Alexandria University

Faculty of Engineering

Communication and Electronics Department

Signals and Systems
Communication Systems Project

Section 4

Menna Reda Mohamed	**22011267**
Mona Yasser Awad	**22011272**

May 13, 2025

Contents

List of Figures ii

0.1 Introduction . . . 1
0.2 GUI layout . . . 1
0.3 GUI code . . . 1
  0.3.1 Transmitter . . . 1
  0.3.2 channel . . . 2
  0.3.3 Noise . . . 4
  0.3.4 Receiver(Filter) . . . 4
0.4 Results . . . 5
  0.4.1 Speech File Results . . . 5
  0.4.2 Music File Results . . . 9
0.5 Comparison . . . 11
0.6 MATLAB Code Implementation . . . 12

List of Figures

1 GUI Layout . . . 1
2 Original Speech Signal . . . 6
3 processed - Delta . . . 7
4 processed - 5000Hz . . . 7
5 processed - 1000Hz . . . 7
6 processed - Sinc function . . . 7
7 processed - Two delta . . . 7
8 Processed speech signal channels . . . 7
9 Noisy Speech Signal . . . 8
10 Filtered Speech Signal . . . 8
11 Original music Signal . . . 9
12 processed - Delta . . . 10
13 processed - 5000Hz . . . 10
14 processed - 1000Hz . . . 10
15 processed - Sinc function . . . 10
16 processed - Two delta . . . 10
17 Processed speech signal channels . . . 10
18 Noisy music Signal . . . 11
19 Filtered music Signal . . . 11

0.1 Introduction

This project aims to simulate a communication system that includes signal transmission through different types of channels. The original audio signal is first processed through one of several predefined channels, such as delta, exponential decay, sinc, or custom. Noise is then added to the channel output to simulate real-world interference. An ideal low-pass filter is finally applied at the receiver to retrieve the original signal. The project is implemented using MATLAB GUI to allow interactive selection of the channel type and noise level, and to visualize the signal in both time and frequency domains.

