Title:

"Analyzing a noisy music file and designing filters for noise removal and Removal of different musical instruments effects"

1. Objectives of project:

Develop an Interactive Audio Filter GUI:

- Create a user-friendly graphical interface for real-time audio signal processing, allowing users to load, visualize, and modify audio signals.

Implement Multiple Filter Types:

- Integrate various filter types (Lowpass, Highpass, Bandpass, Bandstop and Moving average filter) to manipulate audio signals for diverse applications, such as noise reduction or frequency isolation.

Enable Real-Time Noise Adjustment and Removal:

- Provide functionality to add and adjust noise levels in the audio signal, as well as remove noise to restore the original signal, offering real-time manipulation and visualization.

Visualize Audio Signals in Time and Frequency Domains:

- Implement real-time visualization of audio signals in both time and frequency domains using the Fast Fourier Transform (FFT) and filter frequency response plots.

Allow Playback and Pause of Processed Audio:

- Enable users to play and pause the processed audio, with the option to switch between original and filtered versions, making it easier to assess the impact of different filters.

Provide Customizable Filter Parameters:

- Allow users to adjust filter parameters such as cutoff frequencies for lowpass, highpass, and band filters, enhancing flexibility in signal processing.

2. Introduction:

In the field of digital signal processing (DSP), audio signals are often manipulated and filtered to achieve desired effects or to remove unwanted noise. Audio filtering plays a crucial role in a wide range of applications, from music production to speech enhancement and medical signal processing. The effectiveness of audio processing largely depends on the flexibility and precision of the tools used. In this context, a graphical user interface (GUI) provides an intuitive way for users to interact with complex signal processing algorithms without requiring deep technical expertise.

This project focuses on the development of an **Interactive Audio Filter GUI** that allows users to load, visualize, filter, and manipulate audio signals in real-time. The system offers a variety of filtering options, including **Lowpass**, **Highpass**, **Bandpass**, **Bandstop** and **Moving Average** filters, providing users with the ability to isolate or attenuate specific frequencies within an audio signal. Additionally, the system enables real-time **noise adjustment and removal**, simulating different noise conditions and providing an option to restore the original signal.

By integrating advanced signal processing techniques with a user-friendly interface, this project aims to enhance the accessibility of audio filtering tools. It provides a flexible platform for exploring the effects of different filters, observing their impact in both the **time domain** (signal waveform) and **frequency domain** (FFT analysis). The system also features a **play/pause functionality**, enabling users to listen to both the original and filtered audio in real-time, making it ideal for audio engineers, researchers, and students.

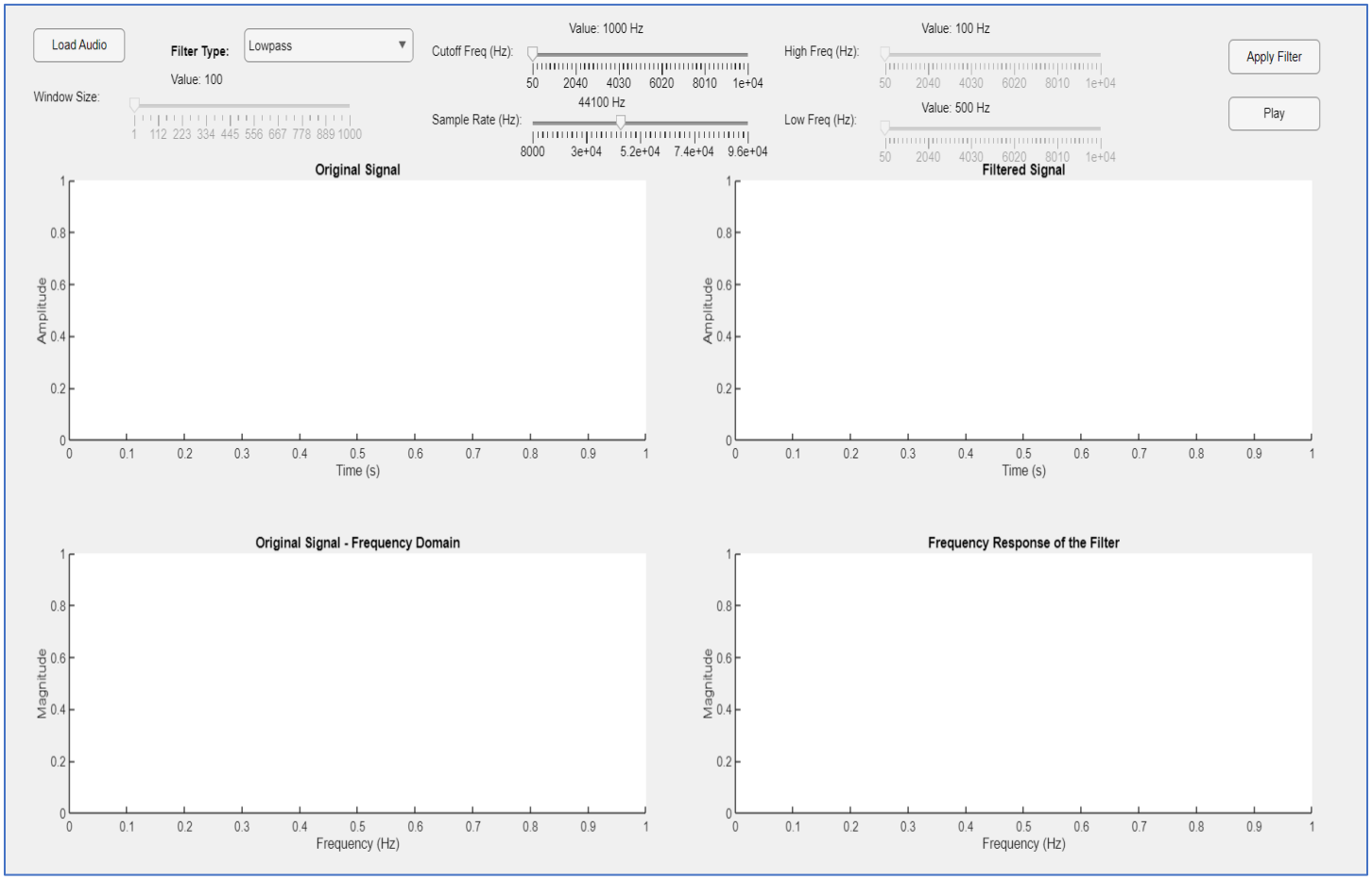
The core objective of this project is to provide an easy-to-use but powerful tool for interactive audio signal processing. It offers the flexibility to adjust various filter parameters dynamically, making it useful for a range of practical applications, from noise reduction in recordings to sound design for multimedia projects. The system is designed to be cross-platform, ensuring that it is accessible to a broad audience.

