Digital Signal Processing (DSP) Project

Analyzing a noisy music file and designing filters for noisy removal and

Removal of different musical instruments effects

Develop a MATLAB program which imports a noisy music file (or adds noise after it is imported). Analyze the music file using different frequency domain analysis techniques to display the frequency content of the data. Design a filter to remove the noise content of the data and then play the music file after noise removal. Since music file is composed of sounds from different musical instruments, design different filters to remove the effects of specific musical instruments. The program should allow visualization of these separated signals and enable playing them individually.

Additionally, design an interactive graphical user interface (GUI) in MATLAB that allows users to apply various filters to the input music signal. The GUI should visualize the filtered output signals, demonstrating the effect of different filters on the input signal. All tuning and filter selection should be seamlessly handled through the GUI.

Abstract

This report discusses the development of a MATLAB program designed to process noisy music signals. The system allows users to import a music file, add synthetic noise, and analyze the frequency content using techniques like FFT and spectrogram. It includes filters to remove noise and isolate musical instrument sounds. An interactive GUI enables users to apply filters, visualize the results, and play back the cleaned signals, improving audio quality.

1. Introduction

Audio signals often contain unwanted noise or overlapping sounds from different musical instruments. This project focuses on analyzing noisy music, removing noise, and isolating individual instruments to enhance the clarity of the music. The system is implemented in MATLAB with a GUI for ease of use.

2. System Design

- **Frequency Analysis**: The music file is analyzed using FFT and spectrogram to identify the frequency components.
- **Noise Removal**: Filters such as low-pass, high-pass, and band-pass are designed to remove various types of noise.
- **Instrument Isolation**: Band-pass filters are used to isolate specific musical instrument frequencies.

- **GUI Design**: The GUI allows users to upload the music file, apply filters, visualize the filtered outputs, and play back the cleaned signals.

2.1 Filter Design:

Various filters are implemented to remove noise and isolate musical instrument sounds:

- **Noise Removal Filters**: Butterworth filters are designed to eliminate unwanted noise, such as background hum or static.
- **Instrument Isolation Filters**: Band-pass filters are created to target the specific frequency ranges where individual instruments typically operate, helping to isolate their signals while minimizing interference from other sources.
- The performance of each filter is validated using MATLAB's Signal Analyzer tool, ensuring they operate as intended.

3. Implementation

- **Signal Processing**: The audio file is loaded, noise is added, and frequency analysis is performed.
- **Filter Design**: Filters are designed using MATLAB's `designfilt` and applied to remove noise and isolate instruments.
- **GUI Development**: The GUI includes controls for adjusting filter parameters and visualizing the effect on the signal.