0.2 GUI layout

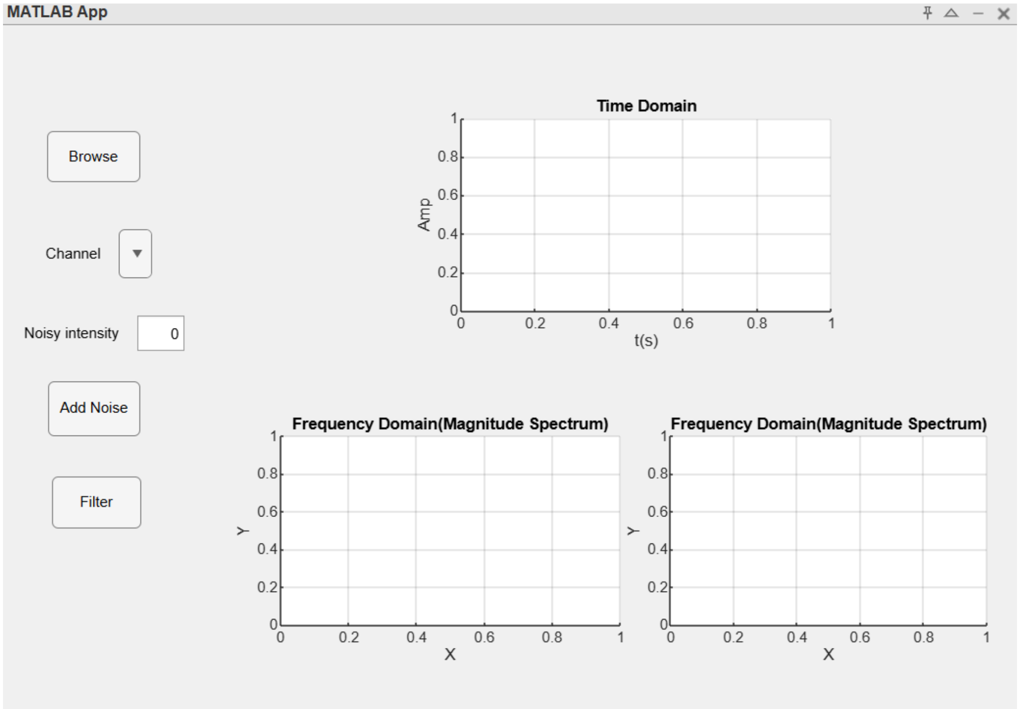

Figure 1: GUI Layout

0.3 GUI code

0.3.1 Transmitter

The Browse button allows the user to select an audio file from the computer. Once selected, the file is read and converted to mono to simplify processing. The audio is then played, and its waveform is displayed in the time domain. The signal is also transformed into the frequency domain, where both the magnitude and phase are plotted. A confirmation message appears to let the user know the file was loaded successfully.

```
% Button pushed function: BrowseButton
    function BrowseButtonPushed(app, event)
        % Transmitter
     [file, path] = uigetfile({'*.wav;*.mp3'}, 'Select Audio File');
     if isequal(file, 0)
```

```
            return;
        end
        [app.y, app.Fs] = audioread(fullfile(path, file));
        app.y = mean(app.y, 2);

        % Play audio
        sound(app.y, app.Fs);
         % Time domain plot
        duration = length(app.y) / app.Fs;
        app.duration = duration;
        t = linspace(0, duration, length(app.y));
        cla(app.UIAxes);
        plot(app.UIAxes, t, app.y, 'k');
        title(app.UIAxes, 'Original Signal in Time Domain');
        xlabel(app.UIAxes, 'Time (s)');
        ylabel(app.UIAxes, 'Amplitude');

         % Frequency domain plots
        N = length(app.y);
        f = linspace(-app.Fs/2, app.Fs/2, N);
        Y = fftshift(fft(app.y));
        Ymag = abs(Y);
        Yphase = angle(Y);

        % Magnitude plot
        cla(app.UIAxes2);
        plot(app.UIAxes2, f, Ymag, 'b');
        title(app.UIAxes2, 'Original Signal Frequency Magnitude');
        xlabel(app.UIAxes2, 'Frequency (Hz)');
        ylabel(app.UIAxes2, 'Magnitude');

        % Phase plot
        cla(app.UIAxes2_2);
        plot(app.UIAxes2_2, f, Yphase, 'r');
        title(app.UIAxes2_2, 'Original Signal Frequency Phase');
        xlabel(app.UIAxes2_2, 'Frequency (Hz)');
        ylabel(app.UIAxes2_2, 'Phase');

        uialert(app.UIFigure, 'Audio Loaded Successfully, Now select a
    channel', 'Loaded');
        end
```

0.3.2 channel

In the channel section, we use a drop-down menu that allows the user to select the type of channel to apply to the signal. The available options include different channel models such as delta, exponential decay, sinc function, and two deltas. Based on the selected option, the signal is processed accordingly to simulate the effects of the chosen channel type. This interactive selection helps in observing how different channel conditions impact the transmitted signal.

```
  % Value changed function: ChannelDropDown
        function ChannelDropDownValueChanged(app, event)
            if isempty(app.y)
                uialert(app.UIFigure, 'Load a file first', 'Error');
                return;
            end

            channel_options = app.ChannelDropDown.Items;
```

```
            selected_channel = app.ChannelDropDown.Value;
            channel_idx = find(strcmp(channel_options, selected_channel));

            data = app.y;
            fs = app.Fs;

            t = linspace(0, app.duration, length(data));

            % Generate impulse response
            switch channel_idx
                case 1
                    impulse_response = [1 zeros(1, length(data)-1)];
                case 2
                    impulse_response = exp(-2*pi*5000*t);
                case 3
                    impulse_response = exp(-2*pi*1000*t);
                case 4
                    x = 2*3000*t;
                    impulse_response = sinc(x)
                case 5
                    impulse_response = [2 zeros(1, fs-2) 0.5 zeros(1, length(data)-fs)];
            end

        % Convolve
        convolved_data = conv(data, impulse_response);

        % Store for playback or later
        app.y_channel = convolved_data;

        % Plot time domain
        t_conv = linspace(0, (length(convolved_data)-1)/fs, length(convolved_data));
        app.t_conv = t_conv;
        plot(app.UIAxes, t_conv, convolved_data);
        title(app.UIAxes, ['Processed - ', selected_channel]);
        xlabel(app.UIAxes, 'Time (s)');
        ylabel(app.UIAxes, 'Amplitude');

        % Plot freq domain (UIAxes3 assumed)
        N = length(convolved_data);
        f = linspace(-fs/2, fs/2, N);
        Y = fftshift(fft(convolved_data));
        Ymag = abs(Y);
        Yphase = angle(Y);
          % Plot magnitude spectrum on UIAxes2
        plot(app.UIAxes2, f, Ymag);
        title(app.UIAxes2, ['Magnitude - ', selected_channel]);
        xlabel(app.UIAxes2, 'Frequency (Hz)');
        ylabel(app.UIAxes2, 'Magnitude');

        % Plot phase spectrum on UIAxes3
        plot(app.UIAxes2_2, f, Yphase);
        title(app.UIAxes2_2, ['Phase - ', selected_channel]);
        xlabel(app.UIAxes2_2, 'Frequency (Hz)');
        ylabel(app.UIAxes2_2, 'Phase');
        uialert(app.UIFigure, 'Channel selected. Now add the noise.', 'Action Required');
        end
```