This project demonstrates the integration of **digital signal processing algorithms** with a **graphical user interface**, providing both the technical robustness needed for audio processing and the user-cantered design that simplifies its operation. It serves as a foundation for further exploration into advanced audio filtering techniques, as well as potential applications in other domains such as telecommunications and medical diagnostics.

3. Without Noise GUI:

3.1. GUI Description:

This is the first GUI window of the project, designed to operate in a noise-free environment. It provides options to import an audio file, along with play/pause functionality. The GUI offers various filtering options, including Lowpass, Highpass, Bandpass, Bandstop, No Filter, and Moving Average filters. It provides the flexibility of adjusting the cut off frequency,low and high frequency limits in case of band selection, window size adjustment in case of Moving Average filter and adjustment of sampling rate. Additionally, it visualizes both the input and output audio signals, demonstrating the effects of the filters on the original signal. The GUI also displays the original and filtered signals in the frequency domain, as well as the frequency response of the applied filter.



3.2.Results:

No filter Applied (fig 3a)

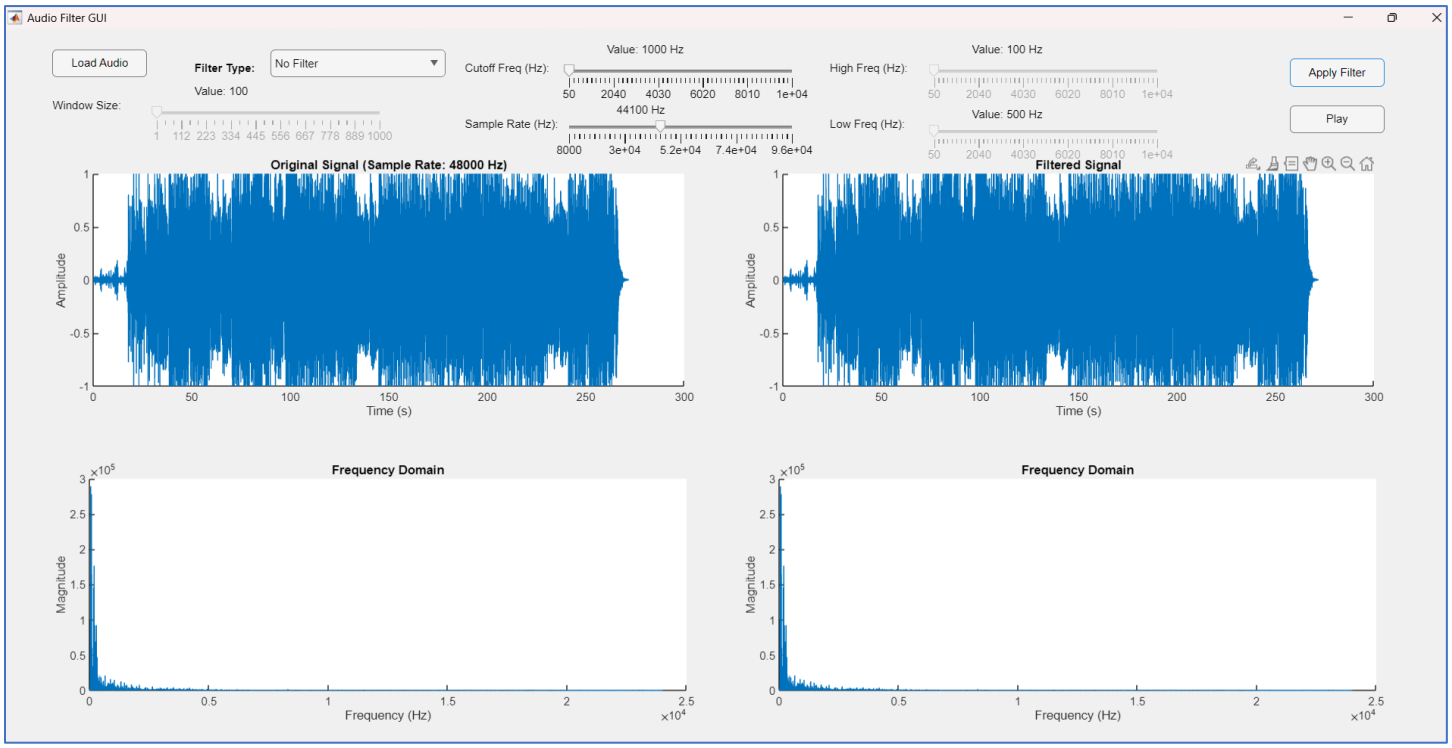


***Fig 3a**: we see that the input signal and output are same as we have applied no filter in the signal*

Lowpass filter applied (fig 3b)