4. Testing and Evaluations

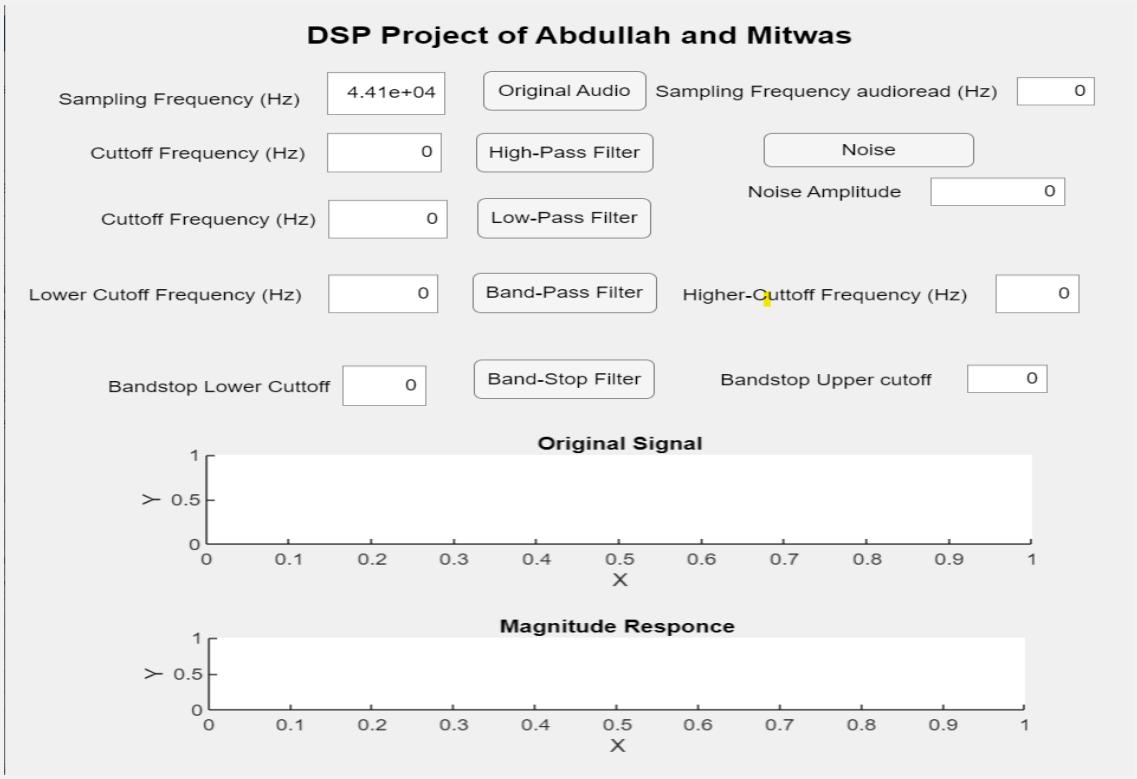
4.1 Frequency Analysis
Spectrograms and FFT plots reveal distinct frequency components corresponding to both noise and musical instruments. The frequency analysis successfully identifies the noise components and helps design filters that target specific ranges for removal.

4.2 Filter Performance
The noise removal filters significantly improve the quality of the signal, as measured by Signal-to-Noise Ratio (SNR) metrics. The instrument isolation filters are effective at isolating the frequencies associated with specific instruments, with minimal distortion to the original signal. The GUI facilitates real-time visualization of the effects of these filters and allows users to listen to the original, filtered, and isolated signals.

4.3 User Interaction via GUI

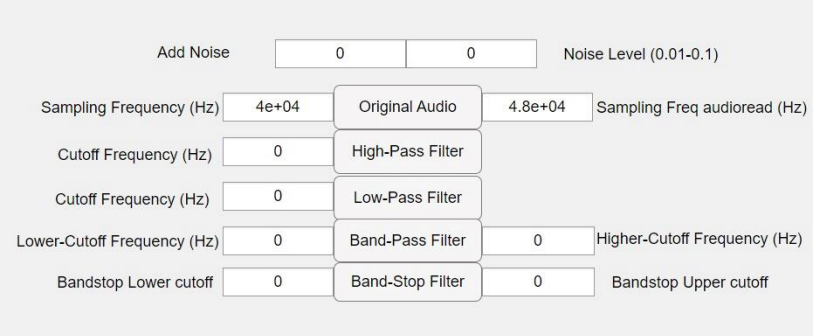
- The GUI provided an intuitive platform for exploring filter effects, improving accessibility for non-expert users.

