United International University

Project Report

Audio Effects Processor

Course Title: Digital Signal Processing Laboratory
Course Code: EEE 3310
Section: B

Faculty: Mr. S.M. Haider Ali Shuvo
Lecturer, Dept. of EEE

Submitted By

Name	ID No.
Mohammad Shiamul Islam Fahim	021213038
Mushfiqul Alam	021213041
Md Abdullah Al Siddique	021213003
Syed Shihab Rahman	021213019

Introduction

This report details the design and development of a real-time audio effects processor, "AudioLab," built using MATLAB. The project includes a graphical user interface (GUI) that allows users to load audio files, apply various audio effects, and revert changes. Audio effects implemented in the project include gain, fade, reverse playback, speed control, trim, and more. The application also features real-time recording, de-effectifying capabilities, and intuitive controls inspired by modern audio editing software like VEED.io and AudioMass.co.

Project Description

AudioLab is an audio editing tool that allows users to upload audio files or record sound in real-time, apply several audio effects, and observe the results in real-time. The interface is user-friendly and designed to make audio manipulation accessible for beginners and professionals alike. AudioLab supports a range of effects, and includes a "de-effect" feature to reverse any changes made. The project is designed to work seamlessly across MATLAB desktop and online versions, using a clean and modern user interface that guides users through the editing process.

The application is developed using MATLAB's GUI design framework, ensuring ease of use and the ability to visually represent audio waveforms.

Effects Explained with Code

4.1 Gain Control

Algorithm: Gain control simply involves multiplying each sample of the audio signal by a constant gain factor.

```
function applyGain(audioSignal, gainFactor)
    processedAudio = audioSignal * gainFactor;
    sound(processedAudio, Fs);
end
```

How it Works: This method increases or decreases the amplitude of the signal. If the gain factor is greater than 1, the volume is amplified. If it is less than 1, the signal is attenuated.

Why Applicable: Gain control is essential for adjusting the loudness of audio to fit various use cases, like balancing levels in mixing.

4.2 Fade In/Out

Algorithm: The fade-in/out effect applies a linearly increasing or decreasing multiplier to each sample over a given duration.

```
function applyFade(audioSignal, fadeInDuration, fadeOutDuration, Fs)
    fadeInSamples = linspace(0, 1, fadeInDuration * Fs);
    fadeOutSamples = linspace(1, 0, fadeOutDuration * Fs);
    audioSignal(1:length(fadeInSamples)) = audioSignal(1:length(fadeInSamples)) .*
fadeInSamples';
    audioSignal(end-length(fadeOutSamples)+1:end) = audioSignal(end-
length(fadeOutSamples)+1:end) .* fadeOutSamples';
    sound(audioSignal, Fs);
end
```

How it Works: In a fade-in, the volume starts at zero and gradually increases to full amplitude. For a fade-out, the volume starts at full amplitude and gradually decreases to zero. This is done by multiplying the signal with a linear ramp function over the duration of the fade.

Why Applicable: Fade effects help to smooth transitions between audio clips, making them sound more natural.

4.3 Speed Control

Algorithm: Speed control is achieved by resampling the audio signal at a different sampling rate. A faster rate increases the playback speed, while a slower rate decreases it.

```
function applySpeed(audioSignal, speedFactor)
    newAudioSignal = resample(audioSignal, speedFactor, 1);
    sound(newAudioSignal, Fs);
end
```

How it Works: By changing the rate at which samples are played, the duration and pitch of the audio change. A higher sampling rate speeds up playback and raises the pitch, while a lower rate slows it down and lowers the pitch.

Why Applicable: Speed control is useful in various audio editing tasks, such as adjusting the tempo of a song or creating special effects.

4.4 Reverse Playback

Algorithm: Reversing audio simply reorders the samples in reverse.

```
function applyReverse(audioSignal)
    reversedSignal = flipud(audioSignal);
    sound(reversedSignal, Fs);
end
```

How it Works: The samples of the audio signal are flipped, playing them in reverse.