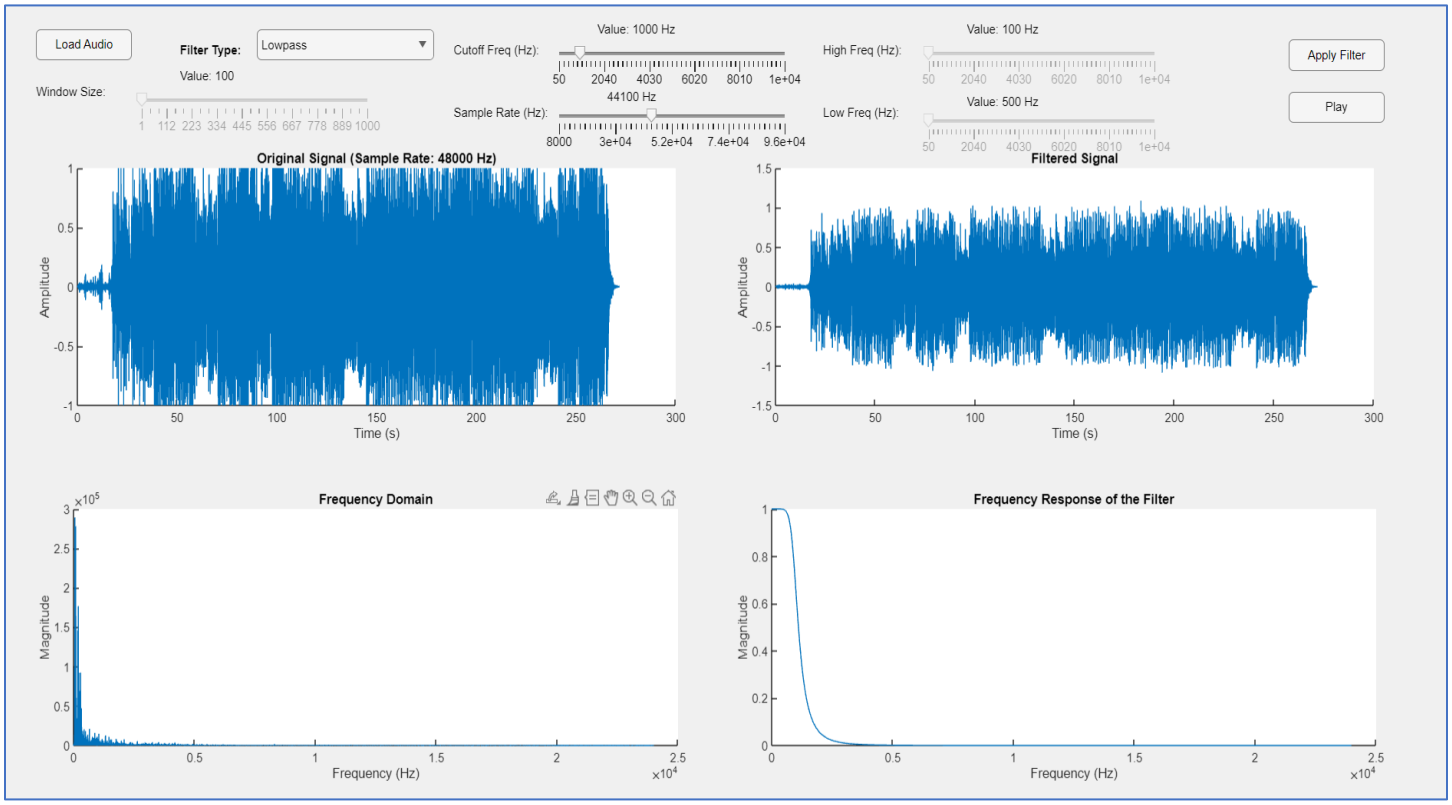


***Fig 3b**: observe the frequency response of the filter as it cut off the high frequency components due to which the visual of input signal has been changed as it can be seen in the Filtered signal graph*

Highpass filter applied (fig 3c)

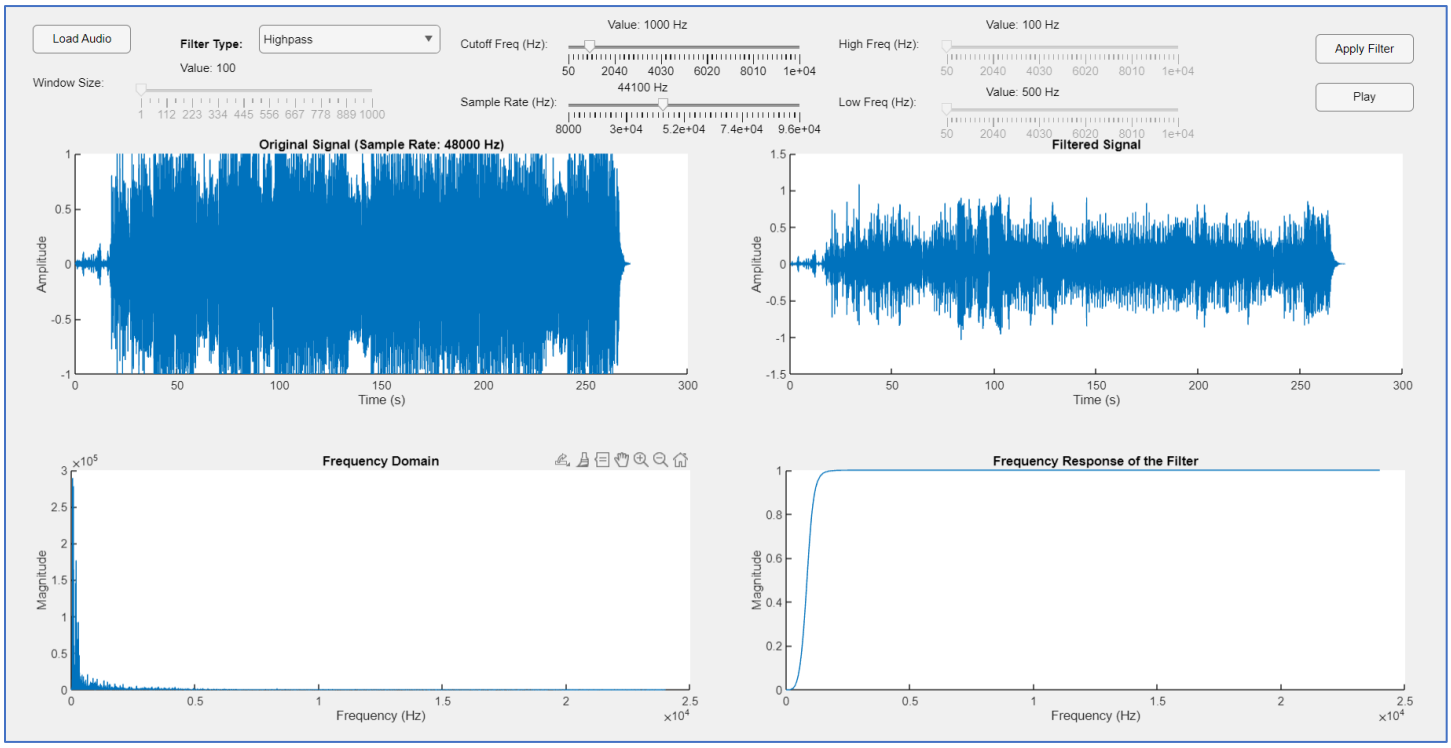


Fig 3c: *observe the frequency response of the filter as it cut off the low frequency components due to which the visual of input signal has been changed as it can be seen in Filtered Signal plot*

Bandpass filter applied (fig 3d)