4.4 Interactive Graphical User Interface (GUI) Design

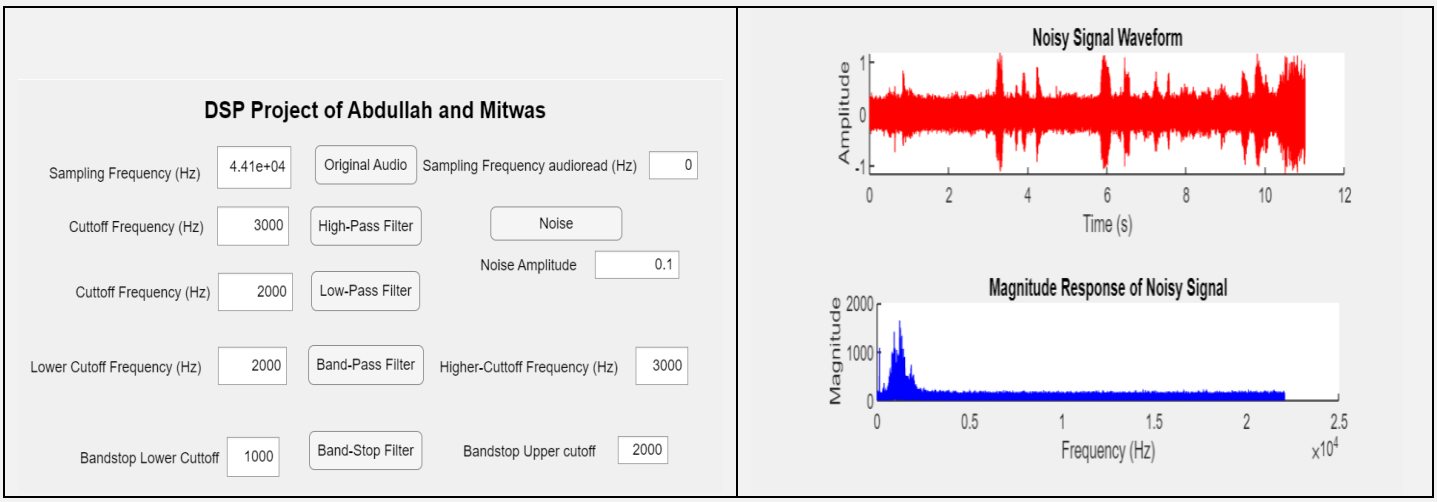


5. Results and Discussion

5.1 original audio

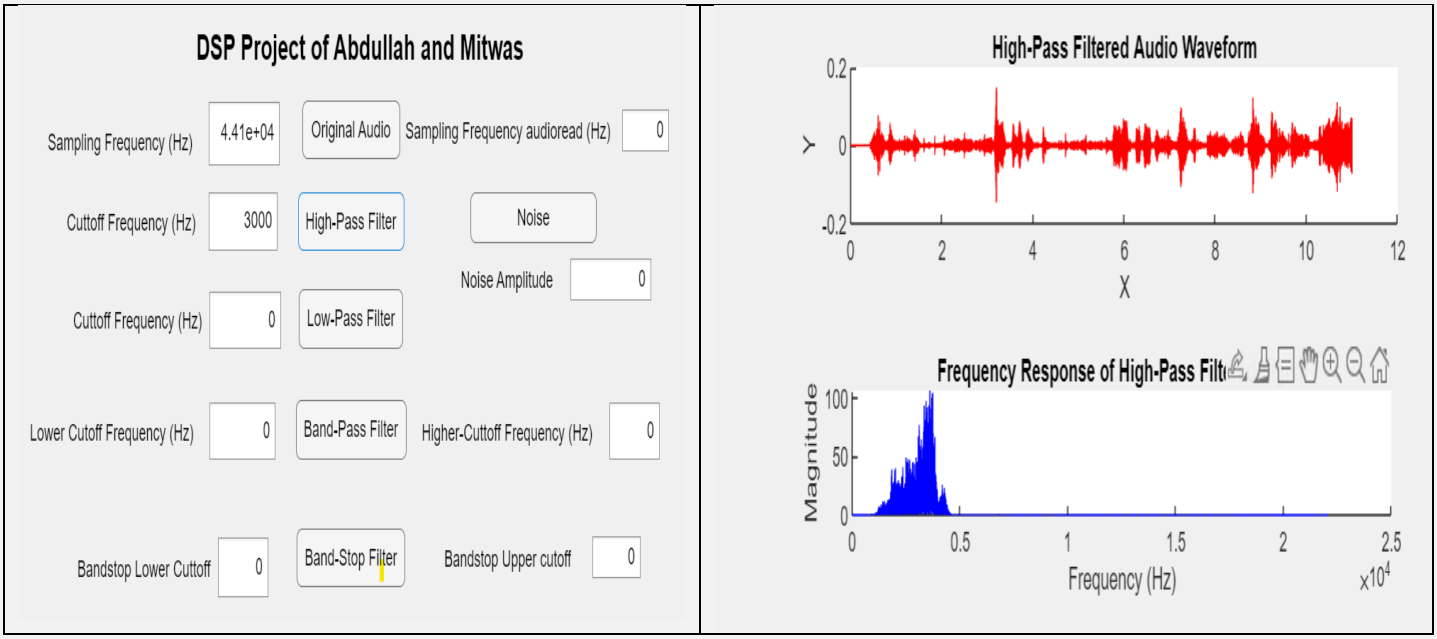