0.3.3 Noise

For the noise section, the user is prompted to input the noise intensity using a text field. This value represents the level of noise that will be added to the signal. After entering the desired noise intensity, the user can press the "Add Noise" button to apply the noise to the signal.

```
 % Noise
       function AddNoiseButtonPushed(app, event)
            if isempty(app.y_channel)
             uialert(app.UIFigure, 'Please select a channel first.', 'Missing Channel');
             return;
            end
           sigma = app.NoisyintensityEditField.Value;
           % Add noise
           Z = sigma * randn(1, length(app.y_channel));
           app.y_noisy = app.y_channel + Z;

           % Play audio
           player_noisy = audioplayer(app.y_noisy, app.Fs);
           play(player_noisy);
           pause(app.duration);
           stop(player_noisy);

           % Time-domain plot on UIAxes
           plot(app.UIAxes, app.t_conv, app.y_noisy);
           title(app.UIAxes, ['Noisy Signal (   = ', num2str(sigma), ') - Time Domain']);
           xlabel(app.UIAxes, 'Time (s)');

           % Frequency-domain plots on UIAxes2 and UIAxes3
           N = length(app.y_noisy);
           f = linspace(-app.Fs / 2, app.Fs / 2, N);
           Y = fftshift(fft(app.y_noisy));
           Ymag = abs(Y);
           Yphase = angle(Y);

           plot(app.UIAxes2, f, Ymag);
           title(app.UIAxes2, 'Noisy signal (Magnitude Spectrum)');
           xlabel(app.UIAxes2, 'Frequency (Hz)');

           plot(app.UIAxes2_2, f, Yphase);
           title(app.UIAxes2_2, 'Noisy signal (Phase Spectrum)');
           xlabel(app.UIAxes2_2, 'Frequency (Hz)');
       end

```

0.3.4 Receiver(Filter)

In the receiver section, a button labeled "Filter" is used to apply a low-pass filter to the noisy signal. The filter is designed to remove high-frequency noise and recover the original signal as much as possible. When the user presses the "Filter" button, the noisy signal undergoes filtering, and the result is displayed both in the time and frequency domains.

```
function FilterButtonPushed(app, event)
            % Apply ideal low-pass filter
           cutoff_freq = 3400; % Hz
           Y_noisy = fftshift(fft(app.y_noisy));
           f_filter = linspace(-app.Fs/2, app.Fs/2, length(Y_noisy));

```

```
            Y_filtered = Y_noisy;
            Y_filtered(abs(f_filter) > cutoff_freq) = 0;

            % Convert back to time domain
            app.y_filtered = real(ifft(ifftshift(Y_filtered)));

            % Play the filtered audio
            player_filtered = audioplayer(app.y_filtered, app.Fs);
            play(player_filtered);
            pause(app.duration);
            stop(player_filtered);

            % Plot time domain on app.UIAxes
            plot(app.UIAxes, app.t_conv, app.y_filtered);
            title(app.UIAxes, 'Filtered Signal - Time Domain');
            xlabel(app.UIAxes, 'Time (s)');

            % Plot magnitude spectrum on app.UIAxes2
            Ymag = abs(Y_filtered);
            plot(app.UIAxes2, f_filter, Ymag);
            title(app.UIAxes2, 'Filtered Signal - Magnitude Spectrum');
            xlabel(app.UIAxes2, 'Frequency (Hz)');

            % Plot phase spectrum on app.UIAxes3
            Yphase = angle(Y_filtered);
            plot(app.UIAxes2_2, f_filter, Yphase);
            title(app.UIAxes2_2, 'Filtered Signal - Phase Spectrum');
            xlabel(app.UIAxes2_2, 'Frequency (Hz)');
        end

```

0.4 Results

we present the results obtained from applying the communication system to two types of audio files: a speech file and a music file. Each file was processed through the communication system, including channel distortion, noise addition, and low-pass filtering.

0.4.1 Speech File Results

The figures below show the processed results for the speech file:

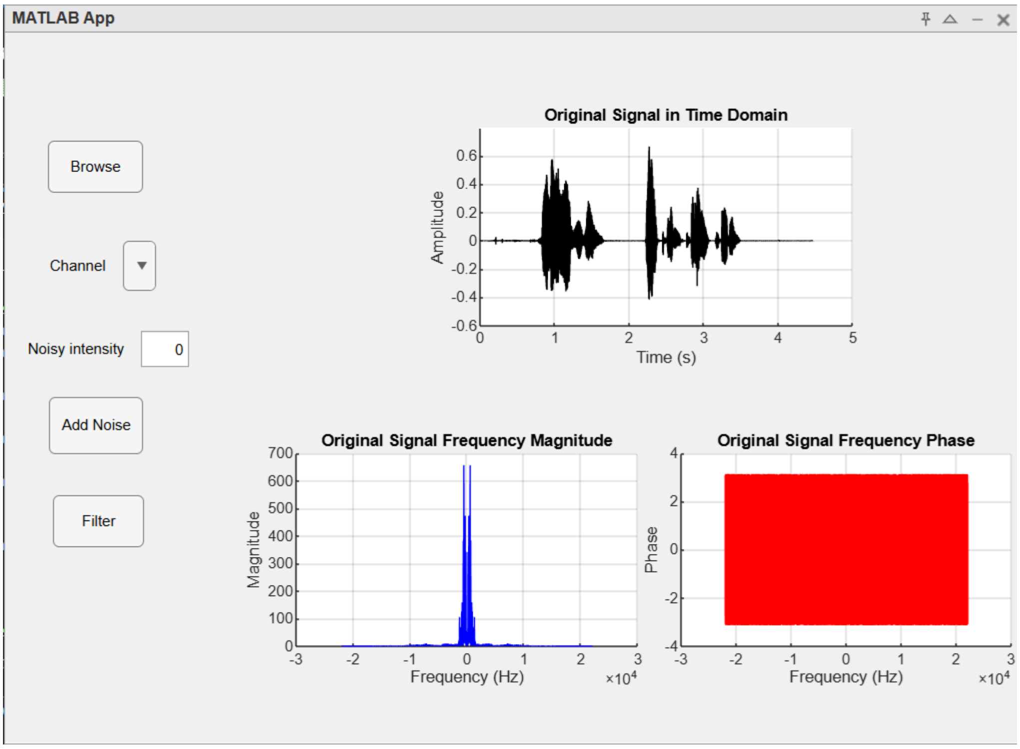

Figure 2: Original Speech Signal

The following figures show the effect of convolving the original speech signal with different channel impulse responses:

3: Delta function keeps the signal unchanged, acting as an identity filter.
4: 5000 Hz exponential decay slightly increases amplitude.
5: 1000 Hz exponential decay further increases amplitude.
6: Sinc function smooths the signal and removes high-frequency components.
7: Two delta pulses , adding a delayed version of the original signal.

