ASIA PACIFIC UNIVERSITY TECHNOLOGY & INNOVATION

EE004-4-3-DIGITAL SIGNAL PROCESSING

INDIVIDUAL ASSIGNMENT

Student Name	**:**	Pedro Fabian Owono Ondo Mangue
Student ID no.	**:**	TP063251
Intake	**:**	APU3F2308CE
Lecturer Name	**:**	Dr. Mukil Alagirisamy
Submission Date	**:**	15th May 2024

Table of Contents

List of Figures

INTRODUCTION

The subsequent paper highlights the design and implementation of a MATLAB App Designer-based speech enhancement application. The project primarily focused on how to amalgamate a friendly GUI with a few state-of-the-art algorithms that are used for the enhancement of audio and for filtering noises. The basic goal of this application is to remove any sort of noises or disturbance from the audio signals. Gigantic research has been carried out to study and find the desired noises and filters that are to be inculcated within the application. Finally, four different filters and four forms of noise were selected for the purpose after a critical evaluation.

The user can record or import an audio file to be analysed. The recorded or imported audio may be noise-free or noisy. The user may then use synthetic noise to a clean signal to simulate noise under various audio environments. The user may then use the introduced filters for attempting noise removal. For an enriched interaction, buttons, different tabs to select the graphs in time and frequency domain, a button where the user can check for the workflow or instructions on how to use the whole program. This allows a user to change some parameters such as cutoff frequency, filter order, and window size. The application includes several important features that allow optimization of use of the filters and make the user experience richer. The features will be discussed in the following sections on design formulation and innovativeness. The filters inbuilt in this application are Kalman Filter, Spectral Subtraction Filter, Chebyshev Filter, and Butterworth Filter. The selected noises include Brownian, White, Gaussian, and Pink Noise. The goal of this project is to see exactly how each filter functions to remove each form of noise. Let's now describe the theoretical concepts to gain a better view of these filters.

Kalman Filter

The Kalman Filter is basically a very powerful algorithmic tool invented by Rudolf E. Kalman in the early 1960s and widely used for the estimation of variables, amidst much uncertainty and great randomness. It works recursively, updating the estimation at every step with new observations further to reduce error and present very fine results, best for application in dynamic systems.

Theoretical Foundations Kalman Filter is an application of linear quadratic estimation. It is a two-step processing of a filter that involves prediction and correction. The filter predicts the state of the system at the next time step, then updates the prediction through new measurements. Mathematically, it functions through a series of equations:

Prediction Equations:

- **State Prediction:** $\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$
- **Covariance Prediction:** $P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$

Figure 1: Prediction Equations for Kalman Filter

Correction Equations:

- **Kalman Gain:** $K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}$
- **State Update:** $\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H_k \hat{x}_{k|k-1})$
- **Covariance Update:** $P_{k|k} = (I - K_k H_k) P_{k|k-1}$

Figure 2: Correction Equations for Kalman Filter

In that equation the some of the parameters are classified in the next way:

F_k = State transition

B_k= Control input model

U_k= Control vector

Q_k= Process noise covariance matrix

K_k= Kalman gain

H_k= Observation model

Z_k= Actual measurement

R_k = Measure of covariance

Spectral Subtraction Filter

A common method for reducing noise in audio signal processing is called spectral subtraction, which improves speech clarity in noisy settings. Boll first presented this technique in his groundbreaking 1979 paper, "Suppression of Acoustic Noise in Speech Using Spectral Subtraction". The method's foundation is the idea that the underlying clean speech signal can be recovered by deducting an estimate of the noise spectrum from the noisy speech spectrum.

The basic idea behind spectral subtraction is based on the supposition that speech and noise signals are additive, and that it is possible to estimate the characteristics of noise when there is little or no speech. Usually, the procedure entails the following crucial steps:

Noise Estimation: The power spectrum of the noise is estimated during non-speech segments, which are segments in which there is only noise.

To estimate the clean speech spectrum, the estimated noise spectrum is subtracted from the noisy speech spectrum. Mathematically. If Y(f) is the spectrum of noise speech and N(f) the estimate noise spectrum, the clean speech of the spectrum S(f) is given by:

$$S(f) = Y(f) - \alpha N(f)$$

Figure 3: First Spectral Subtraction Equation

Spectral Magnitude Adjustment: Half-wave rectification and other techniques are frequently used to modify the spectral values to prevent the creation of negative spectral components, which are not physically possible.

$$S'(f) = \max(S(f), 0)$$

Figure 4: Second Spectral Equation

Phase Reconstruction: Since phase estimation is more difficult and less important for speech intelligibility, the phase of the noisy speech is typically preserved.

Time-Domain Signal Reconstruction: Using the inverse Fourier transform, the final clean speech signal is reconstructed from the modified spectral magnitude and the original phase.

IIR Butterworth Filter

A class of filters called Infinite Impulse Response (IIR) Butterworth Filters is intended to produce a passband with an extremely flat frequency response. A popular choice for signal processing applications, Butterworth filters are known for their ease of use and efficiency. They are named after British engineer Stephen Butterworth, who originally detailed this kind of filter in his 1930 paper "On the Theory of Filter Amplifiers".

Butterworth filters are made to have the smoothest roll-off towards zero in the stopband and the best possible passband sensitivity, which means there are no ripples in the frequency response within the passband. By making sure that the magnitude response squared of the filter is inversely proportional to the total squares of the frequency and the cutoff frequency, this characteristic can be attained.

The order nn of a Butterworth filter, which controls the roll-off's steepness and, consequently, the filter's capacity to discriminate between possible noise and desired signal frequencies, defines the filter's general form. An n-th order Butterworth low-pass filter's frequency response is provided by:

$$H(s) = \frac{1}{1 + (\frac{s}{\omega_c})^{2n}}$$

Figure 5: Equation from Butterworth Filter

Here s is the complex number while w_c is the cutoff frequency.

The process of designing IIR Butterworth filters entails identifying the filter's poles, which are symmetrically positioned about the real axis and uniformly distributed around a circle in the left half of the s-plane. The flat passband response can be achieved with the aid of this symmetry.

Chebyshev Filter

Review of the Chebyshev Filter's Extensive Literature Overview

Compared to other filters like Butterworth filters, Chebyshev filters have a faster roll-off and more passband ripple (Type I) or stopband ripple (Type II). The filter family is named after the Russian mathematician Pafnuty Chebyshev. They are very useful in applications that need sharp frequency cutoffs because they offer an efficient way to achieve a given filter specification with a lower order of the filter.

The distinctive response characteristic of Chebyshev filters is their trade-off between increased roll-off steepness and frequency response monotonicity. This is accomplished by permitting ripples in the Type I passband and the Type II stopband, but not in both. The behaviour of the filter is described mathematically using Chebyshev polynomials.

Type I Chebyshev Filters: The Type I filter preserves a monotonic (non-rippling) response in the stopband while permitting ripple in the passband. A Type I Chebyshev low-pass filter's magnitude response is represented by the following expression:

$$|H(j\omega)| = \frac{1}{\sqrt{1 + \epsilon^2 T_n^2 \left(\frac{\omega}{\omega_c}\right)}}$$

Figure 6: Equation from Chebyshev

In the figure above we have three important parameters which are Tn (nth order Chebyshev polynomial), and the W_C (cutoff frequency), and E^2 (triple factor which controls the amplitude of the ripples in the passband)

Type II Chebyshev Filters: On the other hand, the Type II filter permits ripples in the stopband and has a monotonic passband. Since the Chebyshev polynomials used are inverse, designing its magnitude response is typically more difficult.

Brownian Noise

One of the noises commonly associated with Brownian motion of particles is Brownian noise, also called red noise. In this case, the power density increases at lower frequencies. A PSD that is like $1/f^2$ characterizes it. The physical phenomenon of particles moving at random is the basis

of the Brownian noise model. Consequently, its spectral density profile, like the physical Brownian motion, is most dense at lower frequencies, reflecting the dominance of larger timescales in this process.

White Noise

Background Noise Summary When the PSD is flat, meaning that all the components surrounding each frequency component have the same power, we say that the noise is white noise. Most importantly, it serves as a benchmark in numerous applications due to its simplicity from a statistical perspective, wherein all frequencies are equally probable. A further definition of white noise is noise that is uniform across all frequencies; it has a constant hissing sound. The standard representation in digital signal processing is that of a discrete signal, where each sample is a Gaussian random variable.