5.2 Audio with noise

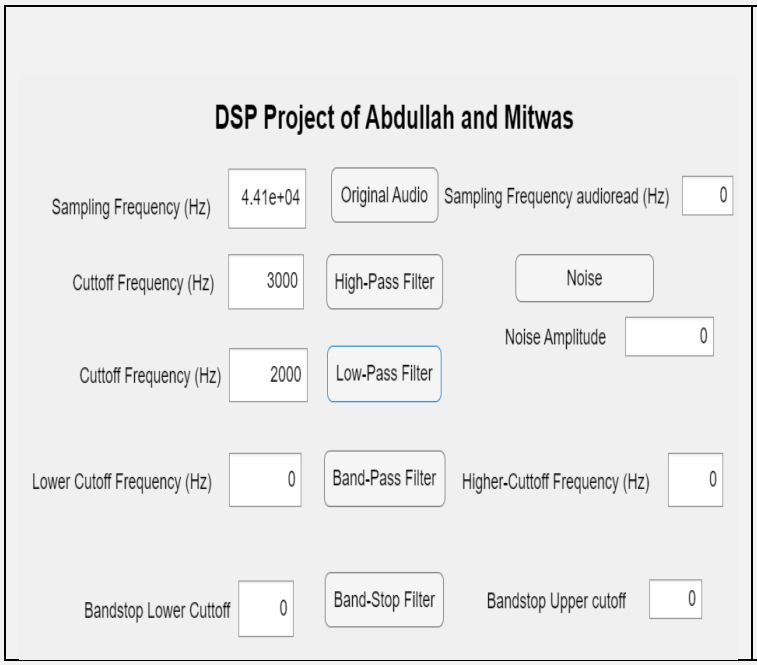


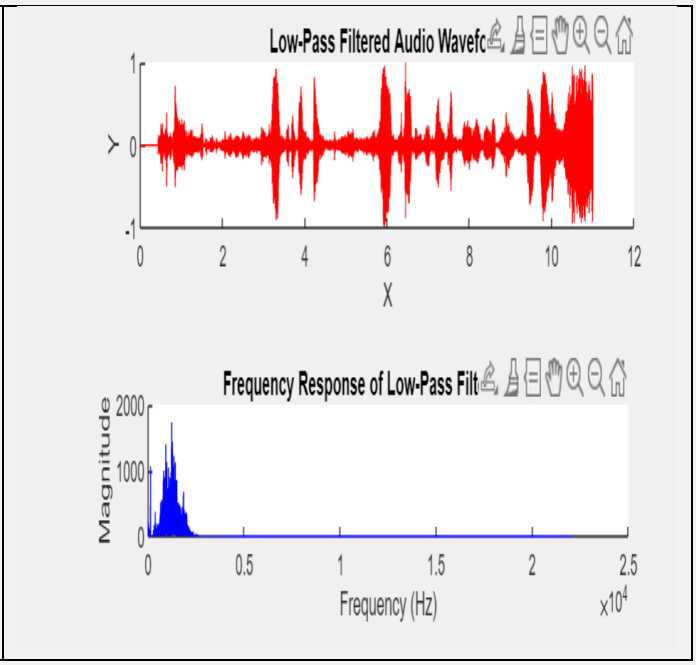
5.4 Applying filters on high noise for better results