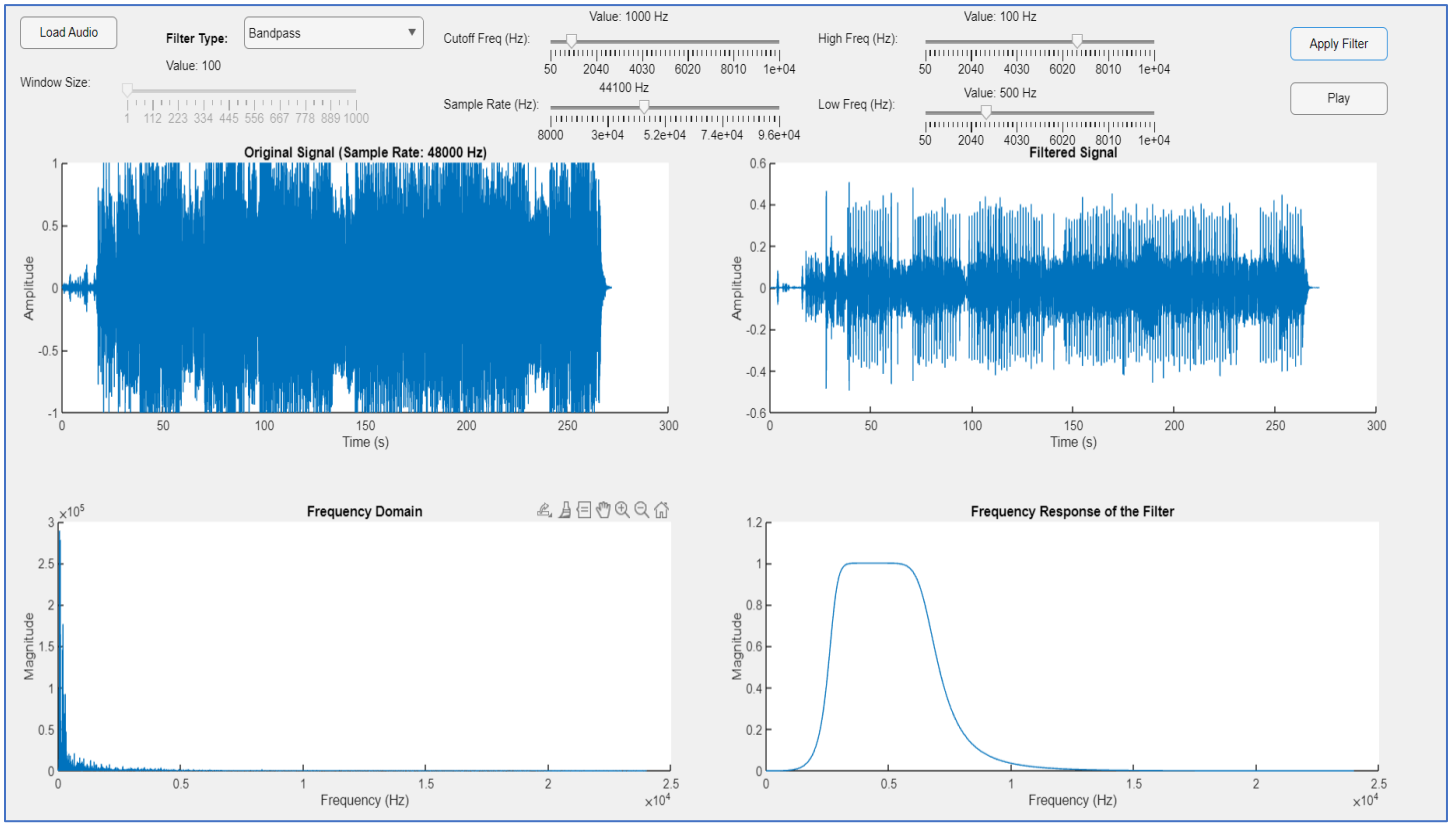


Fig 3d: *observe the frequency response of the filter as it allows a specific band of frequency to be passed, due to which the visual of input signal has been changed as it can be seen in Filtered Signal plot*

Band stop filter (fig 3e):