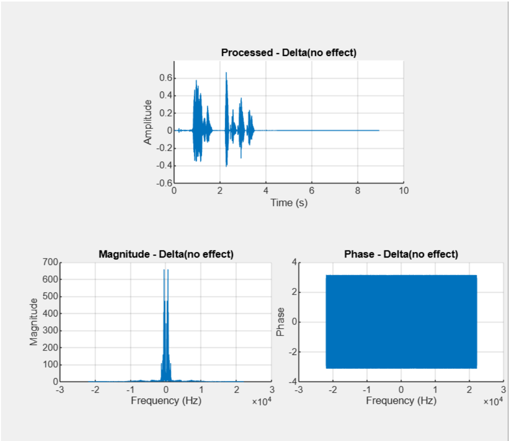

Figure 3: processed - Delta

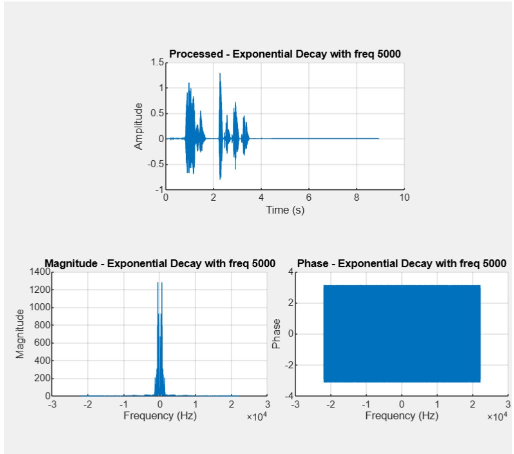

Figure 4: processed - 5000Hz

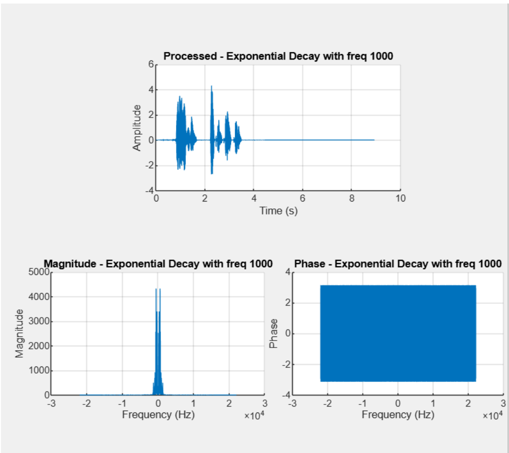

Figure 5: processed - 1000Hz

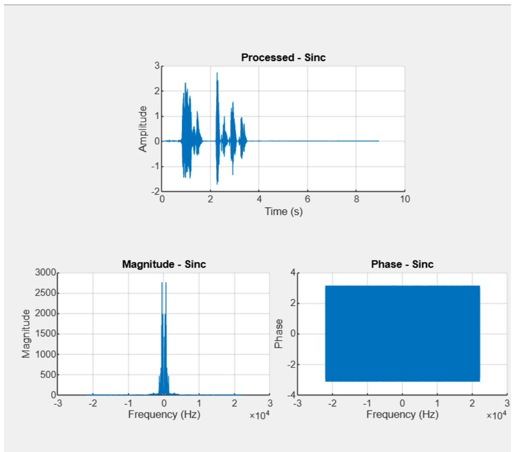

Figure 6: processed - Sinc function

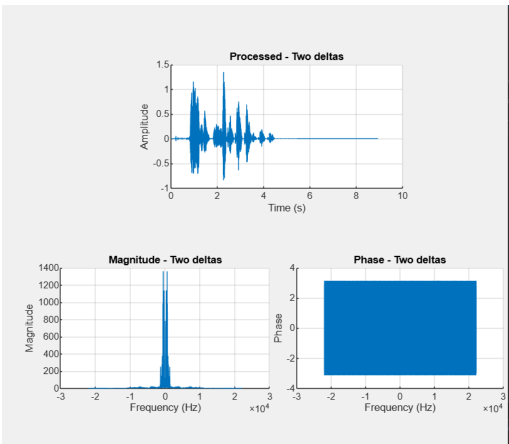

Figure 7: processed - Two delta

Figure 8: Processed speech signal channels

Delta function was chosen for applying the noise and filter because its effect is more evident, allowing for a clearer observation of the changes before the noise image.

After adding the noise with noise internsity (sigma) =0.2 , the amplitude increased significantly and the signal became noisy.

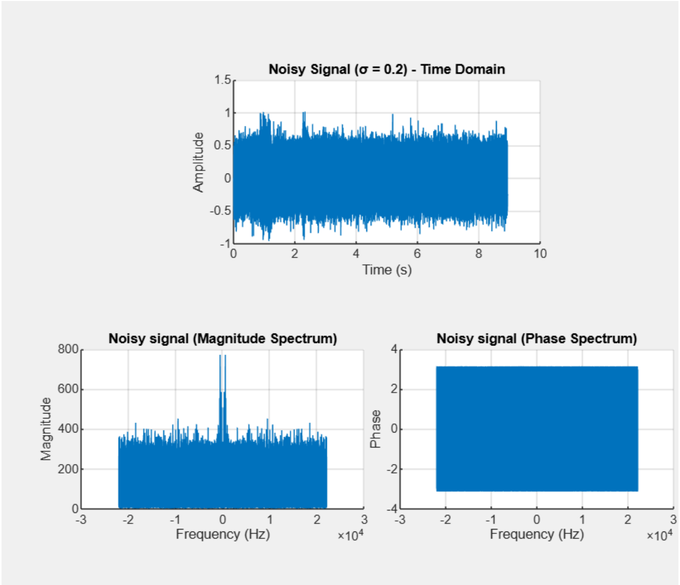

Figure 9: Noisy Speech Signal

After applying the filter, the signal became somewhat similar to the original, with some noise present 10

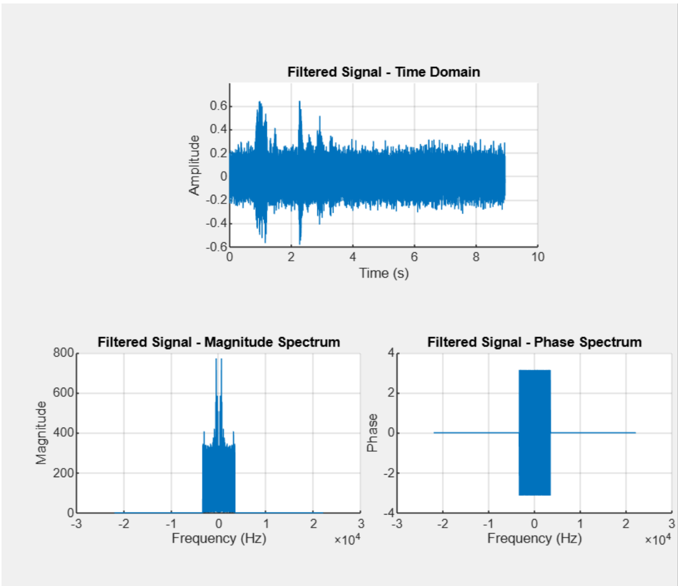

Figure 10: Filtered Speech Signal

0.4.2 Music File Results

The figures below show the processed results for the Music file:

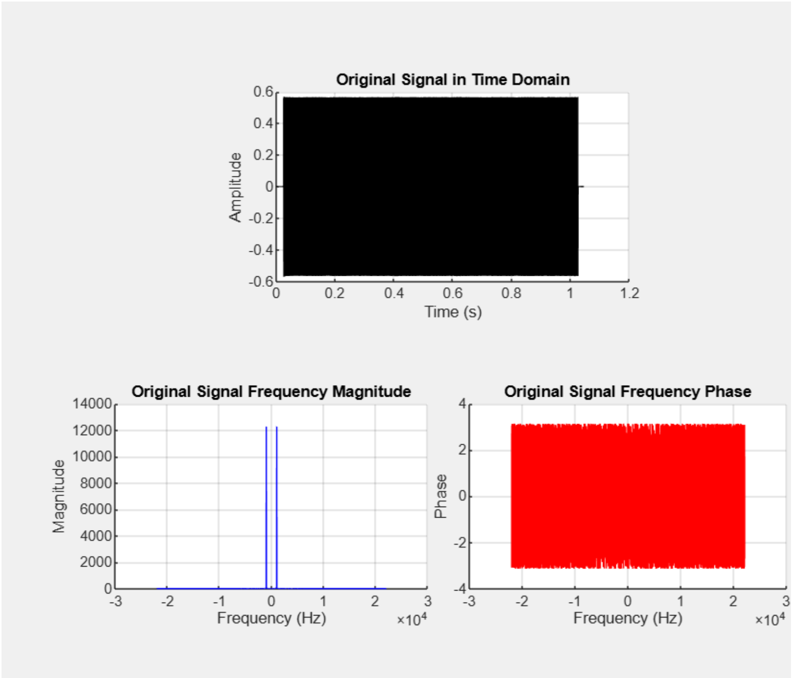

Figure 11: Original music Signal

The following figures show the effect of convolving the original music signal with different channel impulse responses:

12: Delta function keeps the signal unchanged, acting as an identity filter.
13: 5000 Hz exponential decay slightly increases amplitude.
14: 1000 Hz exponential decay further increases amplitude.
15: Sinc function smooths the signal and removes high-frequency components.
16: Two delta pulses , adding a delayed version of the original signal.

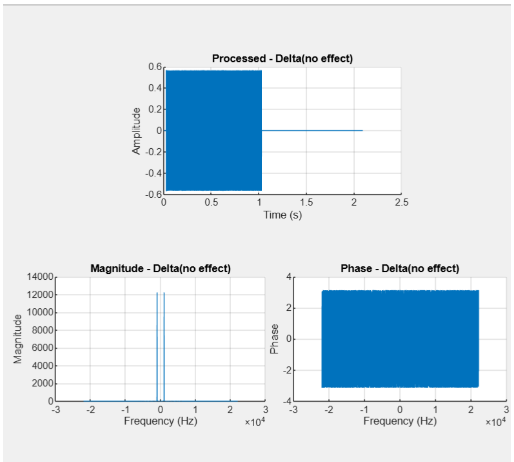

Figure 12: processed - Delta

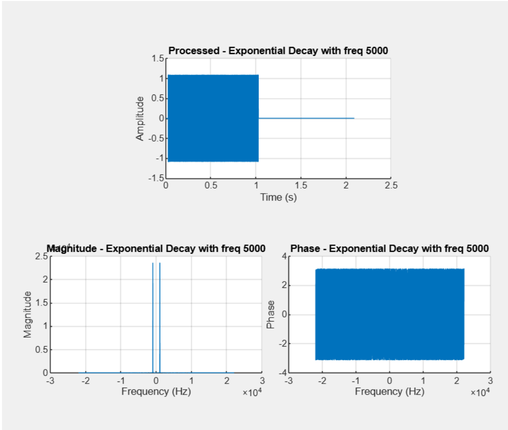

Figure 13: processed - 5000Hz

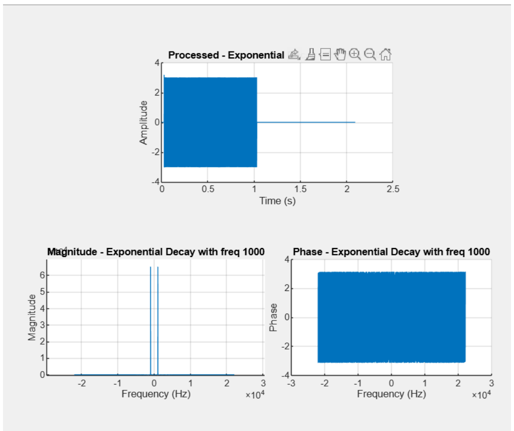

Figure 14: processed - 1000Hz

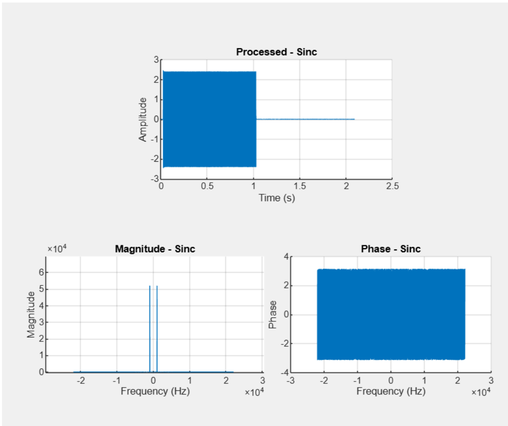

Figure 15: processed - Sinc function

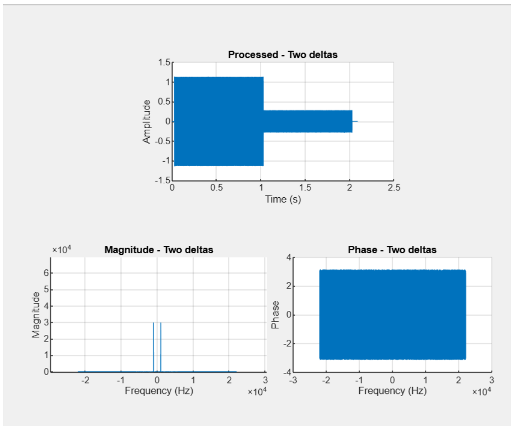

Figure 16: processed - Two delta

Figure 17: Processed speech signal channels

Delta function was chosen for applying the noise and filter because its effect is more evident, allowing for a clearer observation of the changes before the noise image.