- **High Pass Filter**

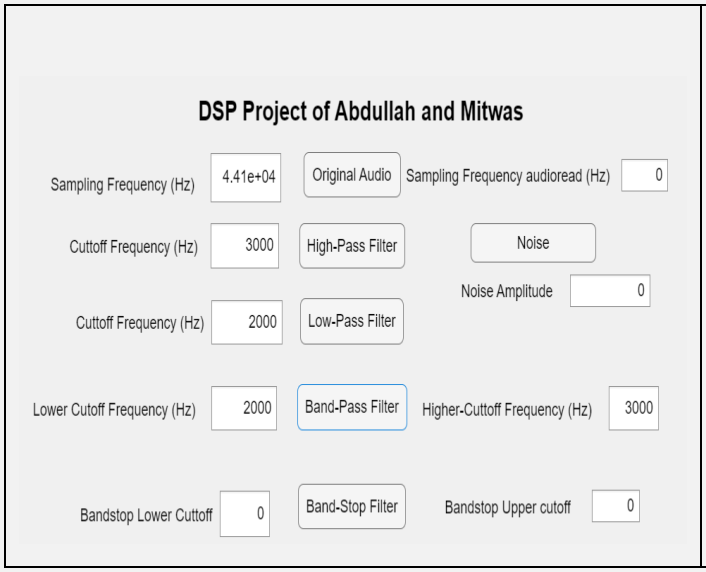


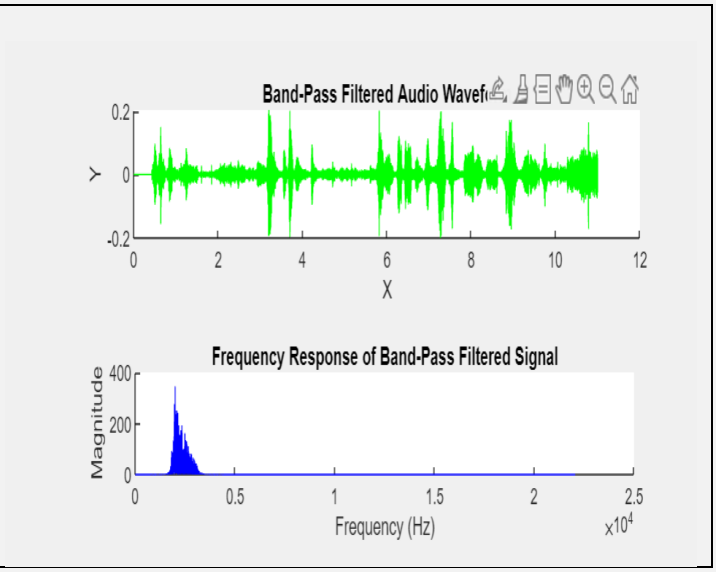
- **Low pass Filter**



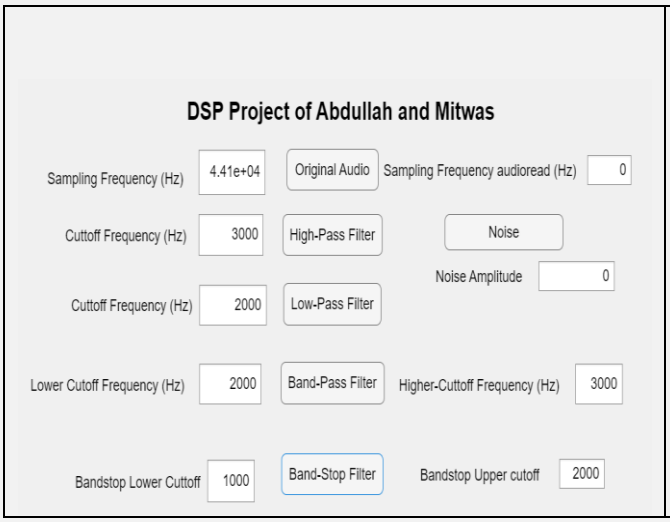


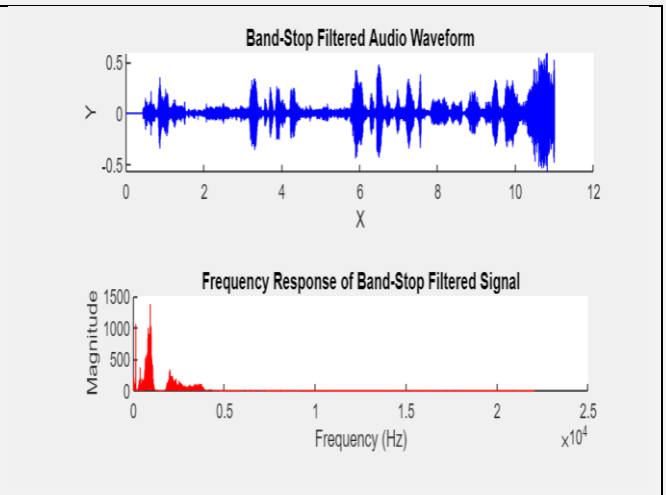
- **Band Pass Filter**





- **Band Stop Filter**





CODE:

```
% Function Declaration
methods (Access = private)

        function y_highpass = applyHighPassFilter(app, y, Fs, cutoffFreq)
        [b, a] = butter(6, cutoffFreq / (Fs / 2), 'high');
        y_highpass = filter(b, a, y);  % Apply the high-pass filter
        end

        function y_lowpass = applyLowPassFilter(app, y, Fs, cutoffFreq)
        [b, a] = butter(6, cutoffFreq / (Fs / 2), 'low');
            y_lowpass = filter(b, a, y);  % Apply the low-pass filter
        end

        function y_bandpass = applyBandPassFilter(app, y, Fs, lowerCutoff,
upperCutoff)
         [b, a] = butter(6, [lowerCutoff, upperCutoff] / (Fs / 2),
'bandpass');
            y_bandpass = filter(b, a, y);  % Apply the band-pass filter
        end


        function y_bandstop = applyBandStopFilter(app,y, Fs, lowerCutoff,
upperCutoff)
            [b, a] = butter(6, [lowerCutoff, upperCutoff] / (Fs / 2),
'stop');
            y_bandstop = filter(b, a, y);  % Apply the band-stop filter
        end

    end
```

```
% Code when Original Audio Button is Pushed
function OriginalAudioButtonPushed(app, event)
            userFs = app.SamplingFrequencyHzEditField.Value;

            % Load the audio file
            [y, Fs] = audioread('Rain + bird.mp3');

            % Resampling with User Sampling Rate
```

```
            y = resample(y, userFs, Fs);

            % Time axis in seconds
            time = (0:length(y)-1) / userFs;

            % Returning Original Sampling rate by audioread function
            app.SamplingFrequencyaudioreadHzEditField.Value = Fs;

            % Create a new figure for plotting the original waveform
            plot(app.UIAxes, time, y);
            app.UIAxes.Title.String = 'Original Audio Waveform';

            % Frequency Response Calculation
            n = length(y);                    % Number of samples
            frequencies = (0:n-1) * (userFs / n);   % Frequency axis in Hz
            magnitudeSpectrum = abs(fft(y));   % Magnitude of FFT

            % Plot Frequency Response
            plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'b'); % Single-sided spectrum
        app.UIAxes2.Title.String = 'Frequency Response of Original Audio';
            app.UIAxes2.XLabel.String = 'Frequency (Hz)';
            app.UIAxes2.YLabel.String = 'Magnitude';

            % Play the original audio at the resampled rate
            sound(y, userFs);
        end
```

```
% When Low Pass Audio Button is Pushed
        function LowPassFilterButtonPushed(app, event)
         % Taking Cutoff Frequency from User
        fclp = app.CuttoffFrequencyHzEditField.Value;

        % Taking Sampling Rate from User
        userFs = app.SamplingFrequencyHzEditField.Value;

