HELMUT SCHMIDT UNIVERSITÄT

Universität der Bundeswehr Hamburg

Web Implementation of Dynamic Range Control Audio Applet

Research Project

by

Aritra Mazumdar

Start date:	01.04.2018
End date:	31.10.2018
Supervisor:	M.Sc. Etienne Gerat
Supervising Professor:	Prof. Dr.-Ing. habil. Udo Zölzer
Email:	aritra.mazumdar@tuhh.de

Abstract

Dynamic Range Control system is a part of the course, Digital Audio Signal Processing, taught by Prof. Zölzer at the Technische Universität Hamburg-Harburg. An audio applet performing the functions of a Dynamic Range Control is used as a demo in lecture to explain the concepts vividly.

This project deals with implementing a Dynamic Range Control application on web so that it can be easily accessed by the students of the university. The reason for the implementation is that the already existing applet is facing compatibility issue with the current browser versions and hence will cease to operate smoothly with even newer versions.

The application has been implemented by using Python on the server side and JavaScript on the client side, communicating with each other via a framework called Flask, with reference to existing MATLAB implementations. Four different modes have been implemented, namely Compressor, Expander, Limiter and Noise Gate with two different sub-modes in Compressor and a slightly advanced implementation in Noise Gate. Finally, the whole application has been hosted on server so as to make it access easy to anyone. The implementation followed all stages of Software Development Life Cycle within the project environment.

Statement

Hereby I do state that this work has been prepared by myself and with the help which is referred within this report.

Hamburg, 31^{st} October 2018

Contents

List of Figures

List of Symbols

$x(n)$	Input audio signal into Dynamic Range Control
D	Delay introduced in input audio signal
$g(n)$	Gain control signal
$x_{PEAK}(n)$	Peak value of the input signal calculated in the step of Level Measurement
$x_{RMS}(n)$	RMS value of the input signal calculated in the step of Level Measurement
$f(n)$	Gain Control Signal before smoothing
$y(n)$	Output audio signal from Dynamic Range Control
AT	Attack coefficient
RT	Release coefficient
TAV	Averaging coefficient
a	Pole Placement coefficient, specific to Noise Gate mode
T	Compressor Threshold
R	Compressor Ratio
$ltrhold$	Lower Threshold specific to Noise Gate mode
$utrhold$	Upper Threshold specific to Noise Gate mode
ht	Holding Time, specific to Noise Gate mode
$H(z)$	Transfer function of blocks in the block diagram
T_S	Sampling interval in Peak level Measurement
T_A	Sampling interval in RMS level Measurement
t_a	Attack time in milliseconds
t_r	Release time in milliseconds
t_M	Averaging time in milliseconds
k	Generic notation for either AT or RT

Chapter 1

Introduction

1.1 Dynamic Range Control

Dynamic range of a signal is defined as the logarithmic ratio of maximum to minimum signal amplitude. Its unit is decibels. In other words, it is the difference in decibels between the highest and lowest volume.

The dynamic range of a signal can be controlled using this application which we name as Dynamic Range Control (DRC). It is used to either amplify low level sounds or reduce the volume of high level sounds. One of its applications is in reproducing music or speech in a noisy environment like a car or a shopping center, where the dynamic range has to be matched with the background noise. If there is a high dynamic range in a content, then users have to adjust the volume of the audio repeatedly while playing. A few other examples will clarify real-time application of DRC. While a music concert is being recorded, the background noise and the sound level of the music, that is being played, keep constantly changing. While uploading the video in Youtube, comes in the role of DRC, which adjusts the dynamic range of the audio signal to the background noise. This application will change the audio content or in technical terms adapt the gain of the audio to achieve the above mentioned audio effect.

1.2 Context

As a part of the curriculum of the course, Digital Audio Signal Processing, taught by Prof. Udo Zölzer in Technische Universität Hamburg-Harburg, a Java Audio Applet is used as a demo in lecture for better understanding of students, of how DRC works and how it reacts by changing parameters. It gives an audio visual effect and hence enhances the concepts more than simply going through theoretical description. It has helped me in that matter for sure.