After adding the noise with noise intensity(sigma) = 0.5 , the amplitude increased significantly and the signal became noisy.

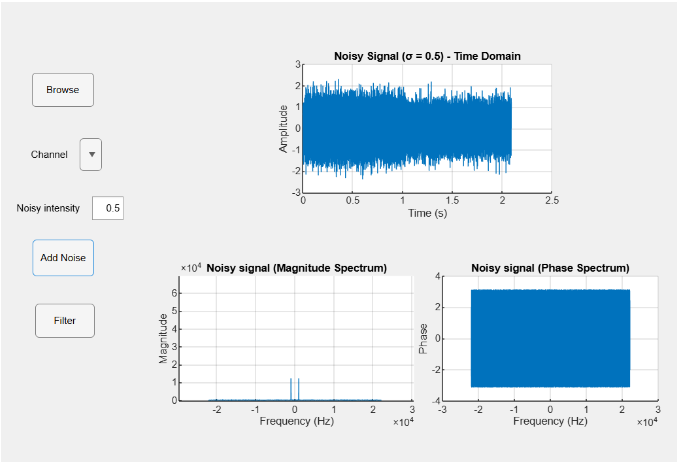

Figure 18: Noisy music Signal

After applying the filter, the signal became somewhat similar to the original, with some noise present 19

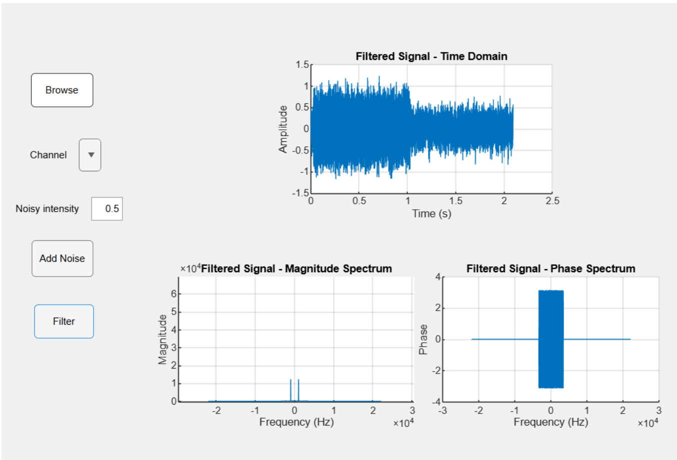

Figure 19: Filtered music Signal

0.5 Comparison

- The **speech signal** becomes closer to the original, with slight distortion.
- The **music signal** remains more distorted due to its complex and wide frequency.
- Filtering is more effective on speech because it has a simpler structure.