Gaussian Noise

Overview of Gaussian Noise Any kind of noise whose probability density function is assumed to follow a normal distribution is referred to as Gaussian noise. Because of the Central Limit Theorem, which asserts that regardless of the initial distribution of the variables, every sum of independent random variables has a sum that is almost normally distributed, this type of noise is the most common. Although the initial distributions of the variables can be any other, it is reasonable to assume that many independent random variables added together will have a distribution close to that of a Gaussian distribution. The amplitudes will follow a bell-shaped distribution with a zero-centered mean; the variance will determine the power.

Pink Noise

A signal where the power spectral density drops as a function of frequency is called pink noise, or 1/f noise. When compared to white noise, pink noise sounds much deeper because it is noisier at lower frequencies. Financial markets, earthquake data, and biological systems are just a few examples of the many complex systems and natural processes that are thought to rely on this property. Many scientific and engineering fields make extensive use of PSD of pink noise due to its 1/f law.

FUNCTIONALITIES OF MY AUDIO PROCESSING APPLICATION

CODING:

Loading button

Functionality: Allow user to load an existing audio recording from local files in wav format. Once the audio gets selected, it will automatically be shown in the graph for the original audio recording.

```
% Button pushed function: LoadButton
function LoadButtonPushed(app, event)
    [file, path] = uigetfile('*.wav', 'Select an audio file');
    if isequal(file, 0) || isequal(path, 0)
        uialert(app.UIFigure, 'No file was selected.', 'Load Error');
    else
        app.originalAudio = audioread(fullfile(path, file));
        plot(app.OriginalAxes, (0:length(app.originalAudio)-1)/app.Fs, app.originalAudio);
        title(app.OriginalAxes, 'Original Audio');
        app.AudioLoaded = true;
        uialert(app.UIFigure, 'Audio file loaded successfully!', 'Load Success');
        app.updateFrequencyPlot(app.originalAudio, app.OriginalAxes_Frequency);  % Update frequency plot
    end
```

Figure 7: Load button

Save button

Functionality: For the save button, the user will basically save the recording after recording. You can appreciate the validations implemented in the code such as the user cannot press the button without recording or loading the audio.

```
% Button pushed function: SaveButton
function SaveButtonPushed(app, event)
    if ~app.AudioLoaded
        uialert(app.UIFigure, 'No audio to save. Please load or record audio first.', 'Save Error');
    else
        [file, path] = uiputfile('*.wav', 'Save the audio file');
        if isequal(file, 0) || isequal(path, 0)
            uialert(app.UIFigure, 'File save canceled.', 'Save Cancelled');
        else
            audiowrite(fullfile(path, file), app.originalAudio, app.Fs);
            uialert(app.UIFigure, 'Audio saved successfully!', 'Save Success');
        end
    end
```

Figure 8: Save button

Recording button

Functionality: The user will be able to record by using the microphone.

```
% Button pushed function: RecordButton
function RecordButtonPushed(app, event)
     % Ensure the recorder is not already active
    if ~isempty(app.recorder) && isrecording(app.recorder)
        uialert(app.UIFigure, 'Recording is already in progress.', 'Recording Error');
        return;
    end

    % Initialize the recorder
    try
        app.recorder = audiorecorder(app.Fs, 16, 1);
        uialert(app.UIFigure, 'Recording has started. Please speak into the microphone.', 'Recording Started');
        recordblocking(app.recorder, 6);

        % Retrieve audio data
        app.originalAudio = getaudiodata(app.recorder);
        if isempty(app.originalAudio)
            uialert(app.UIFigure, 'No audio recorded. Please try again.', 'Recording Error');
            return;
        end

        % Plot the recorded audio
        plot(app.OriginalAxes, (0:length(app.originalAudio)-1)/app.Fs, app.originalAudio);
        title(app.OriginalAxes, 'Original Audio');
        app.AudioLoaded = true;
        uialert(app.UIFigure, 'Recording complete.', 'Record Success');
        app.updateFrequencyPlot(app.originalAudio, app.OriginalAxes_Frequency);

    catch ME
        uialert(app.UIFigure, ['Failed to record audio: ', ME.message], 'Recording Error');
    end
```

Figure 9:Recording button

Play buttons: Play Original, play Noise, play Filtered.

Functionality: Once the user has already recorded or load an audio, there is an option for the user to play the audio which can be the input audio or the original one, as it can be the audio with the noise, or the audio filtered.

```
% Button pushed function: PlayOriginalButton
function PlayOriginalButtonPushed(app, event)
    % Button pushed function: PlayOriginalButton

    if ~app.AudioLoaded
        uialert(app.UIFigure, 'No original audio loaded. Please load or record audio first.', 'Playback Error');
    else
        sound(app.originalAudio, app.Fs);  % Play the original audio
        uialert(app.UIFigure, 'Playing original audio.', 'Playback Started');
    end

end

% Button pushed function: PlayNoisyButton
function PlayNoisyButtonPushed(app, event)
    if isempty(app.noisyAudio)
        uialert(app.UIFigure, 'No noisy audio available. Add noise first.', 'Playback Error');
    elseif isempty(app.currentNoise) || strcmp(app.currentNoise, 'None')
        uialert(app.UIFigure, 'Please select and add noise type before playback.', 'Playback Error');
    else
        sound(app.noisyAudio, app.Fs);  % Play the noisy audio
        uialert(app.UIFigure, 'Playing noisy audio.', 'Playback Started');
    end
end

% Button pushed function: PlayFilteredButton
function PlayFilteredButtonPushed(app, event)
    if isempty(app.filteredAudio)
        uialert(app.UIFigure, 'No filtered audio available. Apply filter first.', 'Playback Error');
    else
        sound(app.filteredAudio, app.Fs);  % Play the filtered audio
        uialert(app.UIFigure, 'Playing filtered audio.', 'Playback Started');
    end
end
```

Figure 10:Play buttons

Restart button

Functionality: By pressing the restart button, the user will be able to go back to the initial state of the program as everything will be clear up or changed to the initial state.

```
% Button pushed function: RestartButton
function RestartButtonPushed(app, event)
    % Check if there is any audio data loaded
    if app.AudioLoaded
        % Reset all audio data and states
        app.originalAudio = [];
        app.noisyAudio = [];
        app.filteredAudio = [];
        app.AudioLoaded = false;

        % Clear all plots
        cla(app.OriginalAxes);
        cla(app.NoiseAxes);
        cla(app.FilteredAxes);
        title(app.OriginalAxes, 'Original Audio');
        title(app.NoiseAxes, 'Noise Audio');
        title(app.FilteredAxes, 'Filtered Audio');

        % Notify user
        uialert(app.UIFigure, 'Application state has been reset.', 'Restart Successful');
    else
        % Notify user that there is nothing to reset
        uialert(app.UIFigure, 'No audio loaded or changes to reset.', 'No Action Needed');
    end
```

Figure 11: Restart Button

Noise and Filter connection:

In my MATLAB, there are two primary ways to manage the connection between noise addition and filtering: the Dropdown ValueChanged functions and a way to check if a filter is appropriate depending on the kind of noise used. In the section that follows, I explain the interplay between these features and how they work together to apply the correct filters to each kind of noise.