1.3 Project Goal

The current applet is getting out of date with every upcoming upgrade in browsers due to the incompatibility between the browser version and the Java version used in the applet. Also, it needs the applet to be installed on the system so as to access it anywhere and anytime. Moreover, moving desktop applications on web has become a recent trend to empower users with the ability to access the same at any point of time and at any place. So, the project deals with building a DRC audio application on web making use of JavaScript. The application will take audio and parameters as two of its essential inputs, plot and play the input, process the input according to the parameters and then eventually plot and play the output as well.

Dynamic Range Control has four modes of operation, namely Compressor, Limiter, Expander and Noise Gate, all of which will be explained in detail in the subsequent chapter. The project entails implementation of the Compressor mode. Additionally, the rest three modes of operation have also implemented , out of personal interest. The report consists of 5 more chapters after this current chapter of Introduction.

- Chapter 2 (Background and Related Work): This chapter gives a detailed description of Dynamic Range Control system and its operational logic, its already existing implementations and the problems in those corresponding implementations.
- Chapter 3 (Approach): This chapter narrates the user the journey to successful implementation of the application through the list of unsuccessful approaches, which formed the basis of Research and Development.
- Chapter 4 (Results): This chapter shows the final output of implementation and also does a relative comparison of the generated results with the desired results in MATLAB.
- Chapter 5 (Discussions): This chapter gives a brief idea about Software Development Life Cycle in relevance to the project, advantages and disadvantages of the implementation and few important user guidelines, which can be of help to the user.
- Chapter 6 (Conclusion and Future Work): This chapter concludes the report with explanation of possible future work that can be taken forward from here.

The next chapter will give you the preliminary information which is necessary to understand the project goal and the idea behind it.

Chapter 2

Background and Related Work

This chapter will guide you through the theoretical explanation of the project and the already existing implementations of the same. It will explain the problems present in those implementations and will clarify the objective of the project even more.

2.1 Theory

In this subsection, working principle of DRC and the individual components present in the block diagram of DRC will be explained in detail.

2.1.1 Block Diagram

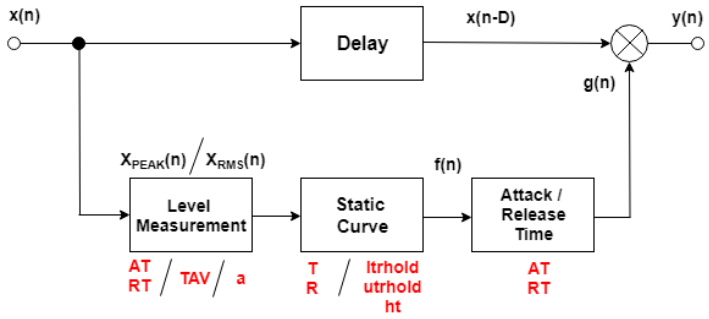

Figure 2.1: *Block Diagram [1]*

The Figure 2.1 above represents a basic Dynamic Range Control system. A signal, x(n), is fed as input and the output y(n) is obtained by multiplying a delayed input x(n-D) with the corresponding signal gain g(n), calculated from the sidechain of the block diagram.

$$y(n) = x(n - D) \cdot g(n) \tag{2.1}$$

In the equation (2.1) [1], D represents the delay in samples and g(n) is obtained through a three stage operation which includes Level Measurement, Static Curve Analysis and Gain Factor Smoothing, all of which will be explained in the subsequent subsection.

2.1.2 Working Principle

The first step in the gain factor calculation involves Level Measurement block.

Level Measurement

This block measures how rapidly the level of the input signal changes. The output of the measurement is fed to the next block which in turn influences its output as well. The coefficients controlling this block are Attack (AT) and Release (RT) coefficients or Averaging coefficient (TAV). While implementing one of the modes, Noise Gate, which will be explained in the next subsection, a Pole Placement coefficient (a) has been used in place of the above mentioned coefficients.

