CSCI3280 Introduction to Multimedia Systems

Group 7 Project Progress Report

1. Introduction

   The voice recorder is implemented using Python, with PyQt5 for the graphical user interface. Apart from the basic features, it also supports Noise Reduction, Pitch Shifting, Equalizer of Low Pass Filter and High Pass Filter, Audio to Text, Selection of Input Device, and Visualization of Waveform.

2. Basic Requirements

2.1 Graphical User Interface

In the GUI, there is a list box on the left showing all the wav files, there will be a waveform showing on the right after selecting a wav file. Under the waveform, there is a bar showing the current position of the audio played. At the bottom, there is a drop list for selecting the input device, a record button, a stop button, the duration of the audio, a play button, a volume control slider, a button for audio to text (the result will pop up after clicking it), and 5 editing options: Audio Trim, Overwrite, Noise Reduction, Pitch Adjust, and an Equalizer. Selecting one of these options, except noise reduction, will create a pop-up window, which requires users to enter the parameter. The processed audio will be saved afterward.

2.2 Sound Recording and Saving

There are two triggers in the GUI to call this function – **play** and **stop**. When **play** is pressed, the user needs to specify his desired **frame per second**, **number of channels (1 or 2)**, and **microphone**. The user will select his desired microphone through the drop list in GUI, supported by the function **getDeviceList()**. The recording function makes use of the **pyAudio** library. When the user presses the **stop** button, it will stop recording and save to the specified path, **struct** library is used to convert manually specified headers to byte objects. The recording is set to PCM format with 32-bit depth. **Threading** library is used to perform asynchronous programming to stop the recording while the previous function is still playing the audio.

2.3 Sound Playback with Speed Control

For wav file parsing, the file is read in **playback.readWav()** according to its header information. After that, validation is done to check if the wav file is valid. Then, all its information is stored in a dictionary.
To provide speed control, the raw data of the audio is converted to a **NumPy** array for further editing and resampling in **playback.getData()**. For 2x speed, the samples in the array are doubled. For 0.5x speed, the samples are grouped as 2 and the average of the 2 samples is taken to reduce the number of samples by half. The NumPy array of normal speed, 2x speed, and 0.5x speed are stored in the dictionary of wav file.
For the sound playback, **PyAudio** is used. After defining a stream using **playback.getStream()**, the sound is played according to the speed, volume, and start time input using **stream.write()**. The volume is controlled by multiplying the NumPy array by the input volume. The start time of the audio is controlled using array slicing with sampling rate of the wav file and the input start time. After these editing, the NumPy array is converted back to bytes.

2.4 Audio Trim

This function consists of 2 parts – **overwriting** and **trimming**. The user will pick one audio and select the start and end points as the input for overwriting, then with use of function 2.2, an audio will be recorded with the number of channels same as the original clip. The output will be changed to 32-bit depth. For trimming, the user will pick one audio, select the start and end points, and specify the desired volume and speed for the trimmed video. The function in 2.3 is utilized for data reading.

3. Enhanced Features

3.1 Audio Editing

There are two parts included in this feature, which are **pitch adjustment** and **audio equalizer** respectively. The pitch of the input wav file if the pitch adjustment function is called. The range of pitch is limited from –12 to 12 for better results and the pitch shift function from **librosa** library is utilized. For the audio equalizer, users can choose using the low pass filter or high pass filter to enjoy different audio effects on sound. The former one will thicken the sound while the latter one will shaper the audio. When the function is used, the **butter** and **filtfilt** API from the **SciPy** library will be used to perform the effects on wav audio data.

3.2 Background Noise Removal

This function will be triggered if the user presses the **noise removal** button and specifies the input and output path. The **NumPy** library is utilized to limit the audio data bound between –1 and 1 and proceed with the **noisereduction** library. The data is converted back to standard format and save is the desired path.

3.3 Audio to Text

This function will be triggered if the user presses the **audio to text** button and specifies the input path. The **speech_recognition** library is used to reflect the text of full audio in the GUI. This function supports 16-bit and 32-bit depth, 1 or 2 channels. If there are 2 channels, the text of the 2 channels will be shown respectively.

3.4 Visualization

For the audio visualization, the data in **NumPy** array is used. Using **Matplotlib**, the amplitude of the audio can be plotted against time.

4. Contributions

Name + SID	Contributions
LEE Lai Yan 1155158772	Sound Playback with Speed Control, Audio Trim, Audio Overwrite, Visualization
LEE Ho Kan 1155157376	Sound Recording, Speech to Text, Noise Reduction, Audio Overwrite, GUI, Other Bug Fixing
NG Ka Pun 1155160398	Basic Graphical User Interface Design, Function Refinements
TAM Lee Yau 1155143460	Audio Editing of Adjusting Pitch, Audio Equalizer of Low Pass Filter & High Pass Filter