Processes for Linking Filters and Noise
Causing Discord:
When the user selects a noise type from the dropdown, the method called AddNoiseButtonDropdownValueChanged handles the input. This method adds the chosen noise type to the audio by calling applyNoise.

```
% Value changed function: AddNoiseButtonDropdown
function AddNoiseButtonDropdownValueChanged(app, event)
    noiseType = app.AddNoiseButtonDropdown.Value;
    if ~app.AudioLoaded
        uialert(app.UIFigure, 'Load or record audio before adding noise.', 'Operation Error');
    else
        app.applyNoise(noiseType);  % Call the encapsulated method
    end
end
```

Figure 12: Noise Dropdown Menu

When a user selects a filter type, the program handles their input using the ApplyFilterButtonDropDownValueChanged method. A comparison between the current noise

type and the required noise type for each filter is performed by the matchesNoise function. If the filters are suitable for the noise, the selected filter is checked.

```
% Button pushed function: PlayFilteredButton
function PlayFilteredButtonPushed(app, event)
    if isempty(app.filteredAudio)
        uialert(app.UIFigure, 'No filtered audio available. Apply filter first.', 'Playback Error');
    else
        sound(app.filteredAudio, app.Fs);  % Play the filtered audio
        uialert(app.UIFigure, 'Playing filtered audio.', 'Playback Started');
    end
end
```

Figure 13:Apply Filter button

To make sure the right filter is applied to each type of noise, the matchesNoise method establishes a relationship between different types of noise and filters. To validate user choices and prevent incorrect filter application, this method is an essential part of the control flow.

```
    % Helper function to match noise and filter type
function isMatch = matchesNoise(app, filterType)
    switch filterType
        case 'Kalman'
            isMatch = strcmp(app.currentNoise, 'Brownian');
        case 'Spectral Subtraction'
            isMatch = strcmp(app.currentNoise, 'White');
        case 'Chebyshev'
            isMatch = strcmp(app.currentNoise, 'Gaussian');
        case 'Butterworth'
            isMatch = strcmp(app.currentNoise, 'Pink');
        otherwise
            isMatch = false;
    end
end
```

Figure 14: Link Method for Noise and Filter

A user can click the AddNoiseButtonDropdown, select the type of noise, for example Gaussian, which upon selection calls the function AddNoiseButtonDropdownValueChanged, and in sequence adds the Gaussian noise to the audio.

Note that when the user selects a filter from ApplyFilterButtonDropDown, which, in this case, is Chebyshev, the ApplyFilterButtonDropDownValueChanged function calls this function and then the matchesNoise function if the type of noise satisfies the filter at the first instance. For this example, if the requirements include a Chebyshev filter with Gaussian noise, the applyFilter function will be called with the selected filter applied on the audio. These techniques are intertwined and guarantee the correctness of the application logic, which gives assurance that the expected results will occur from the user's interactions. All of this makes the audio processing application more usable and effective.

Noise Function:

Application Code for Noise

An audio signal is enhanced with various forms of noise by the applyNoise function. Priority one is checking for loaded audio. If not, an error message appears. The noise is generated and applied to the audio based on the chosen noise type (Brownian, White, Gaussian, Pink).

```
% Utility function to apply noise
function applyNoise(app, noiseType)
    if ~app.AudioLoaded
        uialert(app.UIFigure, 'Load or record audio before adding noise.', 'Operation Error');
        return;
    end

    switch noiseType
        case 'Brownian'
            brownNoise = cumsum(randn(size(app.originalAudio)));  % Generate Brownian noise
            brownNoise = (brownNoise - mean(brownNoise)) / std(brownNoise);  % Normalize
            noise = 0.03 * brownNoise;  % Scale the noise
        case 'White'
            noise = 0.07 * rand(size(app.originalAudio));
        case 'Gaussian'
            noise = 0.03 * randn(size(app.originalAudio));
        case 'Pink'
            noise = 0.04 * filter(1, [1 -0.999], randn(size(app.originalAudio)));
        otherwise
            noise = zeros(size(app.originalAudio));
    end
```

Figure 15: Function Apply Noise Implementation

Function Apply Filter

Application Code Filtering

The noisy audio signal is filtered by the applyFilter function. It confirms that noise has been added and that the audio has loaded. Before applying a proper filter, it verifies that the chosen filter matches the type of noise present at that moment. It sends out a warning if the requirements are not met.

```
% Utility function to apply filter
function applyFilter(app, filterType)
    if ~app.AudioLoaded || isempty(app.noisyAudio)
        uialert(app.UIFigure, 'Add noise before applying a filter.', 'Operation Error');
        return;
    end


    % Debugging line to show current noise and selected filter type
    uialert(app.UIFigure, ['Current Noise: ', app.currentNoise, ', Selected Filter: ', filterType], 'Debug Info');

    % Check if the selected filter matches the current noise
    if ~app.matchesNoise(filterType)
        uialert(app.UIFigure, ['Please select the correct filter for ', app.currentNoise, ' noise.'], 'Filter Error');
        return;
    end

    switch filterType
        case 'Kalman'
            app.filteredAudio = kalmanFilter(app, app.noisyAudio);
        case 'Spectral Subtraction'
            app.filteredAudio = SpectralSubtraction(app, app.noisyAudio, app.Fs);
        case 'Chebyshev'
            app.filteredAudio = applyChebyshevFilter(app, app.noisyAudio, app.Fs);
        case 'Butterworth'
            app.filteredAudio = applyButterworthFilter(app, app.noisyAudio, app.Fs);
        otherwise
            app.filteredAudio = [];
            uialert(app.UIFigure, 'Select a valid filter.', 'Filter Error');
            return;
    end
```

Figure 16: Function Apply Filter Implementation

FILTER TYPE IMPLEMENTATION CODE:

Kalman Filter

A series of noisy measurements are used to estimate the state of a linear dynamic system using the Kalman filter, which is an efficient recursive filter.

The Kalman filter makes use of an initial estimate to make a prediction about the subsequent state. It then adjusts itself based on the measurement noise and the process noise, thereby gradually improving the estimate.

```
% Kalman filter implementation
function output = kalmanFilter(~, input)
    xEst = input(1);
    P = 1;
    Q = 0.05;
    R = 0.5;

    output = zeros(size(input));
    output(1) = xEst;

    for k = 2:length(input)
        xPred = xEst;
        PPred = P + Q;
        K = PPred / (PPred + R);
        xEst = xPred + K * (input(k) - xPred);
        P = (1 - K) * PPred;
        output(k) = xEst;
    end
end
```

Figure 17: Kalman Filter code

Spectral Subtraction

To improve speech quality by lowering background noise, spectral subtraction is utilized. Estimating the power spectral density of noise and then subtracting that value from the spectrum of the noisy signal is how it brings about the desired effect.

Through the process of subtracting the estimated noise from the spectral components of the noisy signal, this method can perform noise reduction.

```
nction output = SpectralSubtraction(~, noisyAudio, fs)
    frameLen = 256;
    overlap = frameLen / 2;
    win = hamming(frameLen);
    noiseFrames = floor(0.5 * fs / frameLen);
    noisePSD = zeros(frameLen, 1);

    for k = 1:noiseFrames
        noiseSegment = noisyAudio((k-1)*frameLen+1 : k*frameLen) .* win;
        noisePSD = noisePSD + abs(fft(noiseSegment));
    end
    noisePSD = noisePSD / noiseFrames;

    numFrames = floor(length(noisyAudio) / overlap) - 1;
    output = zeros(length(noisyAudio), 1);

    for k = 1:numFrames
        startIdx = (k-1) * overlap + 1;
        endIdx = startIdx + frameLen - 1;
        segment = noisyAudio(startIdx:endIdx) .* win;
        segmentFFT = fft(segment);
        subtractedSpectrum = max(abs(segmentFFT) - sqrt(noisePSD), 0) .* exp(1i * angle(segmentFFT));
        reconstructedSegment = real(ifft(subtractedSpectrum)) .* win;
        output(startIdx:endIdx) = output(startIdx:endIdx) + reconstructedSegment;
    end
end
```

Figure 18: Spectral Subtraction code

Chebyshev

Chebyshev filters are a type of infrared (IIR) filter that differ from Butterworth filters in that they have a roll-off that is steeper and more passband ripple. To achieve a more precise frequency selection, the implementation cascades through three stages.

By utilizing a combination of low and high pass filters, this filter can effectively reduce the volume of frequencies that fall outside of its designated passband.

```
% Chebyshev filter implementation
function output = applyChebyshevFilter(~, noisyAudio, ~)
    [b1, a1] = cheby1(4, 1, 0.1, 'low');
    stage1Output = filter(b1, a1, noisyAudio);

    [b2, a2] = cheby1(2, 0.5, 0.05, 'low');
    stage2Output = filter(b2, a2, stage1Output);

    [b3, a3] = cheby1(2, 0.5, 0.01, 'high');
    output = filter(b3, a3, stage2Output);
end
```

Figure 19: Chebyshev Code

Butterworth

A characteristic of Butterworth filters is that they have a frequency response that is flat within the passband. The implementation offers a bandpass filter to the implementation.