There are two ways of doing Level Measurement:

1. **Peak Measurement:** The current absolute value of the input input signal $|x(n)|$ is compared with the last value of the output signal $x_{PEAK}(n)$ from this block, and the difference between the two is multiplied by Attack coefficient (AT) if the former is greater than the later. The difference equation [1] is given by

$$x_{PEAK}(n) = (1 - AT)\cdot x_{PEAK}(n-1) + AT\cdot|x(n)| \tag{2.2}$$

The corresponding transfer function [1] is given by

$$H(z) = \frac{AT}{(1 - AT)\cdot z^{-}1} \tag{2.3}$$

If the absolute value of the input signal $|x(n)|$ is less than the last value of the output signal $x_{PEAK}(n)$ then the difference equation [1] is given by

$$x_{PEAK}(n) = (1 - RT)\cdot x_{PEAK}(n-1) \tag{2.4}$$

The corresponding transfer function [1] is given by

$$H(z) = \frac{1}{(1 - RT)\cdot z^{-}1} \tag{2.5}$$

The Attack and Release coefficients [1] are given by

$$AT = 1 - exp(\frac{-2.2\cdot T_S}{t_a/1000}) \tag{2.6}$$

$$RT = 1 - exp(\frac{-2.2\cdot T_S}{t_r/1000}) \tag{2.7}$$

T_s represents the sampling interval, t_a represents the attack time in milliseconds and t_r represents the release time in milliseconds.

2. **RMS Measurement:** The difference between the square of the current value of the input signal $x^2(n)$ and the square of the last value of the output signal $x^2_{RMS}(n)$ from the block is weighted by Averaging coefficient (TAV). The difference equation [1] is given by

$$x^2_{RMS}(n) = (1 - TAV)\cdot x^2_{RMS}(n-1) + TAV\cdot x^2(n) \tag{2.8}$$

The corresponding transfer function [1] is given by

$$H(z) = \frac{TAV}{(1 - TAV)\cdot z^{-}1} \tag{2.9}$$

The RMS value [1] is computed as

$$x_{RMS}(n) = \sqrt{\frac{1}{N} \cdot \sum_{i=0}^{N-1} x^2(n-i)} \tag{2.10}$$

The Averaging coefficient [1] is given by

$$AT = 1 - exp(\frac{-2.2 \cdot T_A}{t_M/1000}) \tag{2.11}$$

T_A represents the sampling interval and t_M represents the averaging time in milliseconds.

The level of the signal, measured in this step, is fed to the next block to calculate the gain reduction factor. The second step in the process of calculation is the Static Curve Analysis block.

Static Curve

This block is responsible for finding out the relation between the input signal level, measured in the previous block, and the weighting level or in other words the gain control factor. The coefficients for this block are Threshold (T) and Compression Ratio (R). Threshold is the level of the signal beyond which dynamic range shifting of the signal is desired to be executed. It is expressed in decibels. Compression Ratio is defined as the ratio of the change in input signal level to the change in output signal level for a specific range. Its value ranges from one to infinity.

A typical static curve is shown in Figures 2.2 a and b.

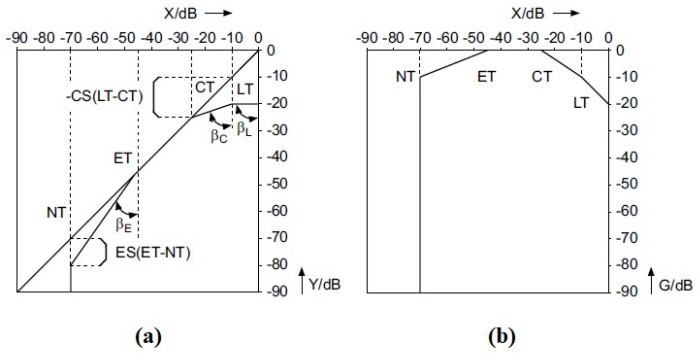

Figure 2.2: *Static Curves: (a) Input Level vs. Output Level, (b) Input Level vs. Gain Level [1]*

The Figure 2.2 a shows a static curve plot of input signal level in dB vs. output signal level in dB. The Figure 2.2 b shows a static curve plot of input signal level in dB vs. gain level in dB. Ratio determines the slope of the Static Curve, after signal crosses Threshold and also the mode of operation of the Dynamic Range Control system

- R = Infinity implies Limiter mode. With reference to the Figures 2.2 a and b, it is the region above the point, LT, in the Static Curve. LT represents Limiter Threshold. In this mode, the output signal level gets limited, when the input signal level exceeds the Limiter

Threshold.