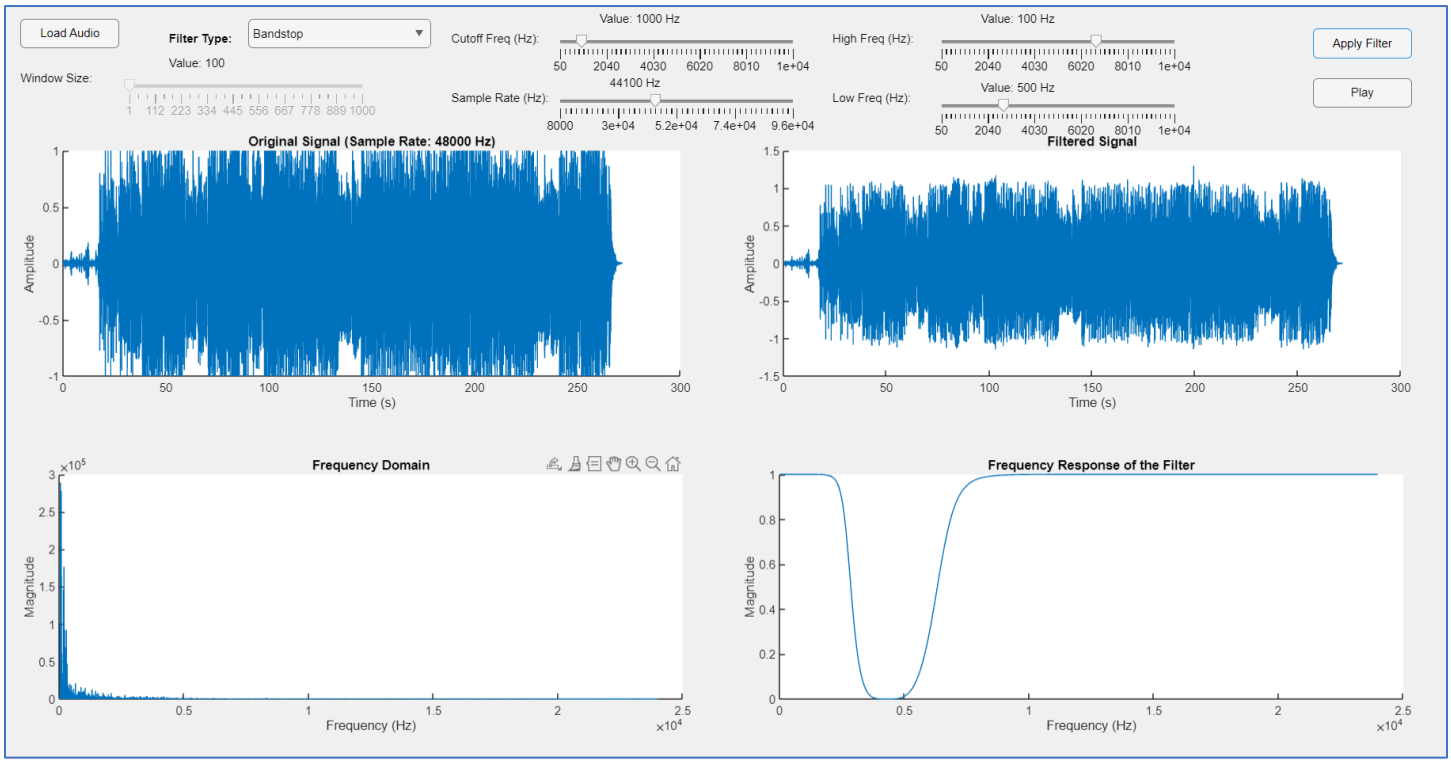


Fig 3e: *observe the frequency response of the filter as it stops a specific band of frequency, due to which the visual of input signal has been changed as it can be seen in Filtered Signal plot*

Moving Average Filter (fig 3f)

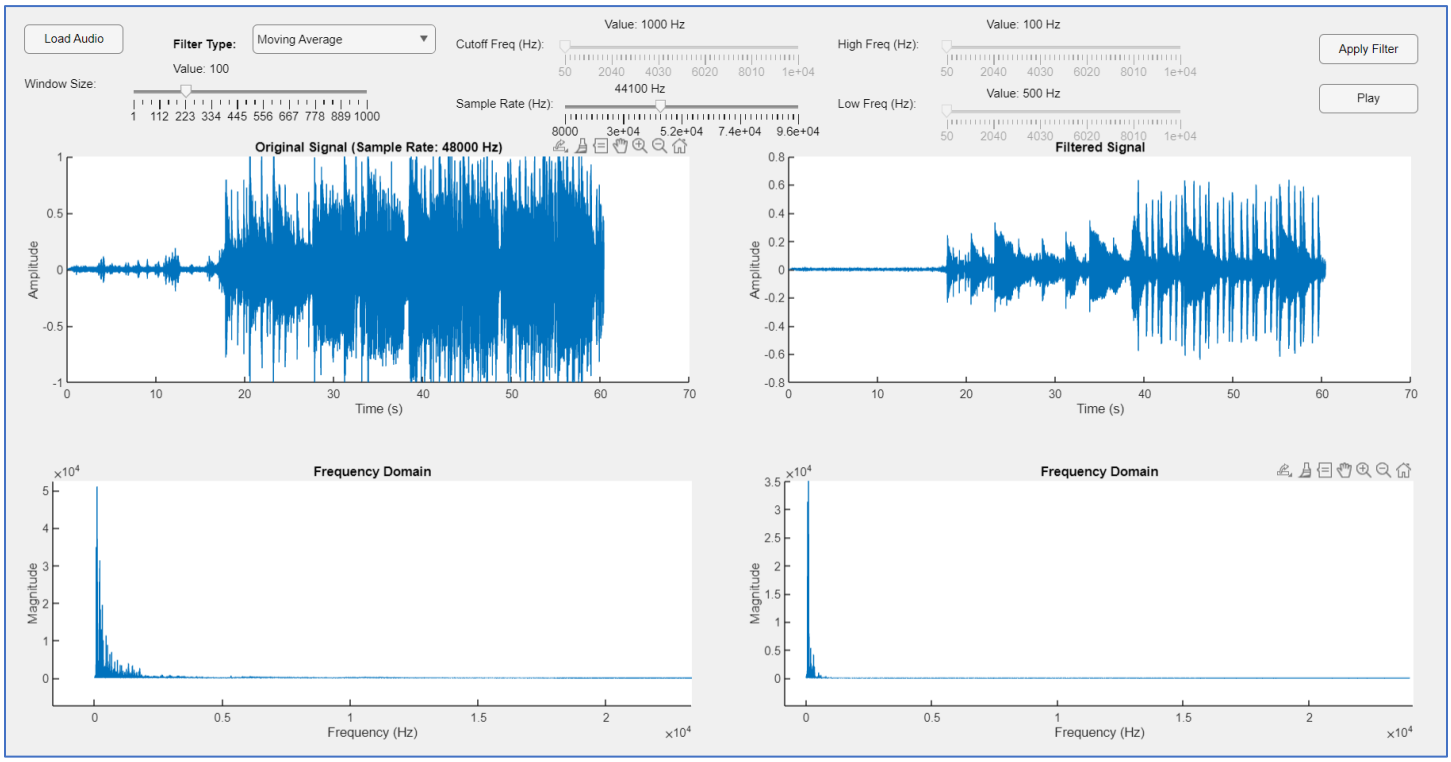


Fig 3f: *observe the frequency domain of the filtered signal it smooths the signal and no high frequency components can be seen due to the smoothing nature of moving average filter.*