0.6 MATLAB Code Implementation

MATLAB script code was used to verify the effect of convolution, noise addition, and filtering on the signal. This allowed us to test and confirm the expected behavior of each stage in the system.

```
[file, location] = uigetfile({'*.mp3;*.wav'}, 'Select an audio File');

disp(['User selected ', fullfile(location, file)]);

% Load the audio file
full_path = fullfile(location, file);
[data, fs] = audioread(full_path);
data = mean(data, 2);


% Play the audio
player = audioplayer(data, fs);
play(player);  % To stop: type stop(player)

% Calculate the duration of the audio
duration = length(data) / fs;

% Time domain plot of original signal
t = linspace(0, duration, length(data));
figure;
plot(t, data);
title('Original Signal in Time Domain');
xlabel('Time (s)');

% Frequency domain plot of original signal
N = length(data);
f = linspace(-fs/2, fs/2, N);
Y = fftshift(fft(data));
Ymag = abs(Y);
Yphase = angle(Y);

figure;
subplot(2,1,1); % For Magnitude
plot(f, Ymag);
title('Original Signal Frequency Magnitude');
xlabel('Frequency (Hz)');

subplot(2,1,2); % For Phase
plot(f, Yphase);
title('Original Signal Frequency Phase');
xlabel('Frequency (Hz)');

% Channel selection
channel_options = {'Delta(no effect)', 'Exponential Decay with freq 5000', 'Exponential Decay with freq 1000', 'Sinc', 'Two deltas'};
[channel_idx, ok] = listdlg('Name', 'Channel Selection', ...
                            'PromptString', 'Select a channel type:', ...
                            'ListString', channel_options, ...
                            'SelectionMode', 'single');

% Generate impulse response
%t = t(:);  % Ensure column vector
switch channel_idx
    case 1
```

```
        impulse_response = [1 zeros(1,length(data)-1)];  % Delta
    case 2
        impulse_response = exp(-2*pi*5000*t);  % Fast exponential decay
    case 3
        impulse_response = exp(-2*pi*1000*t);  % Slower exponential decay
    case 4
        x = 2* 3000 * t;
        impulse_response = sinc(x); % Sinc function
    case 5
        impulse_response = [2 zeros(1, fs-2) 0.5 zeros(1, (length(data)-fs))];
end

%impulse_response = impulse_response(:);  % Force column vector

% Convolve audio with impulse response
convolved_data = conv(data, impulse_response);

% Time domain plot of processed signal

%t_conv= linspace(0,(duration)*2,length(data)*2-1);

t_conv = linspace(0, (length(convolved_data)-1) / fs, length(convolved_data));
%t_conv = t;
figure;
plot(t_conv, convolved_data);
title(['Processed Signal (Channel: ', channel_options{channel_idx}, ') - Time Domain']);
xlabel('Time (s)');

% Frequency domain plot of processed signal
N_conv = length(convolved_data);
f_conv = linspace(-fs/2, fs/2, N_conv);
Y_conv = fftshift(fft(convolved_data));
Ymag_conv = abs(Y_conv);
Yphase_conv = angle(Y_conv);

figure;
subplot(2,1,1); % For Magnitude
plot(f_conv, Ymag_conv);
title(['Processed Signal Magnitude (Channel: ', channel_options{channel_idx}, ')']);
xlabel('Frequency (Hz)');

subplot(2,1,2); % For Phase
plot(f_conv, Yphase_conv);
title(['Processed Signal Phase (Channel: ', channel_options{channel_idx}, ')']);
xlabel('Frequency (Hz)');

% Noise Addition
sigma = input('Enter the noise intensity (sigma): ');
Z = sigma * randn(1,length(convolved_data));

% Add noise to the channel output
noisy_signal = convolved_data + Z;


% play the noisy audio
player_noisy = audioplayer(noisy_signal, fs);
```

```
disp('Playing noisy signal...');
play(player_noisy);
pause(duration);
stop(player_noisy);
disp('Noisy signal stopped.');

% Time domain plot of noisy signal
figure;
plot(t_conv, noisy_signal);
title(['Noisy Signal (  = ', num2str(sigma),') - Time Domain']);
xlabel('Time (s)');

% Frequency domain plot of noisy signal
N_noisy = length(noisy_signal);
f_noisy = linspace(-fs/2, fs/2, N_noisy);
Y_noisy = fftshift(fft(noisy_signal)); %Y _noisy is the freq of original signal and the noise
Ymag_noisy = abs(Y_noisy);
Yphase_noisy = angle(Y_noisy);

figure;
subplot(2,1,1); % Magnitude
plot(f_noisy, Ymag_noisy);
title(['Noisy Signal (  = ', num2str(sigma), ') - Frequency Domain(Magnitude Spectrum)']);
xlabel('Frequency (Hz)');

subplot(2,1,2); % Phase
plot(f_noisy, Yphase_noisy);
title(['Noisy Signal (  = ', num2str(sigma), ') - Frequency Domain(Phase Spectrum)']);
xlabel('Frequency (Hz)');

% Receiver
% Apply ideal low-pass filter with cutoff at 3400 Hz

cutoff_freq = 3400; % Hz
Y_filtered = Y_noisy;
% Zero frequencies outside [-3400, 3400] Hz
f_filter = linspace(-fs/2, fs/2, length(Y_filtered));
Y_filtered(abs(f_filter) > cutoff_freq) = 0;
% Convert back to time domain

filtered_signal = real(ifft(ifftshift(Y_filtered)));

%filtered_signal = filtered_signal(:);


% Play the filtered audio
player_filtered = audioplayer(filtered_signal, fs);
disp('Playing filtered signal...');
play(player_filtered);
pause(duration);
stop(player_filtered);
disp('Filtered signal stopped.');

% Time domain plot of noisy signal
figure;
plot(t_conv, filtered_signal);
title(['Filtered Signal (', channel_options{channel_idx}, ') - Time Domain']);
```

```
xlabel('Time (s)');

% Frequency domain plot of filtered signal
Y_filtered_mag = abs(Y_filtered);
Y_filtered_phase = angle(Y_filtered);

figure;
subplot(2,1,1);
plot(f_filter, Y_filtered_mag);
title(['Filtered Signal (', channel_options{channel_idx}, ') - Magnitude Spectrum']);
xlabel('Frequency (Hz)');

subplot(2,1,2);
plot(f_filter, Y_filtered_phase);
title(['Filtered Signal (', channel_options{channel_idx}, ') - Phase Spectrum']);
xlabel('Frequency (Hz)');
```