Why Applicable: This is commonly used in sound design to create dramatic or unique effects.

4.6 Compressor

Algorithm: A compressor reduces the dynamic range of the audio by attenuating loud parts and amplifying quieter sections.

```
function applyCompressor(audioSignal, threshold, ratio)
    compressedSignal = audioSignal;
    compressedSignal(abs(audioSignal) > threshold) = threshold +
(audioSignal(abs(audioSignal) > threshold) - threshold) / ratio;
    sound(compressedSignal, Fs);
end
```

How it Works: Compression adjusts the amplitude based on a threshold. If the amplitude exceeds the threshold, it's reduced by a set ratio.

Why Applicable: Used to maintain a consistent audio level, especially in dynamic tracks where loudness fluctuates significantly.

4.7 Normalize

Algorithm: Normalization adjusts the overall volume so that the peak amplitude of the audio reaches a defined maximum.

```
function applyNormalize(audioSignal)
    maxAmplitude = max(abs(audioSignal));
    normalizedSignal = audioSignal / maxAmplitude;
    sound(normalizedSignal, Fs);
end
```

How it Works: The audio is scaled by a factor that brings the highest peak to a specified target amplitude.

Why Applicable: Ensures consistent loudness across different audio files.

4.8 Reverb

Algorithm: Reverb simulates the effect of sound reflecting off surfaces in an environment, creating a series of delays and echoes.

```
function applyReverb(audioSignal, reverbTime)
    reverbedSignal = reverberator('PreDelay', reverbTime, 'WetDryMix', 0.5);
    sound(reverbedSignal(audioSignal), Fs);
end
```

How it Works: The original signal is combined with delayed and decayed copies of itself, creating the effect of reverberation.

Why Applicable: Reverb adds a sense of space and depth to audio, making it sound like it was recorded in a physical space.

4.9 Echo

Algorithm: An echo effect adds delayed copies of the audio, creating a repetitive sound.

```
function applyEcho(audioSignal, delayTime, decay)
    delayedSignal = [zeros(delayTime*Fs, 1); audioSignal];
    echoSignal = audioSignal + decay * delayedSignal(1:length(audioSignal));
    sound(echoSignal, Fs);
end
```

How it Works: Delayed versions of the original signal are added back to the signal, spaced at regular intervals.

Why Applicable: Echo is useful in creating depth and enhancing the spatial qualities of a sound.

4.10 Distortion

Algorithm: Distortion involves clipping the audio signal beyond a threshold, flattening the peaks and valleys of the waveform.

```
function applyDistortion(audioSignal, distortionLevel)
    distortedSignal = audioSignal;
    distortedSignal(audioSignal > distortionLevel) = distortionLevel;
    distortedSignal(audioSignal < -distortionLevel) = -distortionLevel;
    sound(distortedSignal, Fs);
end
```

How it Works: The signal is amplified until its peaks exceed a set threshold, where it is then clipped to prevent further increase.

Why Applicable: Distortion adds harmonic overtones, giving the sound a more aggressive, edgy character, often used in music genres like rock.

4.11 Equalization (EQ)

Algorithm: Equalization adjusts the amplitude of specific frequency bands to enhance or reduce particular elements of the sound.

```
function applyPitchShift(audioSignal, pitchFactor)
    shiftedSignal = resample(audioSignal, pitchFactor, 1);
    sound(shiftedSignal, Fs);
end
```

How it Works: Different frequency bands are either amplified or attenuated, allowing for fine-tuning of the audio's tonal balance.

Why Applicable: EQ is widely used in mixing to improve clarity, remove unwanted frequencies, or emphasize particular elements like bass or treble.