3.3.Filter design and Apllication Code:

```
function applyFilter()
    if isempty(audioData)
        uialert(fig, 'Please load an audio file first.', 'Error');
        return;
    end

    % Get Filter Type and Parameters
    filterType = dropdownFilter.Value;
    cutoffFreq = sliderCutoff.Value;
    windowSize = sliderMovAvg.Value;

    try
        switch filterType
            case 'Lowpass'
                [b, a] = butter(4, cutoffFreq / (sampleRate / 2), 'low');
            case 'Highpass'
                [b, a] = butter(4, cutoffFreq / (sampleRate / 2),
'high');
            case 'Bandpass'
                lowFreq = sliderLowFreq.Value;
                highFreq = sliderHighFreq.Value;
                if lowFreq >= highFreq
                    uialert(fig, 'Low frequency must be less than high
frequency.', 'Error');
                    return;
                end
                [b, a] = butter(4, [lowFreq, highFreq] / (sampleRate /
2), 'bandpass');
            case 'Bandstop'
                lowFreq = sliderLowFreq.Value;
                highFreq = sliderHighFreq.Value;
                if lowFreq >= highFreq
                    uialert(fig, 'Low frequency must be less than high
frequency.', 'Error');
                    return;
                end
                [b, a] = butter(4, [lowFreq, highFreq] / (sampleRate /
2), 'stop');
            case 'No Filter'
                filteredAudio = audioData; % No filtering applied, just
use the original signal
                plot(ax2, (0:length(filteredAudio)-1) / sampleRate,
filteredAudio); % Plot the original signal
                % Plot frequency response for No Filter (same as original
signal)
                plotFFT(axFFT2, filteredAudio, sampleRate);
                return;
            case 'Moving Average'
                filteredAudio = movmean(audioData, windowSize); % Apply
moving average filter
```

```
                plot(ax2, (0:length(filteredAudio)-1) / sampleRate,
filteredAudio); % Plot filtered signal
                % Plot frequency response for Moving Average filter
                plotFFT(axFFT2, filteredAudio, sampleRate);
                return;
            otherwise
                error('Unknown filter type');
        end

        % Apply Butterworth Filter and Plot Frequency Response
        filteredAudio = filter(b, a, audioData);
        plot(ax2, (0:length(filteredAudio)-1) / sampleRate,
filteredAudio); % Plot filtered signal
        plotFrequencyResponse(axFFT2, b, a, sampleRate); % Plot frequency
response

    catch
        uialert(fig, 'An error occurred while applying the filter.',
'Error');
    end
end
```

4. With Noise GUI:

This is the second GUI window of the project, it provides the options to import an audio file, it has the feature of adding noise to audio signals, and control the intensity of noise. The GUI offers various filtering options, including Lowpass, Highpass, Bandpass, Bandstop, and No Filter, it provides the flexibility of adjusting the cut off frequency,low and high frequency in case of band selection, and adjustment of sampling rate Additionally, it visualizes both the input and output audio signals, demonstrating the effects of the filters , noise and other adjustable parameters on the original signal. The GUI also displays the original and filtered signals in the frequency domain.

GUI Buttons and sliders (fig 4a)

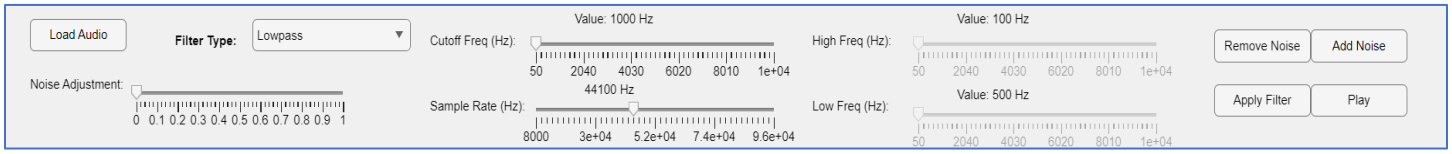


Fig 4a: *This figure shows the different features which are presented and accessible as buttons, and adjustable parameters as slider windows*

4.1. Results:

No filter applied on non-noisy audio (fig 4b)

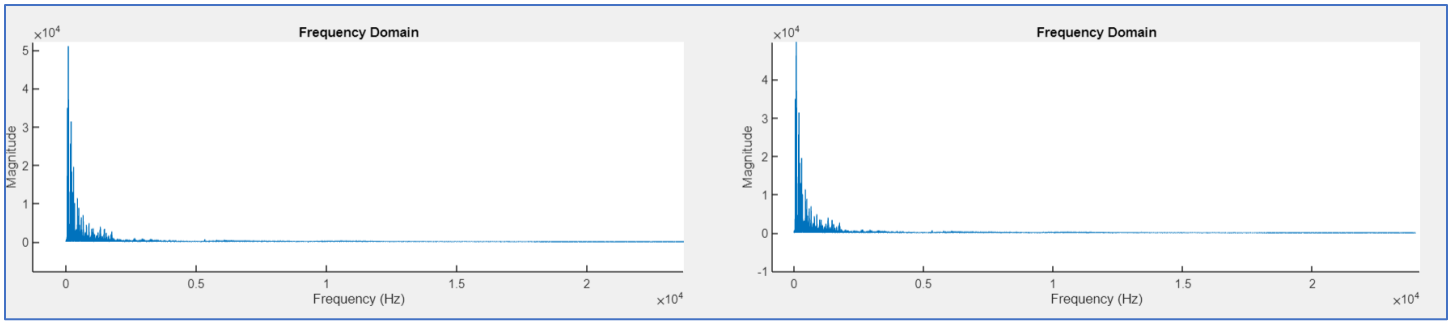


Fig 4b: *The input and output in frequency domain remains same as we have added no noise and applied any filter to it*

No filter applied on a noisy audio (fig 4c)

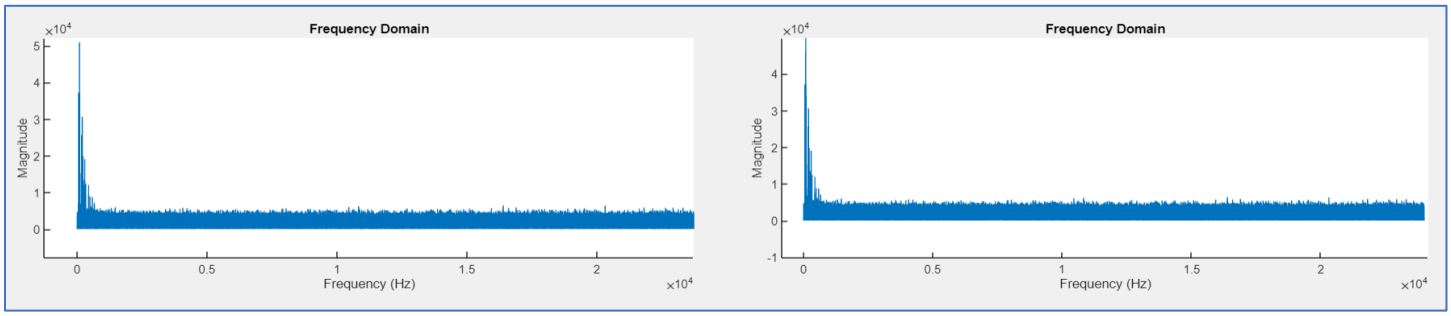


Fig 4b: *In this figure we can see the white gaussian noise effect, but the input and out remains the same as we have applied no filter to it*

