and

16000 Hz) to avoid

interference.

In task (e):

Demodulation in Time Domain:

The demodulation is performed by multiplying the transmitted signal by the carrier signal. This operation removes the carrier and leaves the original message signal. To remove high-frequency components introduced during demodulation, a low-pass filter is applied.

Demodulation in Frequency Domain:

The demodulation is performed by

convolution

**Code:**

% Task a: Record voice

% Set the sampling frequency and bit depth

fs = 44100;

bitDepth = 16;

%Record the first voice segment

rec = audiorecorder(fs, bitDepth, 1);

disp('Start');

recordblocking(rec, 10);

input1 = getaudiodata(rec);

audiowrite('input1.wav', input1, fs);

disp('End');

%Record the second voice segment

disp('Start');

recordblocking(rec, 10);

input2 = getaudiodata(rec);

audiowrite('input2.wav', input2, fs);

disp('End');

%====================================================================%

%Task B: filterdesigner

input1 = audioread("D:\Ahmed\Projects\Signals\Done\2.A\input1.wav");

input2 = audioread("D:\Ahmed\Projects\Signals\Done\2.A\input2.wav");

% Apply the LPF to the signals

filteredInput1 = filter(f5000, input1);

filteredInput2 = filter(f5000, input2);

%Listen to filtered signals

%sound(filteredInput1, fs);

%sound(filteredInput2, fs);

%====================================================================%

% Task c: Plot magnitude spectrum before and after filtering

% Compute the FFT of the original and filtered signals

fftInput1 = fftshift(fft(input1));

fftInput2 = fftshift(fft(input2));

fftFilteredInput1 = fftshift(fft(filteredInput1));

fftFilteredInput2 = fftshift(fft(filteredInput2));

freq = linspace(-fs/2, fs/2, length(fftInput1));

% Plot magnitude spectrum

figure;

subplot(2, 2, 1);

plot(freq, abs(fftInput1), 'b');

title('Magnitude Spectrum Before Filtering (Input1)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(2, 2, 2);

plot(freq, abs(fftInput2), 'r');

title('Magnitude Spectrum Before Filtering (Input2)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(2, 2, 3);

plot(freq, abs(fftFilteredInput1), 'b');

title('Magnitude Spectrum After Filtering (Input1)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(2, 2, 4);

plot (freq, abs(fftFilteredInput2), 'r');

title('Magnitude Spectrum After Filtering (Input2)');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

%====================================================================%

% Task d: Frequency-division multiplexing (Amplitude Modulation)

Fc1 = 6000;

Fc2 = 16000;

% Modulate signals

modulatedSignal1 = filteredInput1 .\* cos(2 \* pi \* Fc1 \* (1/fs) \* (1:length(filteredInput1)).');

modulatedSignal2 = filteredInput2 .\* cos(2 \* pi \* Fc2 \* (1/fs) \* (1:length(filteredInput2)).');

transmittedSignal = modulatedSignal1 + modulatedSignal2;

% Plot the magnitude spectrum of the transmitted signal

fftTransmitted = fftshift(fft(transmittedSignal));

freqTransmitted = linspace(-fs/2, fs/2, length(fftTransmitted));

figure;

plot(freqTransmitted, abs(fftTransmitted));

title('Magnitude Spectrum of Transmitted Signal');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

%======================================================================%

%Task d:Demodulation

% Demodulate signals

demodulatedSignal1 = transmittedSignal .\* cos(2 \* pi \* Fc1 \* (1/fs) \* (1:length(transmittedSignal)).');

demodulatedSignal2 = transmittedSignal .\* cos(2 \* pi \* Fc2 \* (1/fs) \* (1:length(transmittedSignal)).');

% Use a low-pass filter to extract the original signals

demodulatedSignal1Filtered = 2\*filter(f5000, demodulatedSignal1);

demodulatedSignal2Filtered = 2\*filter(f5000, demodulatedSignal2);

% Save the demodulated signals

audiowrite('output1.wav', demodulatedSignal1Filtered, fs);

audiowrite('output2.wav', demodulatedSignal2Filtered, fs);

% Plot the magnitude spectrum of the demodulated signals

fftDemodulated1 = fftshift(fft(demodulatedSignal1Filtered));

fftDemodulated2 = fftshift(fft(demodulatedSignal2Filtered));

freqDemodulated = linspace(-fs/2, fs/2, length(fftDemodulated1));

figure;

subplot(2, 1, 1);

plot(freqDemodulated, abs(fftDemodulated1), 'b');

title('Magnitude Spectrum of Demodulated Signal 1');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

subplot(2, 1, 2);

plot(freqDemodulated, abs(fftDemodulated2), 'r');

title('Magnitude Spectrum of Demodulated Signal 2');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

%=========The End=========%

# References

|  |  |
| --- | --- |
| [1] | G. Brown, "Digital Audio Basics: Audio Sample Rate and Bit Depth," izotope, May 9, 2021. |
| [2] | Matlab, "Help," mathworks. |