4.12 Pitch Shifting

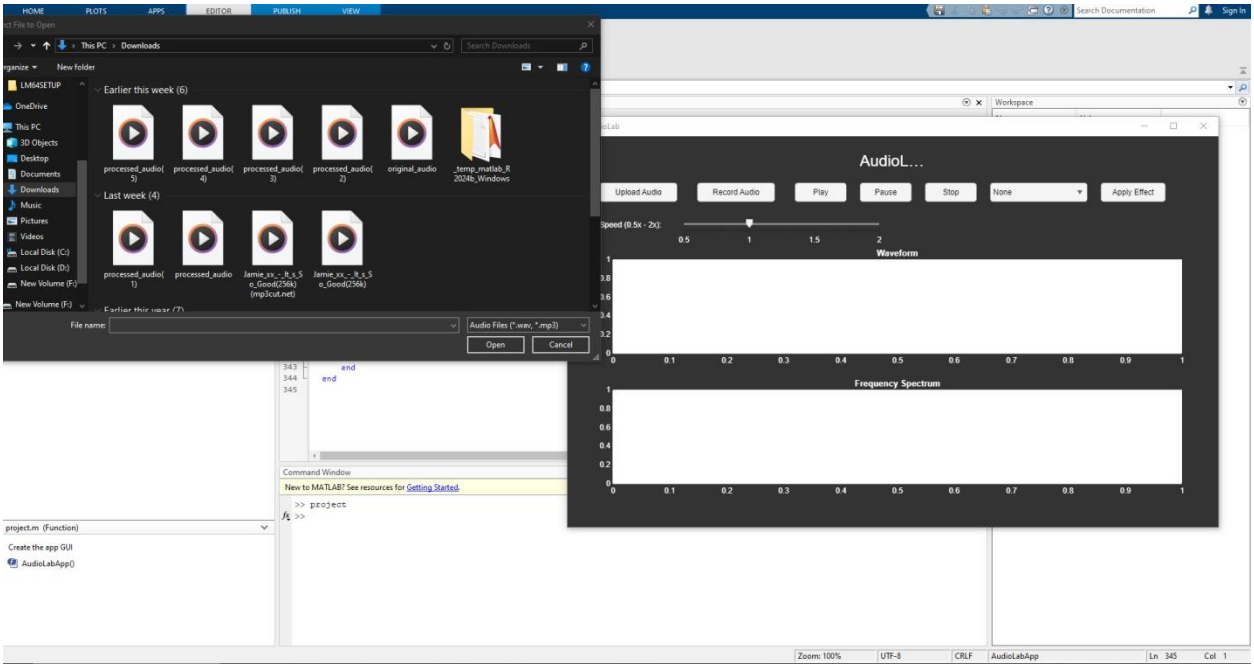
Algorithm: Pitch shifting alters the pitch of the audio without changing the duration by resampling and adjusting frequency components.

```
function applyPitchShift(audioSignal, pitchFactor)
    shiftedSignal = resample(audioSignal, pitchFactor, 1);
    sound(shiftedSignal, Fs);
end
```