Lowpass filter effect on noisy signal (fig 4d)

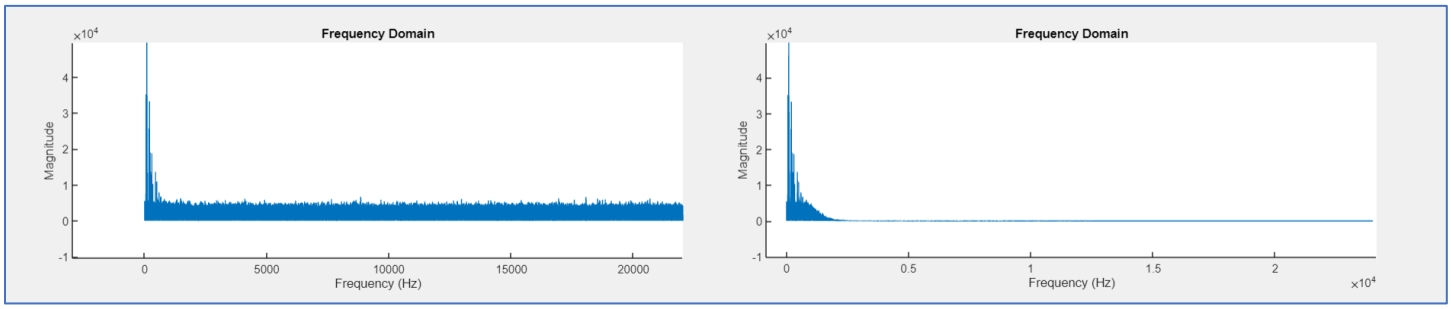


Fig 4d: *In this figure we can see the white gaussian noise effect on the input signal, but due to the application of low pass filter only low frequency components can be seen at the output*

High pass filter effect on noisy audio (fig 4e)

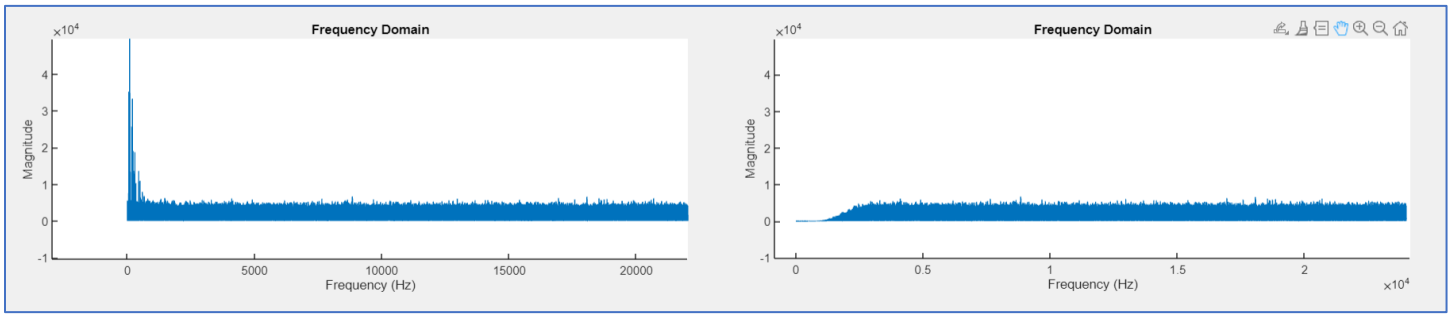


Fig 4e: *In this figure we can see the white gaussian noise effect on the input signal, but due to the application of high pass filter only high frequency components can be seen at the output*

Band pass filter applied on noisy audio (fig 4d)

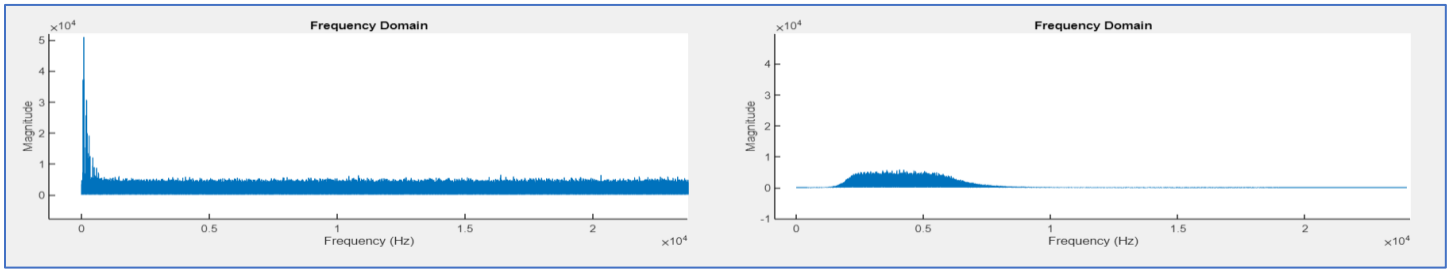


Fig 4d: *In this figure we can see the white gaussian noise effect on the input signal, but due to the application of band pass filter only specific band of frequency components can be seen at the output*

Band stop filter applied on noisy audio (fig 4f)

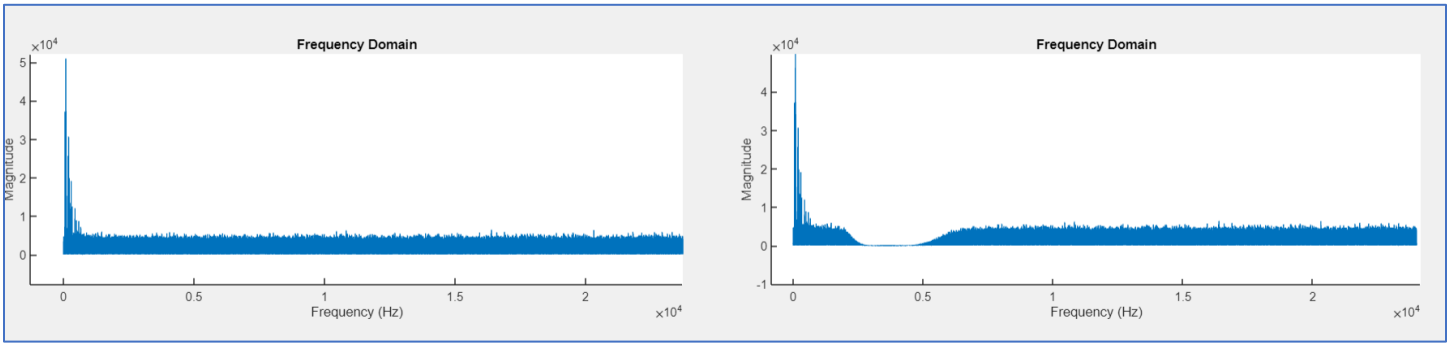


Fig 4f: *In this figure we can see the white gaussian noise effect on the input signal, but due to the application of bandstop filter the specific band of frequency is cut off while remaining is allowed to pass.*