Keeping a particular frequency range while rejecting others is the purpose of this filter, which is implemented here as a bandpass filter. Its purpose is to minimize the distortion that occurs within the passband.

```
% Butterworth filter implementation
function output = applyButterworthFilter(~, noisyAudio, ~)
    N = 6;  % Order of the filter
    Wn = [0.05, 0.3];  % Normalized frequency band
    [b, a] = butter(N, Wn, 'bandpass');
    output = filtfilt(b, a, noisyAudio);
end
```

Figure 20: Butterworth Code

GUI OVERVIEW AND INNOVATIONS:

The designed "Audio Processing Application for Speech Enhancement" GUI has a complete, user-friendly interface where users interact with audio data for both processing and enhancement. Brief description of the functionalities and the layout of this application:

Key Features

Open, Save, Record: Users can be able to open files saved on their system, save processed files, and record audio directly on the application.

Playback Controls: The three provided playback controls are PlayOriginal, PlayNoisy, and **PlayFiltered**. Using these, the user can listen to the original, the one with added noise, and the filtered sounds to acquire immediate audio feedback on what processing technique to use.

Noise and Filter Application: The interface will have some dropdown menus, namely: AddNoiseButton and ApplyFilterButton. These will provide an option for users on what type of noise or what type of filter, respectively, they want to introduce to the audio. Options here probably include various noises—say, White or Gaussian—and filters like Kalman and Butterworth, discussed in a couple of previous interactions.

Control Buttons: There are other useful buttons as well, such as the Restart button, which resets the application, and the Exit button, which closes the application.

Help, Export Status, Tutorial: These should include the buttons most likely to be used for showing user help, export processing logs or results, and on-the-fly tutorials regarding how to effectively use the application.

Theme Toggler: A dropdown that lets users switch their GUI themes to make the user experience and accessibility more fun.

The graphical user interface consists of tabs that represent the audio signal in both the time domain and the frequency domain:

Time Domain: Plots of these graphs are likely done on Tab1 with a display of the amplitude of the audio signal with respect to time.

Frequency Domain: The plots on Tab2 show a representation of the audio spectrum in amplitude over several frequencies given in decibels. This aids in the analysis of the audio's spectral properties and how effective the applied noise and filters are.

Innovations

Now, the list of the **innovations** included in the GUI are:

1. **Tabbed Interface for Time and Frequency Domain View**

Innovation: It has an integrated tab interface to show the audio signals in both time and frequency domains. This is very important in detailed audio analysis, where a clear view of the effect of noises and filters on the signal is to be seen in both time and frequency domains simultaneously. By combining both views in a tabbed format window, the user experience becomes more straightforward, and workflow efficiency is boosted in the process. That way, a user will be able to easily compare both versions of an original, noisy, and filtered signal in both the time and frequency domains without ever leaving the main interface.

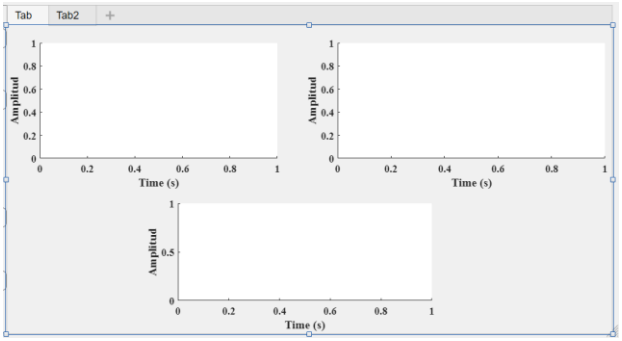

Figure 21: Tab 1 and Tab 2

2. **Real-time audio manipulative with control interactivity Innovation**

It has the interactive controls to add, for example, different noise and filters just directly through its dropdowns 'AddNoiseButton' and 'ApplyFilterButton'. The system allows for manipulation in real-time, thus offering real-time auditory feedback. These features are important for effective real-time audio editing and enhancement. All users, from novice to expert, will find this feature to significantly increase the practical learning curve: they can just try and learn the effects of various acoustic manipulations without processing delays.

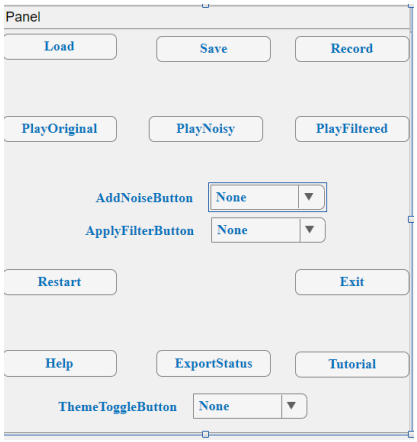

Figure 22:Pannel with buttons

3. **Integrated Help and Tutorial Features**

Inclusion of `Help' and `Tutorial' buttons on the main interface indicates the application to provide on board guidance and tutor screens. This can be advantageous for educational and self-learning purposes.

The embedded educational tools allow users to learn the usage of the software without reaching out for additional resources. This feature enhances user autonomy and can greatly boost the user's capacity to make use of software features effectively, thereby increasing user retention and satisfaction.

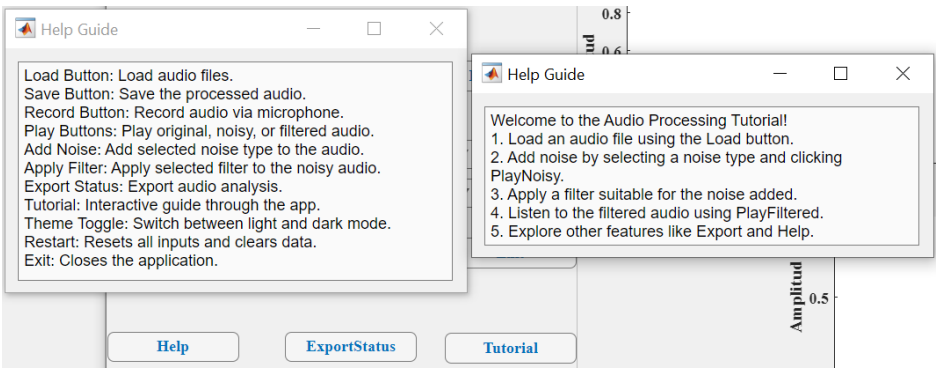

Figure 23: Help and Tutorial buttons

4. **Theme Customization Options Innovation**: Users can switch the appearance of the application by using the 'ThemeToggleButton'. This is for the aesthetic purpose of the app, and it also helps relieve strain on a user's eyes so they can view the interface comfortably under different lighting conditions by people with vision impairment. Customizability in software interfaces, especially for applications dealing with fine detail, such as audio editing, can raise user comfort and hence productivity to new levels. Therefore, allowing users to select comfortable themes for their eyes means they would use the application longer without stressing their eyes, making it even more friendly.

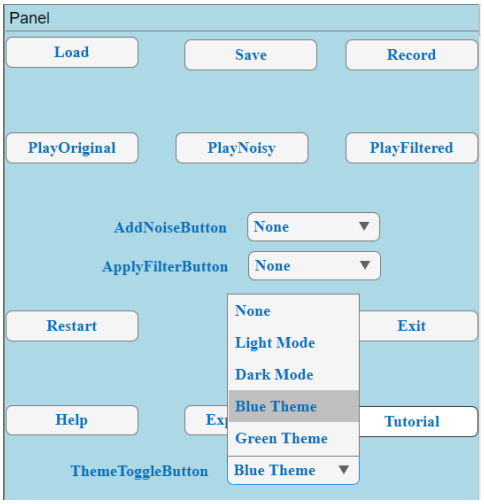

Figure 24: Theme options

SIMULATON RESULTS

In this section there will be a demonstration on how the GUI works, after that, a process will be followed to plot the graphs and observe how they behave based on each filter and noise selected.

Workflow Overview:

Startup initialization:

The user interface gets initialized, the default values for dropdowns are set, and the environment is prepared for user interaction.

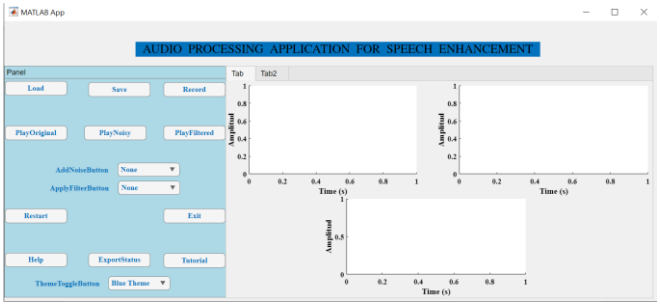

Figure 25:Main App Interface

Loading and Recording Audio:

The 'Load' button allows a user to load an audio file, causing it to respond to file selection and loading of the audio.