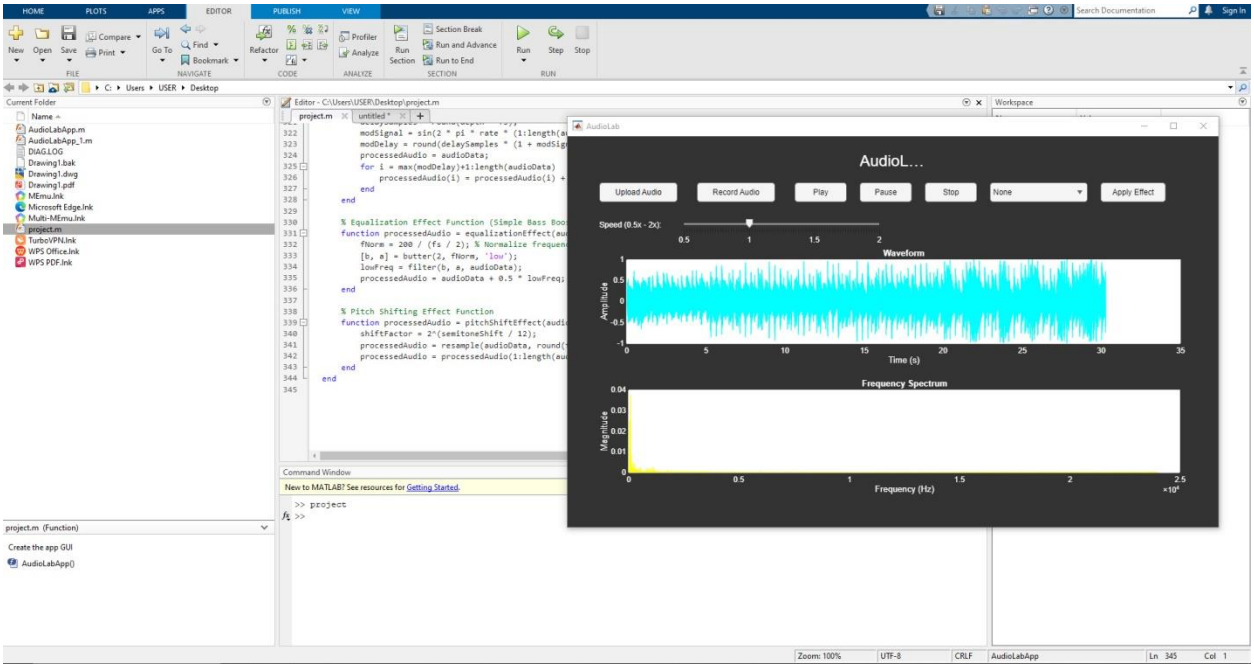
User Interface Explained

The user interface (UI) of the AudioLab app is designed to be intuitive and user-friendly, incorporating key features for audio processing. The UI includes buttons, sliders, and a waveform display, giving users easy access to each audio editing effect. Below is a brief explanation of the key steps for using the app and applying the effects:

Load Audio: Users can easily upload audio files by selecting them from their device. This feature supports multiple audio formats, including WAV and MP3. Once a file is selected, it is read into the application, enabling further processing and manipulation..