4.2.Filter design and Apllication Code:

```
function applyFilter()
    if isempty(audioData)
        uialert(fig, 'Please load an audio file first.', 'Error');
        return;
    end

    % Get Filter Type and Parameters
    filterType = dropdownFilter.Value;
    cutoffFreq = sliderCutoff.Value;

    try
        switch filterType
            case 'Lowpass'
                [b, a] = butter(4, cutoffFreq / (sampleRate / 2), 'low');
            case 'Highpass'
                [b, a] = butter(4, cutoffFreq / (sampleRate / 2),
'high');
            case 'Bandpass'
                lowFreq = sliderLowFreq.Value;
                highFreq = sliderHighFreq.Value;
                if lowFreq >= highFreq
                    uialert(fig, 'Low frequency must be less than high
frequency.', 'Error');
                    return;
                end
                [b, a] = butter(4, [lowFreq, highFreq] / (sampleRate /
2), 'bandpass');
            case 'Bandstop'
                lowFreq = sliderLowFreq.Value;
                highFreq = sliderHighFreq.Value;
                if lowFreq >= highFreq
                    uialert(fig, 'Low frequency must be less than high
frequency.', 'Error');
```

```
                    return;
                end
                [b, a] = butter(4, [lowFreq, highFreq] / (sampleRate /
2), 'stop');
            case 'No Filter'
                filteredAudio = audioData;  % No filtering applied, just
use the original signal
                plot(ax2, (0:length(filteredAudio)-1) / sampleRate,
filteredAudio);  % Plot the original signal
                plotFFT(axFFT2, filteredAudio, sampleRate);  % Update the
FFT with the filtered signal
                return;
            otherwise
                error('Unknown filter type');
end
% Apply Butterworth Filter and Plot Frequency Response
        filteredAudio = filter(b, a, audioData);
        plot(ax2, (0:length(filteredAudio)-1) / sampleRate,
filteredAudio);  % Plot filtered signal
        plotFrequencyResponse(axFFT2, b, a, sampleRate);  % Plot
frequency response

        % Update the FFT of the filtered signal
        plotFFT(axFFT2, filteredAudio, sampleRate);

    catch
        uialert(fig, 'Error applying the filter. Please check the filter
parameters.', 'Error');
    end
end
```

4.3.Noise Addition Code:

```
% Callback: Add Noise
function addNoise()
    noiseLevel = sliderNoiseAdjustment.Value;
    noise = noiseLevel * randn(size(audioData));  % White Gaussian noise
    audioData = originalAudioData + noise;  % Add noise to the original
audio data

    % Update the visualization for the noisy signal
    cla(ax1);  % Clear the original signal plot
    plot(ax1, (0:length(audioData)-1) / sampleRate, audioData);
    title(ax1, 'Original + Noise');  % Update title to indicate noise
added

    % Update the FFT plot to show the frequency domain effect of the
noise
    plotFFT(axFFT1, audioData, sampleRate);
end
```

5. Observations and Conclusion:

Based on the above output and experiment, it is evident that the implemented filters effectively allow for the isolation or removal of specific frequency components from the audio signal. By applying different types of filters**Lowpass**, **Highpass**, **Bandpass**, **Bandstop**, and **Moving Average** we are able to selectively target certain frequency ranges and either attenuate or eliminate unwanted frequencies. This demonstrates the utility of filters in obtaining the desired output and focusing on the most relevant or interesting data within the signal.

1. **Lowpass Filter:** This filter successfully attenuates high-frequency noise or unwanted signals, allowing lower frequencies to pass through. The effect can be clearly observed in the time-domain signal as well as the frequency-domain representation (FFT), where the higher frequencies are reduced.
2. **Highpass Filter:** By removing low-frequency components, the highpass filter is useful for eliminating hums or other low-frequency disturbances. In the frequency-domain plot, we observe a reduction in the low-frequency components, leaving the higher frequencies intact.
3. **Bandpass Filter:** The bandpass filter enables the extraction of a specific frequency range, which is beneficial in applications like speech or music signal processing, where certain frequency bands are of primary interest. The plot shows the passing of frequencies within the defined band while attenuating the others.
4. **Bandstop Filter:** This filter is effective in removing specific unwanted frequency ranges, such as noise or interference at certain frequencies. The frequency-domain plot clearly indicates the attenuation of the targeted frequency range, which leads to a cleaner signal in the time domain.
5. **Noise Adjustment and Removal:** The system also demonstrates the ability to simulate and control the addition of noise to the signal, as well as the removal of noise, restoring the signal to its original state. The noise adjustment feature is particularly useful for testing how different filtering techniques can handle noisy data.
6. **Moving Average Filter:** The **Moving Average Filter** works by averaging the signal over a window of samples, which helps in smoothing out rapid variations or noise in the signal. The effect of the moving average filter is noticeable in the time-domain plot, where it reduces the sharp fluctuations and smooths the signal. This filter is particularly effective for removing high-frequency noise, but it can also distort the signal to some extent by introducing a delay or smoothing out sudden transitions. In the frequency-domain plot, the effect of the moving average filter is visible as a reduction in the higher frequency components, as it effectively acts as a low-pass filter. However, this filter might not be suitable for all applications where preserving high-frequency details is critical.

In conclusion, the ability to apply filters in real-time and observe their effects both in the time domain and frequency domain proves the utility of this tool in processing and analysing audio signals. It allows users to manipulate the signal for specific applications, such as noise reduction, frequency isolation, and signal enhancement, thereby providing valuable insights and clean data for further analysis or use. The addition of the moving average filter provides an extra layer of flexibility in smoothing noisy signals, although its use should be considered carefully depending on the nature of the data and the desired outcome.