Recording is also available within the app through the 'Record' button, which uses an inbuilt audio recorder.

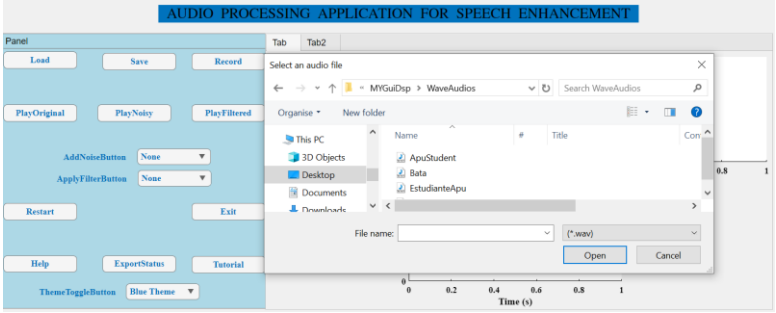

Figure 26:Loading Audio

When the sound is recorded or loaded, it is plotted in the time domain and frequency domain based on 'OriginalAxes' which is in time domain, and the spectrum is shown on 'OriginalAxes_Frequency' that is in frequency domain on the other tab.

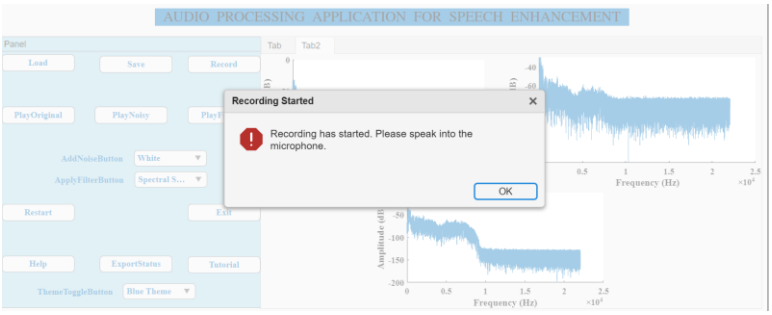

Figure 27: Recording Audio

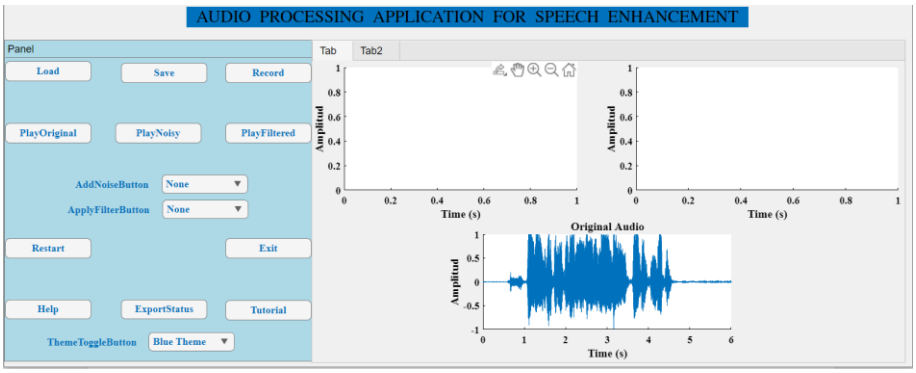

Figure 28: Time Domian Original Audio

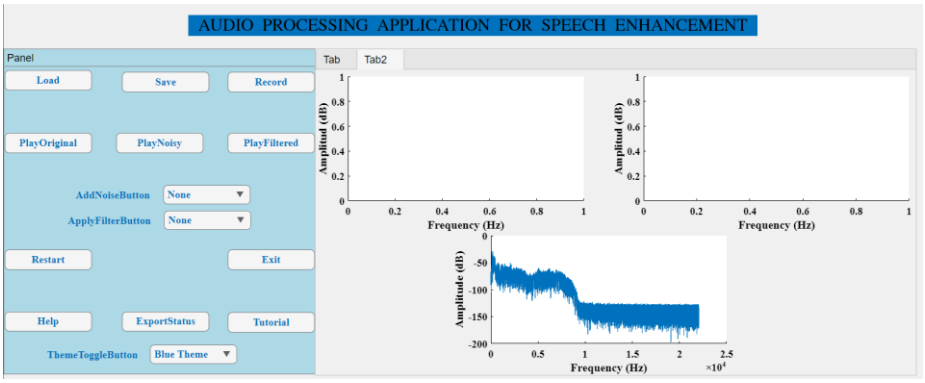

Figure 29: Frequency Domain Original Audio

Audio Playback:

This way, the app provides users with access to buttons for listening to the original version.

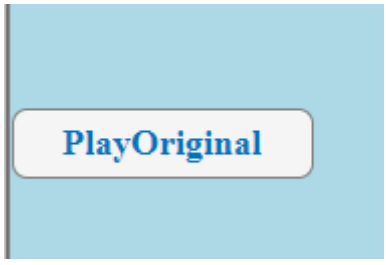

Figure 30: Play Original Audio

Adding Noise:

Users can add Brownian, White, Gaussian, and pink noise to the audio with the dropdown provided. The superimposed noise in relation to the original audio is represented on 'NoiseAxes' and 'NoiseAxes_Frequency'.

Figure 31: Select noise

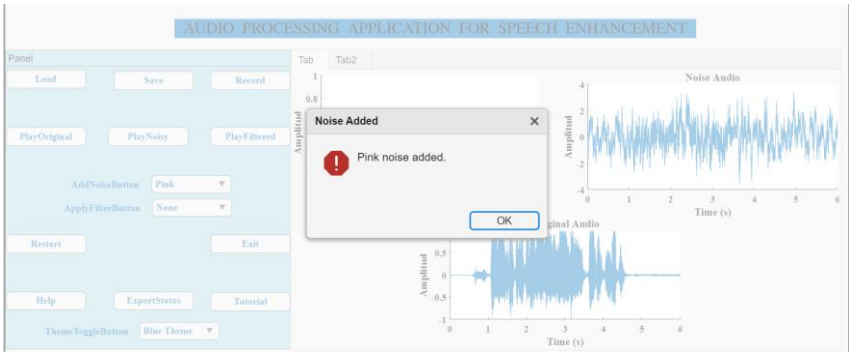

Figure 32: Pink Noise Added

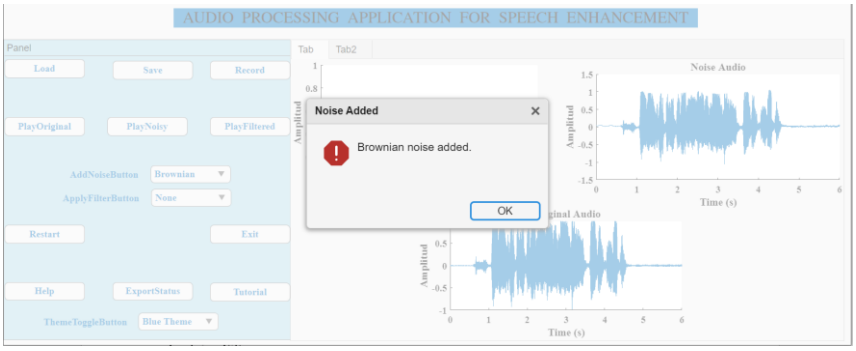

Figure 33: Brownian Noise Added

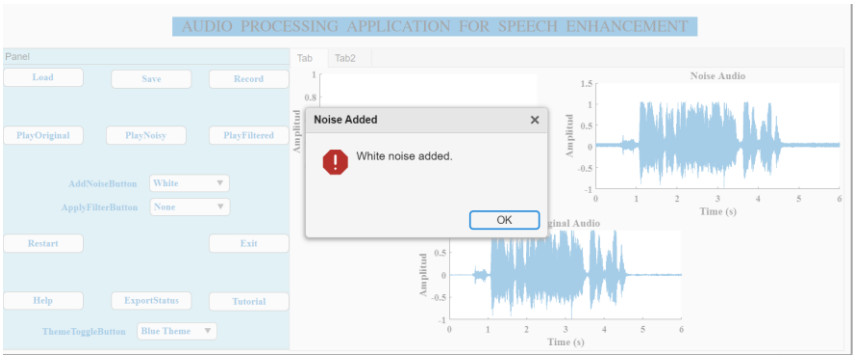

Figure 34: White Noise Added

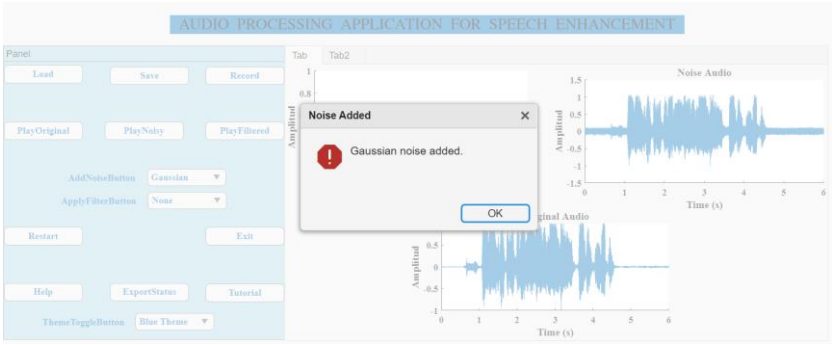

Figure 35: Gaussian Noise Added

Applying Filters:

Users could select the filters to be applied for the added noise. Upon the addition of noise, this step has validation ensuring that the appropriate filter is applied with respect to the type of noise added. The filtered audio is visualized on `FilteredAxes` and its frequency response on `FilteredAxes_Frequency`.

As you observe in the next figure, there is an error because the filter type selected is not assigned to the noise added:

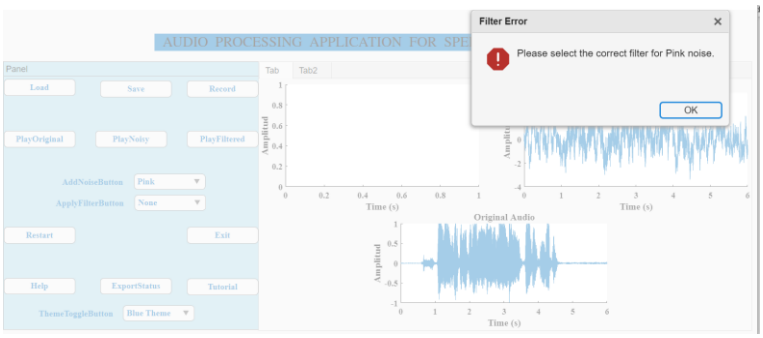

Figure 36: Filter Error Selection

The noise selected is pink and it was assigned to the Butterworth filter, so basically the user must choose the Butterworth filter:

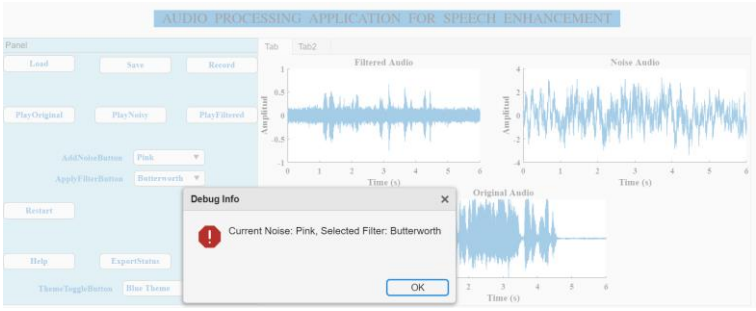

Figure 37: Butterworth filter selected

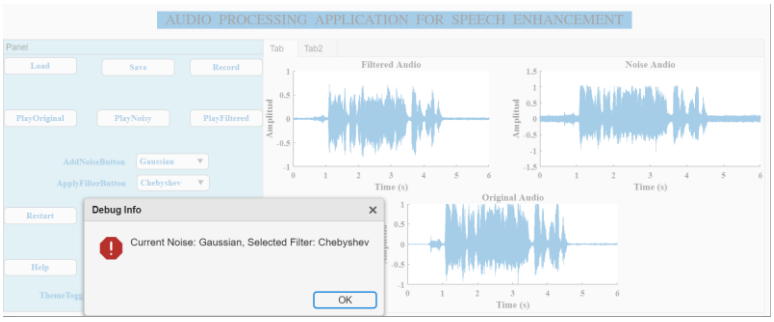

Figure 38: Chebyshev filter selected

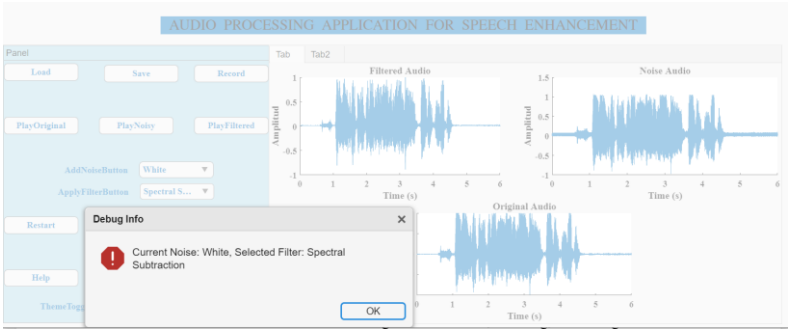

Figure 39: Spectral Subtraction filter selected

Saving and Exporting Audio

To save a processed audio file, click on the 'Save' button.

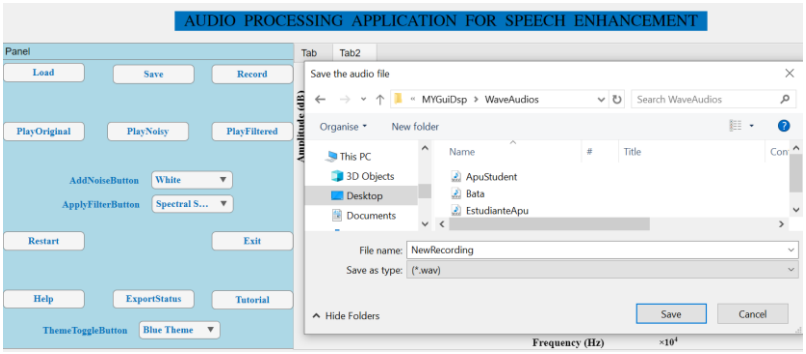

Figure 40: Save Audio

The plots for time and frequency domains of the original, noisy, and filtered audio are exported by the app with the use of 'Export Status' button.

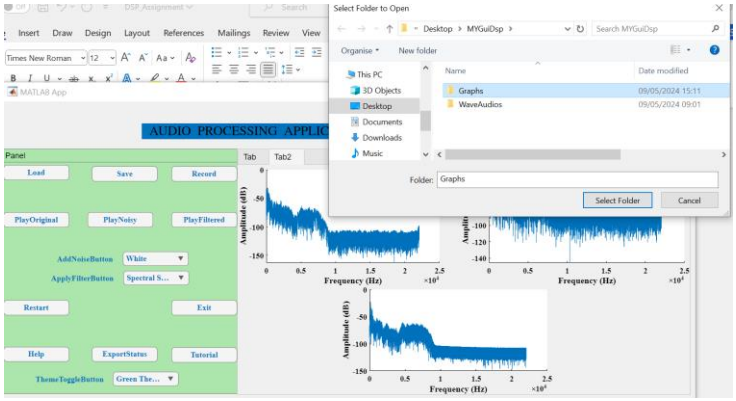

Figure 41: Exporting Graphs

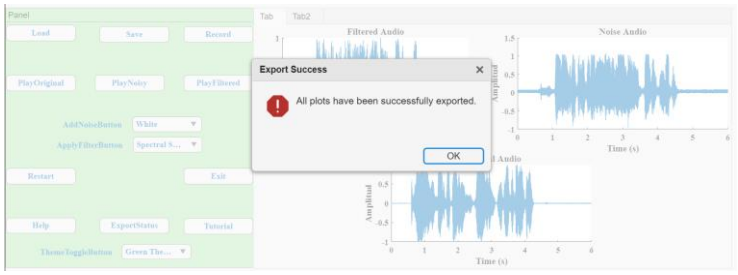

Figure 42:Saved plots

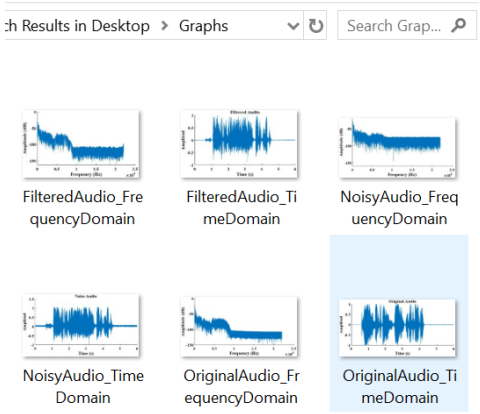

Figure 43: Graphs exported in local file

Restart and Exit Process:

If the user press exits the application, the whole system will be closed. On the other hand, if the user press restart, all the graphs will go back to the initial state as you can observe in the next figure:

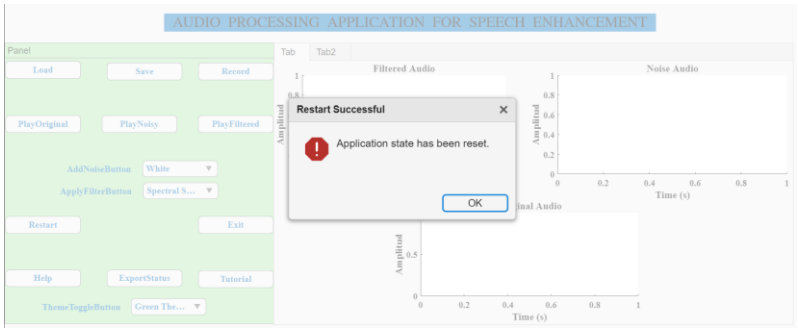

Figure 44: Restart

Validation and Error Handling:

It checks for loaded audio before allowing noise addition, filtering, or saving. It further validates the match between the type of noise and the selected filter, which will eventually be put to work for actual audio processing. Error messages are offered for actions performed in the wrong order or in the absence of necessary preconditions.

As an example: the user cannot simply press save button without a recording process or a load process:

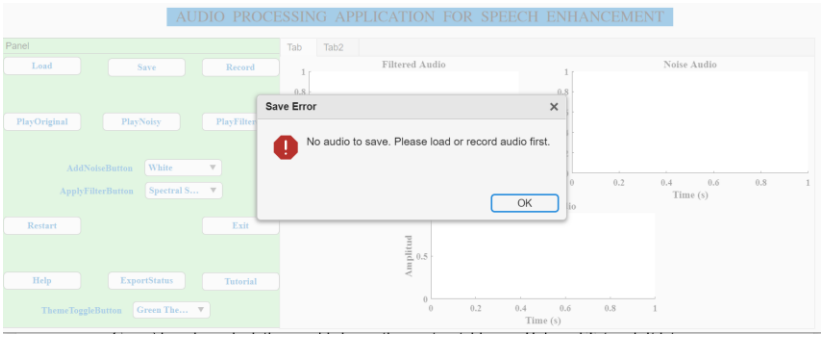

Figure 45: Validation for saving

Customization of theme: The users can toggle through the UI themes (Light, Dark, Blue, Green) based on which they could choose the most suitable one. Help and Tutorial: It lets users know how to use the application and navigate through it.

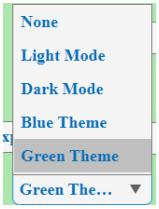

Figure 46: Theme colour

Light colour:

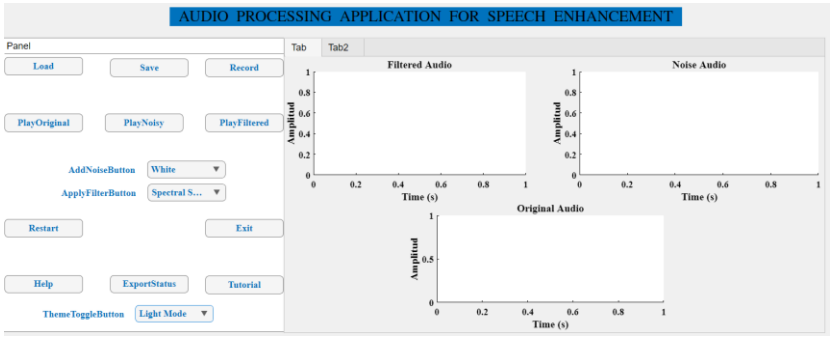

Figure 47: Light colour

Dark Mode or colour:

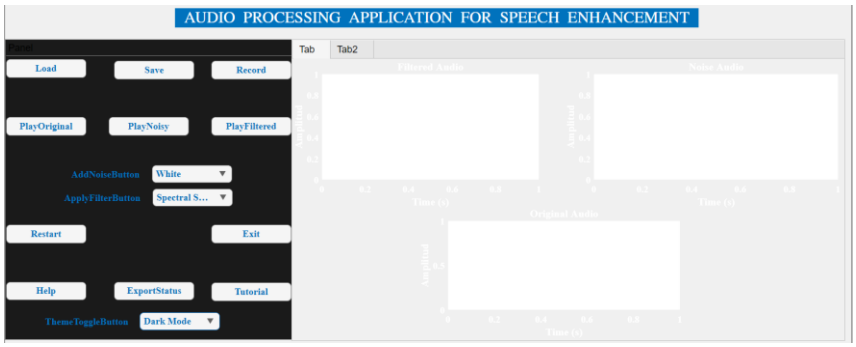

Figure 48: Dark Mode

Blue Mode:

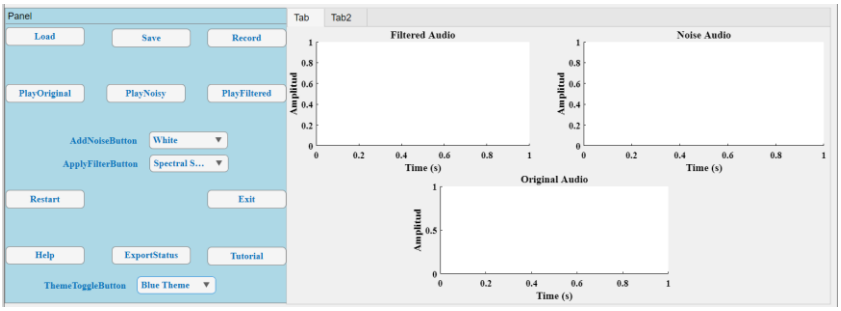

Figure 49: Blue Mode

DISCUSSION

Spectral Subtraction Noise Analysis

The next set of plots in figure 50 provides a very good visual comparison of the audio signal at different points during the processing that is performed on the signal, using the Spectral Subtraction method for noise removal. Here is an extended description of the three states of the audio signal: Original, Noise-added, and filtered.

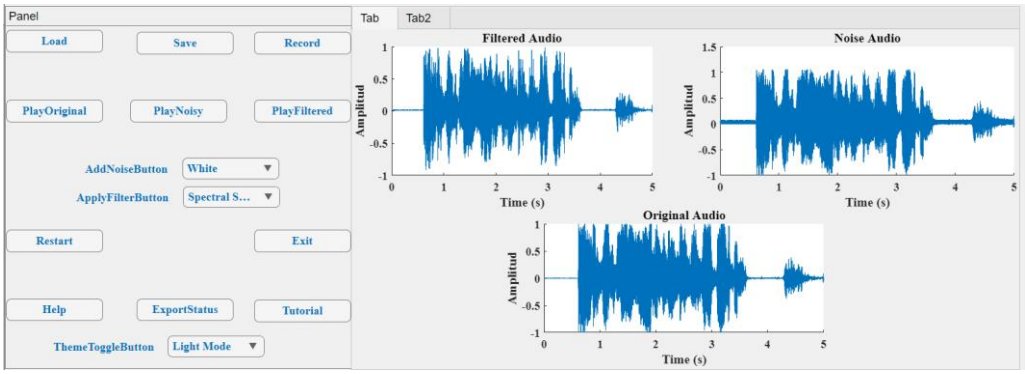

Figure 50: Spectral Subtraction and White Noise

Original Audio

The Original Audio wave provides the waveform in the time domain. It might be natural sound or human speech, given the amplitude varies and the sound intensity is distributed over time. As seen, this waveform is not symmetrical, which is normal for the natural fluctuation of the sound source amplitude in the original source. The salient feature of this is that it is crystal clear—no artificial disturbances or noises—which becomes a reference to compare how effectively noise is introduced and then filtered at later stages.