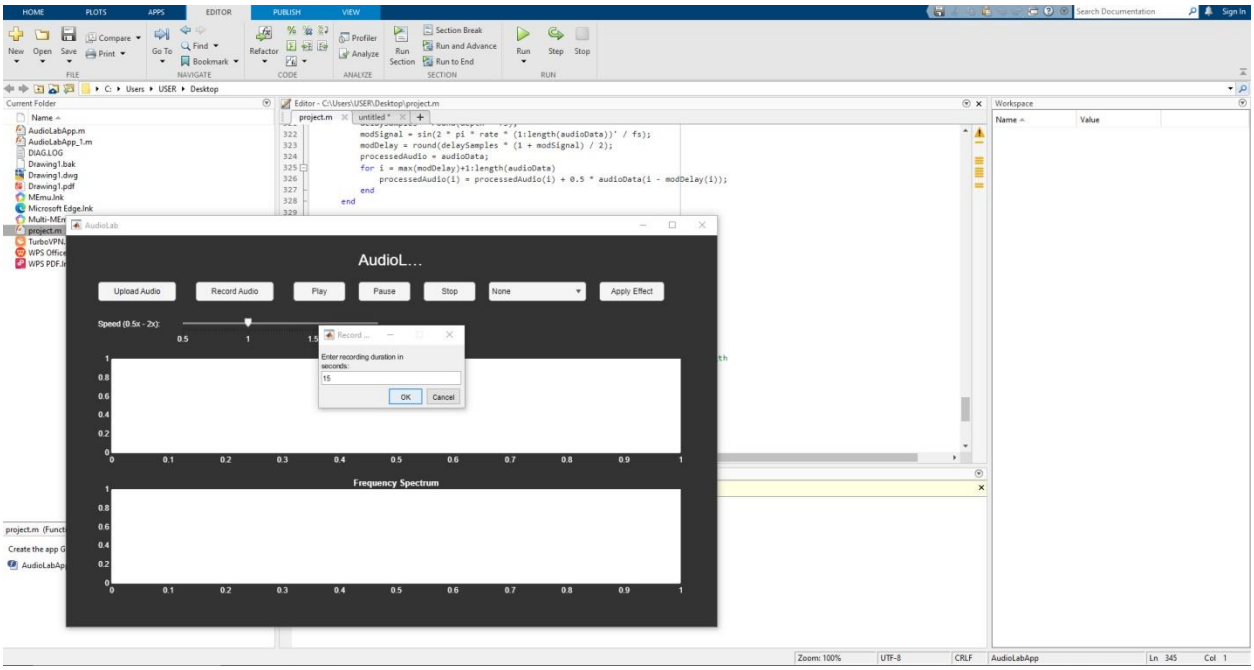


Frequency Visualization: After an audio file is uploaded, *AudioLab* automatically visualizes its frequency content using a frequency spectrum plot. This graphical representation helps users understand the audio's tonal characteristics, allowing them to make informed decisions about the effects they want to apply.