        % Load the audio file
        [y, Fs] = audioread('Rain + bird.mp3');

        % Resampling with User Sampling Rate
        y = resample(y, userFs, Fs);

        % Time axis in seconds
        time = (0:length(y)-1) / userFs;

        noiseAmplitude = app.NoiseAmplitudeEditField.Value;
        noise = noiseAmplitude * randn(size(y));  % Gaussian noise

        % Add the noise to the original signal
        y_noisy = y + noise;

        % Apply low-pass filter with cutoff frequency of fclp
        y_lowpass = app.applyLowPassFilter(y_noisy, userFs, fclp);

        % Create a new figure for plotting the waveform
        plot(app.UIAxes, time, y_lowpass, 'r');
```

```
        app.UIAxes.Title.String = 'Low-Pass Filtered Audio Waveform';

        % Frequency Response Calculation
        n = length(y_lowpass);                  % Number of samples
        frequencies = (0:n-1) * (userFs / n);   % Frequency axis in Hz
        magnitudeSpectrum = abs(fft(y_lowpass)); % Magnitude of FFT

        % Plot Frequency Response
        plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'b'); % Single-sided spectrum
        app.UIAxes2.Title.String = 'Frequency Response of Low-Pass Filtered
Signal';
        app.UIAxes2.XLabel.String = 'Frequency (Hz)';
        app.UIAxes2.YLabel.String = 'Magnitude';

        % Play the low-pass filtered audio
        sound(y_lowpass, userFs);

        end
```

```
% When HighPass Filter Button is pushed and values are given
function HighPassFilterButtonPushed(app, event)
            % Taking Cutoff Frequency from User
            fchp = app.CuttoffFrequencyHzEditField_2.Value;

            % Taking Sampling Rate from User
            userFs = app.SamplingFrequencyHzEditField.Value;

            % Load the audio file
            [y, Fs] = audioread('Rain + bird.mp3');

            % Resampling with User Sampling Rate
            y = resample(y, userFs, Fs);

            % Time axis in seconds
            time = (0:length(y)-1) / userFs;

        noiseAmplitude = app.NoiseAmplitudeEditField.Value;
        noise = noiseAmplitude * randn(size(y));  % Gaussian noise

        % Add the noise to the original signal
        y_noisy = y + noise;

            % Apply high-pass filter with cutoff frequency of fchp
            y_highpass = app.applyHighPassFilter(y_noisy, userFs, fchp);

   % Create a new figure for plotting the high-pass filtered waveform
            plot(app.UIAxes, time, y_highpass, 'r');
            app.UIAxes.Title.String = 'High-Pass Filtered Audio Waveform';

            % Frequency Response Calculation for high-pass filtered signal
            n = length(y_highpass);                     % Number of samples
            frequencies = (0:n-1) * (userFs / n);    % Frequency axis in Hz
            magnitudeSpectrum = abs(fft(y_highpass));  % Magnitude of FFT

            % Plot Frequency Response on UIAxes2
```

```
            plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'b');  % Single-sided spectrum