Audio with noise

The second plot is the sound after the addition of White Noise. White Noise has the characteristic of having spectral density constant at any frequency. The signal in all time components sounds like a continuous noise signal without any variation. The impact of White Noise can be seen with the visual effect of disguising the underlying original waveform and making the graph more chaotic and populated. This addition of noise is supposed to simulate the common real-world scenario where the audio recording is marred by ambient or systemic noise.

Filtered Audio

The final spectrogram is the audio that has been processed using the Spectral Subtraction technique. This technique attempts to model the power spectral density of the noise computed in the first few silent frames and subtract that from the spectrum of the noisy signal. The components of noise are much less visible than in the rest. The underlying audio is clearer compared to the Noise-added audio. However, the process is not ideal because certain residual noise is still evident, which could be heard either as a kind of distortion or that the waveform is no longer smooth as before.

Butterworth Filter Noise Analysis

Diagrams have been refreshed, showing the stages of processing audio with Pink Noise together with a Butterworth filter as reflected in the next figure. Stages represented by the graphs, in this case, are the original audio, audio with added noise, and filtered audio. By this, the performance of Butterworth filters at reducing Pink Noise is reflected in the time domain.

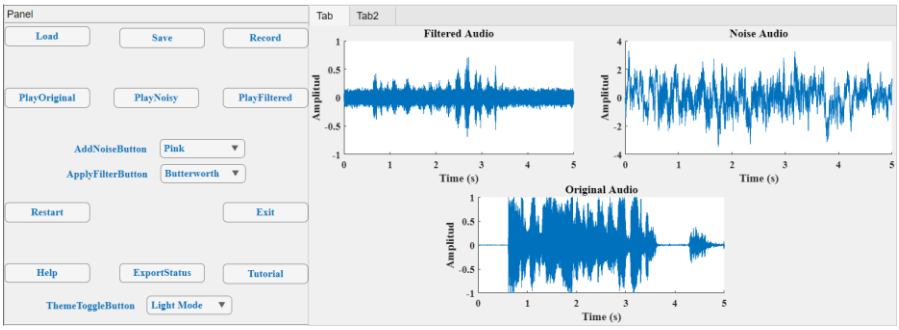

Figure 51: Butterworth with Pink noise

Original Audio

The Original Audio plot is a plot of the unmodified audio waveform. This is the plot that is used as the reference for all other modifications. The amplitude and frequency characteristics are diverse in this waveform as in any natural audio recording, for example speech, environmental sounds. A dynamic range can be seen with peaks and troughs to suggest the presence of different sounds or different elements of speech, but not some noise blanket that is uniform.

Noisy Audio

This plots the Noise-added Audio plot, in which Pink Noise has been superimposed over the original audio. The general property of pink noise is that the power density decreases with increasing frequency. An increase at low frequencies and decrease at higher frequencies may give rise to a noise profile that is more intense at low frequencies and diminishes at higher

ones. Such a characteristic is somewhat visible in the plot since the noise does not spread over the waveform uniformly over all sections but gives a dense irregular pattern. In fact, it destructively largely destructively distorts the original audio. This is a very destructive type of noise, as it can mask significant low-frequency information in the original audio.

Filtered Audio

The Filtered Audio plot is the waveform after a Butterworth filter is used. This class of filter is known for its flat frequency response in the passband, and its use is generally preferred for minimum ripple characteristics. In this case, the use of the Butterworth filter will remove the superimposed Pink Noise but keep the original feel of the audio. The noise level has been considerably reduced, as is the case with the filtered audio, wherein it is most prominent in the smooth parts, where the features of the original audio start to come out again. However, there is still the occurrence of the regions showing the influence of the noise, which means that while the filter succeeds in reducing much of the noise, it does not, by any chance, filter it out in totality, which is the case in most practical scenarios, where a filter tries to strike the balance between reducing the noise and keeping the original audio intact.

The Butterworth filter used here likely operates as a low-pass or band-pass filter, targeting the effective frequencies where Pink Noise is dominant. The choice of the cutoff frequency, and hence the filter order, is critical: too low a cutoff might remove essential parts of the original audio, while too high allows too much noise to remain. The following plots give an indication of how moderate success is achieved, where a trade-off is struck in such a way that most of the original audio becomes clearer, but some noise artifacts remain.

Chebyshev Filter Noise Analysis

The plots in the next figure depict the process of transforming an audio signal: first, a noise type is added, and then the noise is reduced by the filter. In these plots, Gaussian noise has been added to original audio, and the Chebyshev filter is used in noise reduction. Let's see the effect of the Chebyshev filter on Gaussian noise, and each of the plots represents each of the phases.

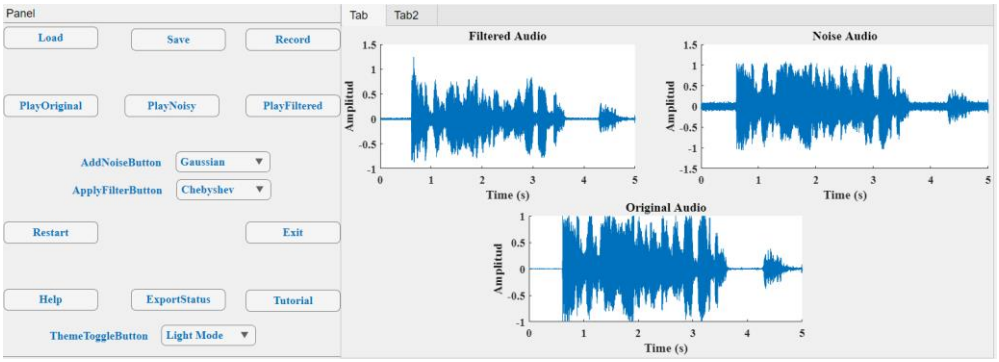

Figure 52: Chebyshev with Gaussian noise

Original Sound

The Original Audio plot is the representation of the baseline audio with no changes. The waveform takes on the typical variations of a real-world audio signal, potentially including speech, music, or environmental sounds. This waveform provides the baseline that we compare against after noise addition and noise reduction processes.

Noisy Audio

The Noise Audio plot shows the effect of injecting Gaussian noise into the original audio. Because of the bell shape in its probability density function, Gaussian noise is uniformly scattered throughout the signal, such that the variance in amplitude generally increases in the signal, and the original audio starts losing its identity. This kind of noise especially covers finer details within the audio signal; it is almost impossible to pick subtle variations within it.

Sound Filter

The Filtered Audio plot is the signal after it has been passed through a multistage Chebyshev filter. It is a cascade of low-pass and high-pass filters to take off as much noise as possible while keeping the shape of the original signal. An abrupt Chebyshev filter, with ripples, is helpful in separating the noise from the real signal. You can see the general attenuation of the noise signal here, particularly at the upper part of the frequency owing to the addition of low-pass filters. In this sense, high-pass filtering is useful to keep sharpness in the signal so that there is no loss of detail.

The remarkable effect noise has on an audio signal becomes more pronounced when these plots are seen in sequence; the filter proves to be a good corrective means back to something closer

to normal. The transition from original, to noise-added, and then to filtered audio makes clearer the problem and solution that one often deals with in practice: noise is of major concern in the audio processing environment.

CONCLUSION

The Assignment Report presents a design and features of a MATLAB App Designer based speech enhancement application. The application integrates many types of noises and filters to it including Kalman, Spectral Subtraction, Chebyshev, Butterworth filters, Brownian noise, White, Gaussian, and pink noises to enhance the quality of the audio. To a user-friendly graphical interface, the key functionalities of recording, playing, adding noise, filtering, and saving are incorporated. This interface allows customization of themes and carries interactive tutorial and help sections to further enhance user experience. This application supports an in-depth test of the combined use of various noises and filters and demonstrates the efficiency of each filter in the elimination of definite types of noise from audio signals.

REFERENCES

Haykin, S. &. (2008). *Signals and System(2nd).* Wiley.

Lyons, R. G. (2011). *Understanding Digital Signal Processing (3rd ed.).* Prentice Hall.

Oppenghelm, A. V. (2013). *Discrete-Time Signal Processing (3rd ed).* Pearson.

Proakis J. G., M. D. (2013). *Digital Signal Processing: Principles, Algorithms, and Applications (4th Edition).* Pearson.

Rabiner, L. R. (1975). *Theory and Application of Digital Signal Processing.* Prentice Hall.