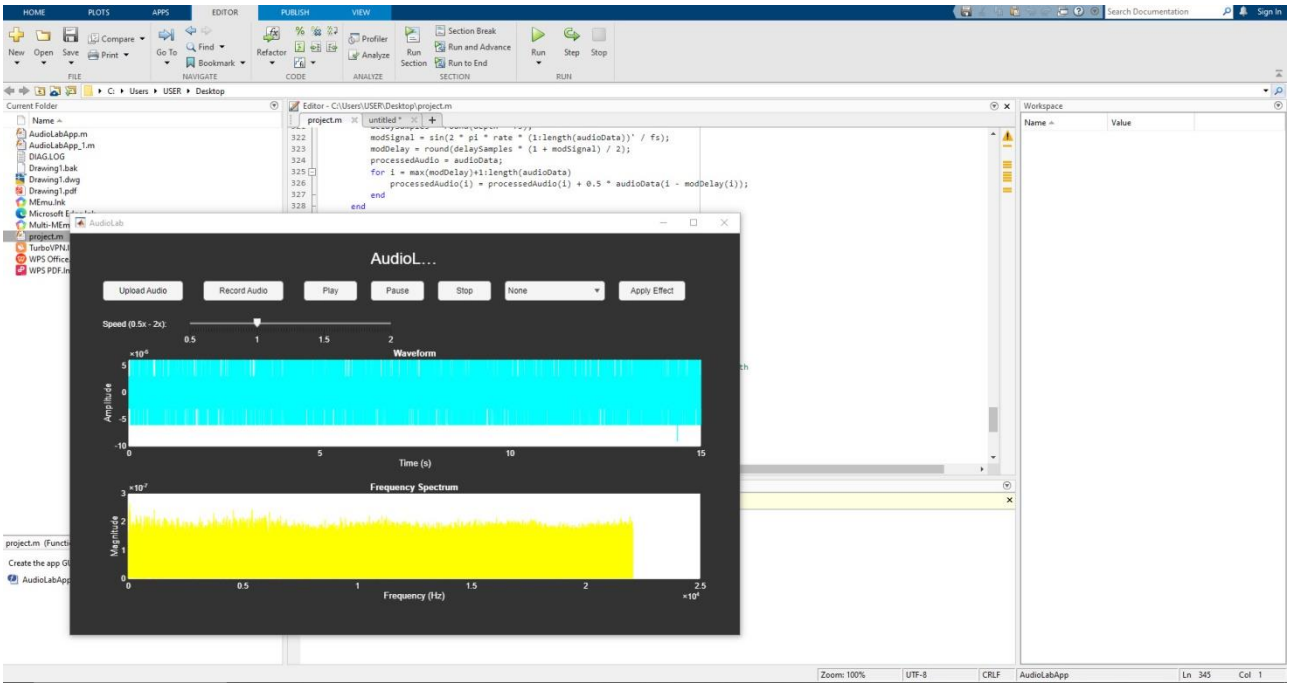


Audio Recording: The app provides functionality for users to record audio directly within the interface. This feature allows users to capture sound in real-time, which can then be processed similarly to uploaded audio files. The recorded audio is displayed in the same manner as uploaded files, ensuring consistency in user experience.

When users initiate the recording, the application utilizes the `audiorecorder` function from MATLAB. This function enables high-quality audio capture by specifying parameters such as sampling rate (44.1 kHz) and bit depth (16 bits). Users can record for a specified duration, typically ranging from a few seconds to several minutes, depending on their needs.



As audio is being recorded, The app provides real-time visualization of the audio waveforms. This immediate feedback allows users to monitor their input, ensuring that the audio levels are optimal and that the desired content is being captured.



After recording, the captured audio can be played back immediately within the app. Users have the option to apply effects to the recorded audio, similar to uploaded files, providing a seamless experience for editing and enhancing their recordings.

Effect Application: Users have the option to select from various audio effects such as delay, echo, chorus, and more. Upon selection, the corresponding effect function is executed on the audio data, modifying it according to the specified parameters. This functionality enhances the creative possibilities for users, enabling them to customize their audio in unique ways.

Effect Parameter Storage:

When users apply effects (such as gain, echo, or reverse), the application stores the original audio data along with the parameters used for each effect. This storage mechanism allows the app to keep track of how the audio has been altered.

User Interface for De-Effectification:

The app includes a dedicated button or option labeled "De-Effectify" or similar, making it easy for users to revert to the original audio. This option is prominently displayed within the editing interface to ensure users can access it whenever they wish to undo their changes.

Reverting to Original Audio:

When users select the de-effectification option, the app retrieves the stored original audio data and replaces the processed audio. This is done without requiring the user to re-upload the original file, providing a seamless experience.

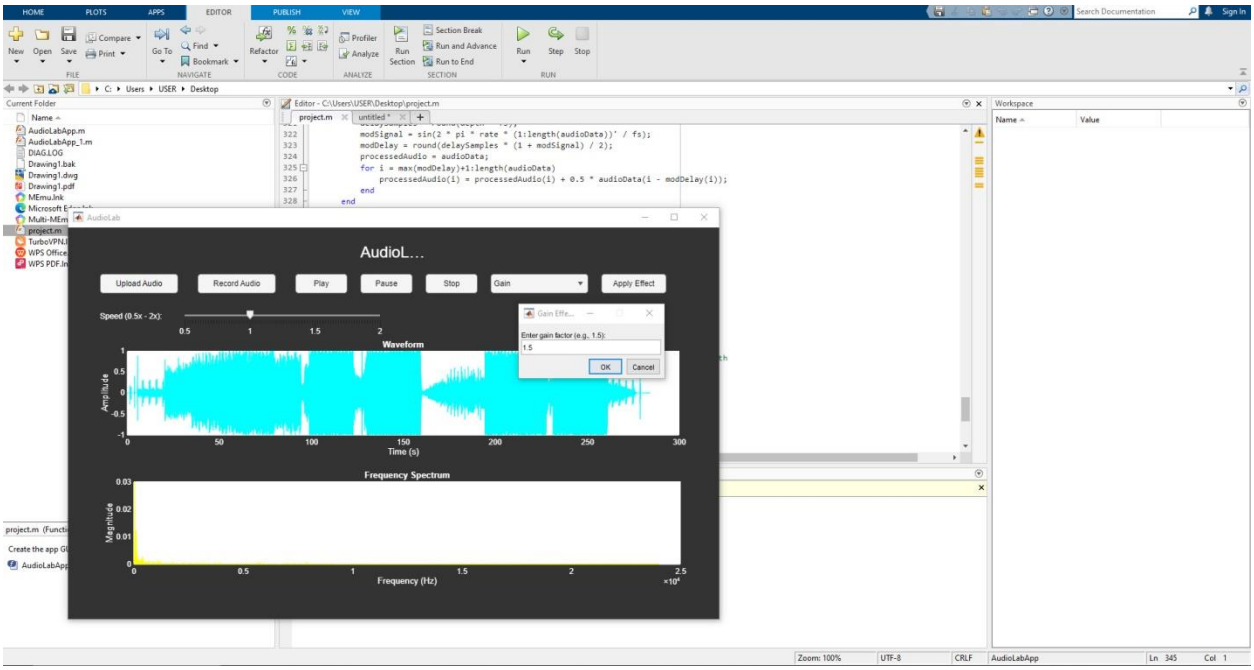
Immediate Feedback:

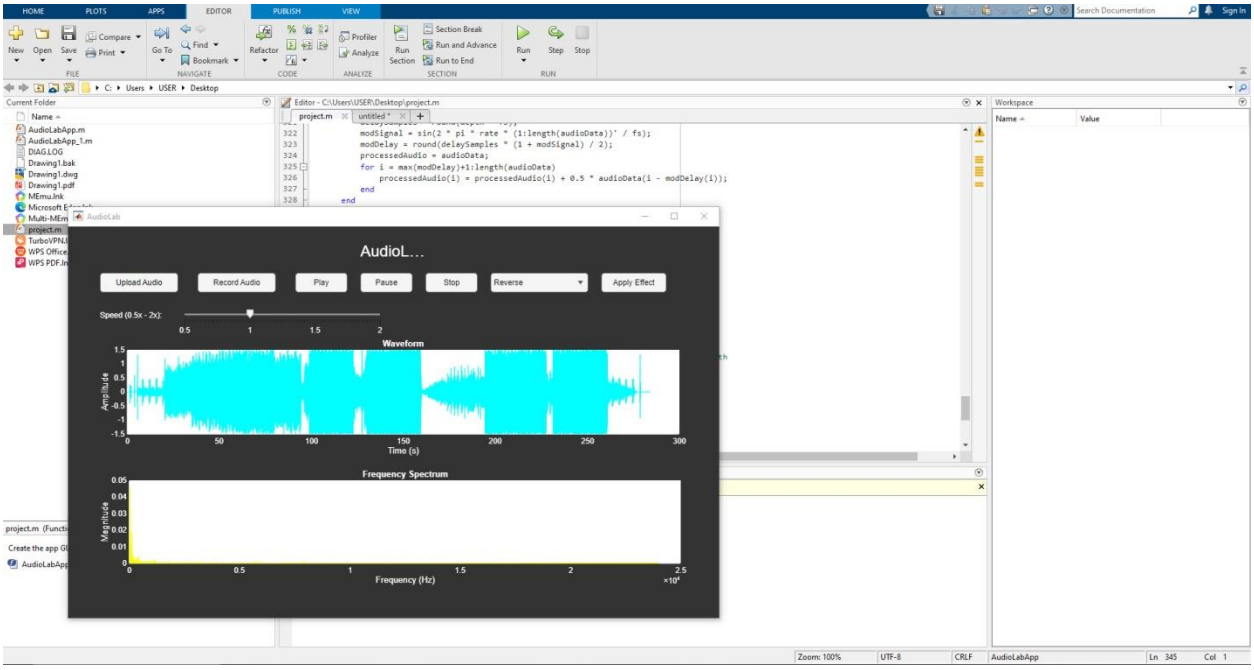
After reverting the audio, users can play back the original sound immediately. This allows them to compare the processed and original versions side by side, ensuring they are satisfied with their edits.