            app.UIAxes2.Title.String = 'Frequency Response of High-Pass
Filtered Signal';
            app.UIAxes2.XLabel.String = 'Frequency (Hz)';
            app.UIAxes2.YLabel.String = 'Magnitude';

            % Play the high-pass filtered audio
            sound(y_highpass, userFs);

        end
```

```
% When Band Pass Filter Button is pushed and values are given
function BandPassFilterButtonPushed(app, event)
    % Taking Cutoff Frequencies from User
        lowerCutoff = app.LowerCutoffFrequencyHzEditField.Value;
        upperCutoff = app.HigherCuttoffFrequencyHzEditField.Value;

        % Taking Sampling Rate from User
        userFs = app.SamplingFrequencyHzEditField.Value;

        % Load the audio file
        [y, Fs] = audioread('Rain + bird.mp3');

        % Resampling with User Sampling Rate
        y = resample(y, userFs, Fs);

        % Time axis in seconds
        time = (0:length(y)-1) / userFs;

        noiseAmplitude = app.NoiseAmplitudeEditField.Value;
        noise = noiseAmplitude * randn(size(y));  % Gaussian noise

        % Add the noise to the original signal
        y_noisy = y + noise;

        % Apply band-pass filter
        y_bandpass = app.applyBandPassFilter(y_noisy, userFs, lowerCutoff,
upperCutoff);

        % Create a new figure for plotting the band-pass filtered waveform
        plot(app.UIAxes, time, y_bandpass, 'g');
        app.UIAxes.Title.String = 'Band-Pass Filtered Audio Waveform';

        % Frequency Response Calculation for band-pass filtered signal
        n = length(y_bandpass);                          % Number of samples
        frequencies = (0:n-1) * (userFs / n);            % Frequency axis in Hz
        magnitudeSpectrum = abs(fft(y_bandpass));        % Magnitude of FFT

        % Plot Frequency Response on UIAxes2
        plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'b');  % Single-sided spectrum
        app.UIAxes2.Title.String = 'Frequency Response of Band-Pass Filtered
Signal';
        app.UIAxes2.XLabel.String = 'Frequency (Hz)';
        app.UIAxes2.YLabel.String = 'Magnitude';
```

```
        sound(y_bandpass, userFs);
        end
```

```
% When Band Stop Filter Button is pushed and values are given
function BandStopFilterButtonPushed(app, event)
    % Taking Cutoff Frequencies from User
        lowerCutoff = app.BandstopLowerCuttoffEditField.Value;
        upperCutoff = app.BandstopUppercutoffEditField.Value;

        % Taking Sampling Rate from User
        userFs = app.SamplingFrequencyHzEditField.Value;

        % Load the audio file
        [y, Fs] = audioread('Rain + bird.mp3');

        % Resampling with User Sampling Rate
        y = resample(y, userFs, Fs);

        noiseAmplitude = app.NoiseAmplitudeEditField.Value;
        noise = noiseAmplitude * randn(size(y));  % Gaussian noise

        % Add the noise to the original signal
        y_noisy = y + noise;

        % Apply band-stop filter
        y_bandstop = app.applyBandStopFilter(y_noisy, userFs, lowerCutoff,
upperCutoff);

        % Time axis in seconds
        time = (0:length(y)-1) / userFs;

        % Create a new figure for plotting the band-stop filtered waveform
        plot(app.UIAxes, time, y_bandstop, 'b');
        app.UIAxes.Title.String = 'Band-Stop Filtered Audio Waveform';

        % Frequency Response Calculation for band-stop filtered signal
        n = length(y_bandstop);                          % Number of samples
        frequencies = (0:n-1) * (userFs / n);        % Frequency axis in Hz
        magnitudeSpectrum = abs(fft(y_bandstop));      % Magnitude of FFT

        % Plot Frequency Response on UIAxes2
        plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'r');  % Single-sided spectrum
        app.UIAxes2.Title.String = 'Frequency Response of Band-Stop Filtered
Signal';
        app.UIAxes2.XLabel.String = 'Frequency (Hz)';
        app.UIAxes2.YLabel.String = 'Magnitude';

        % Play the band-stop filtered audio
        sound(y_bandstop, userFs);
        end
```

```
% When Noise Button is Pushed and value is given in Noise Amplitude
function NoiseButtonPushed(app, event)
            % Taking Sampling Rate from User
```

```
    userFs = app.SamplingFrequencyHzEditField.Value;

    % Load the original audio file
    [y, Fs] = audioread('Rain + bird.mp3');

    % Resample the audio to the user-defined sampling rate
    y = resample(y, userFs, Fs);

    % Time axis for plotting the signal
    time = (0:length(y)-1) / userFs;

    % Generate random noise (Gaussian white noise)
    noiseAmplitude = app.NoiseAmplitudeEditField.Value;
    noise = noiseAmplitude * randn(size(y));  % Gaussian noise

    % Add the noise to the original signal
    y_noisy = y + noise;

    % Create a new figure for plotting the noisy signal's waveform
    plot(app.UIAxes, time, y_noisy, 'r');
    app.UIAxes.Title.String = 'Noisy Signal Waveform';
    app.UIAxes.XLabel.String = 'Time (s)';
    app.UIAxes.YLabel.String = 'Amplitude';

    % Frequency Response Calculation for the noisy signal
    n = length(y_noisy);                        % Number of samples
    frequencies = (0:n-1) * (userFs / n);       % Frequency axis in Hz
    magnitudeSpectrum = abs(fft(y_noisy));      % Magnitude of FFT

    % Plot Frequency Response of the noisy signal
    plot(app.UIAxes2, frequencies(1:floor(n/2)),
magnitudeSpectrum(1:floor(n/2)), 'b');
    app.UIAxes2.Title.String = 'Magnitude Response of Noisy Signal';
    app.UIAxes2.XLabel.String = 'Frequency (Hz)';
    app.UIAxes2.YLabel.String = 'Magnitude';

    % Play the noisy audio
    sound(y_noisy, userFs);


        end
```

6. Conclusion

The project successfully demonstrated MATLAB's capabilities in music signal processing. The system effectively removed noise, isolated musical instruments, and provided an interactive platform for user engagement. Future enhancements could include machine learning-based classification of musical instruments.