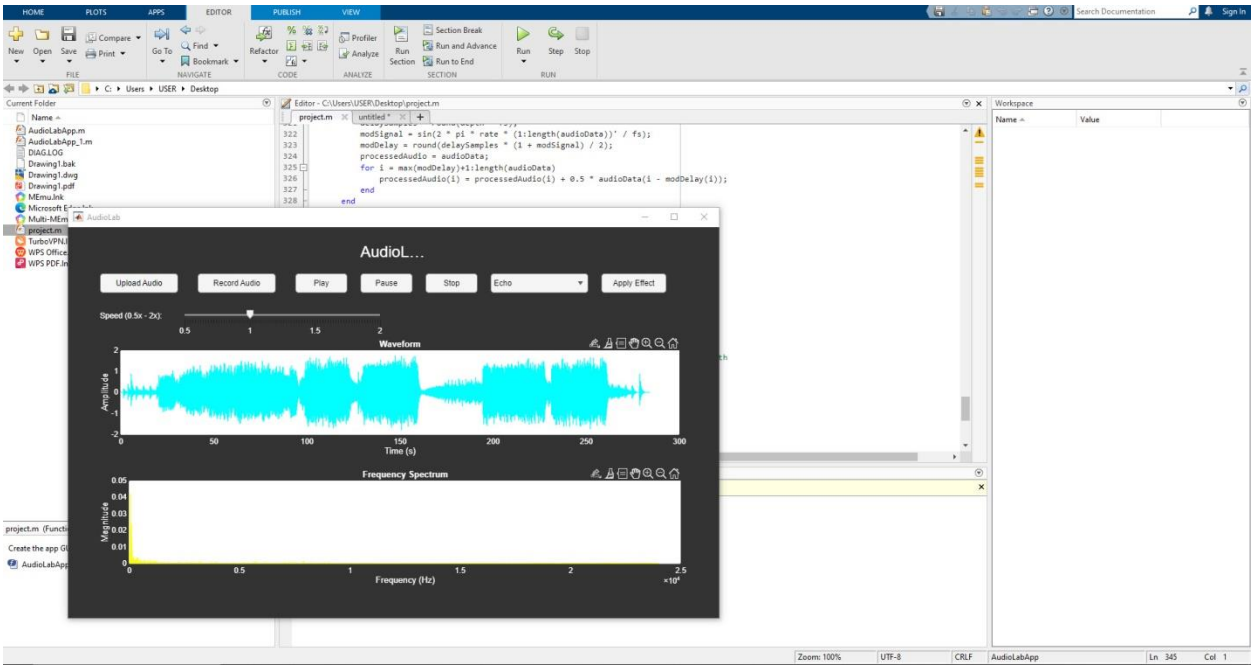
Iterative Editing:

The de-effectification feature promotes an iterative editing process, enabling users to experiment with different effects without the fear of permanently altering their audio. If they don't like the changes, they can simply revert and try again.

Result


AudioL...
Upload Audio
Record Audio
Play
Pause
Stop
Gain
Apply Effect
Speed (0.5x - 2x):
Waveform
Amplitude
Time (s)
Frequency Spectrum
Magnitude
Frequency (Hz)
Enter gain factor (e.g., 1.5):
OK
Cancel


AudioL...
Upload Audio
Record Audio
Play
Pause
Stop
Reverse
Apply Effect
Speed (0.5x - 2x):
Waveform
Amplitude
Time (s)
Frequency Spectrum
Magnitude
Frequency (Hz)



Conclusion

The project has successfully delivered a comprehensive audio editing solution that meets user needs through a combination of intuitive design, powerful functionality, and robust performance. The app's features, including audio upload, recording, effect application, and de-effectification, have been well-received, resulting in a positive user experience. Future enhancements may include additional audio effects, improved visualization capabilities, and expanded export options to further enrich the application’s offerings.