- $R > 1$ implies Compressor mode. With reference to Figures 2.2 a and b, it is the region between the points ,CT and LT, in the Static Curve. CT represents Compressor Threshold. This mode reduces the dynamic range of the signal by mapping a change in input signal level to a smaller change in output signal level. It can be used in combination with a constant gain to make a signal louder by amplifying the quiet parts.
- $0 < R < 1$ implies Expander mode. With reference to the Figures 2.2 a and b, it is the region between the points ,NT and LT, in the Static Curve. NT represents Noise Gate Threshold. This mode increases the dynamic range of the signal by mapping a change in input signal level to a larger change in output signal level. It can be used to suppress noise by attenuating the quiet parts which mostly consists of noise.
- $R = 0$ implies Noise Gate mode. With reference to Figures 2.2 a and b, it is the region below the point, LT, in the Static Curve. It is basically used to suppress noise by attenuating the quiet parts which mostly consists of noise. In the implementation, two different thresholds, Lower Threshold (ltrhold) and Upper Threshold (utrhold), along with Holding Time (ht) have been used for an advanced implementation.

The gain factor calculated in this step is fed to the next block to smooth it.

The last step in gain factor calculation is the Gain Factor Smoothing block.

Gain Factor Smoothing

The purpose of this block is to smooth the gain control factor, calculated in the previous block. This block has two controlling factors, Attack (AT) and Release (RT) coefficients. There exists a hysteresis curve, which determines whether the gain control factor is in attack or release state and likewise gives the coefficient AT or RT. The difference equation [1] , in generalized form, is given by

$$g(n) = (1 - k) \cdot g(n - 1) + k \cdot f(n) \tag{2.12}$$

k is either AT or RT.

The corresponding transfer function [1] is given by

$$H(z) = \frac{k}{(1 - k) \cdot z^{-}1} \tag{2.13}$$

The current gain control factor value thus obtained is multiplied with the current input signal value delayed by D to achieve the target equation, which gives the instantaneous output value of the signal specified in (2.1). Similar operation is performed on each and ever sample value of the input signal to get its corresponding output value which when put together makes the output audio.

2.2 Related work

In this subsection, already existing implementations of Dynamic Range Control system across different platforms will be discussed. Following that, the problems in those implementations

will also be explained. An understanding about those problems will give the reader an idea about the project goal.

2.2.1 Java Applet

Figure 2.3 an audio applet built in Java, which does real time DRC operations on audio.

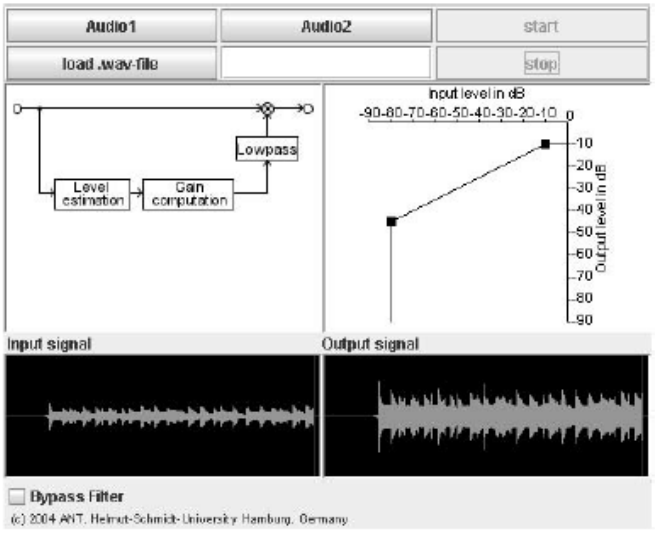

Figure 2.3: *Java Applet currently in use [1]*

Description

This applet has 5 buttons. The first two buttons give the user an option to choose among two sample audios. The third button loads the audio after selection. There are also buttons to start and stop audio processing. There is a block diagram of Dynamic Range Control. Finally, there is one dynamic Static Curve with two operational points. Those two operational points can be moved around for different input parameters and the corresponding effects can be visualized realtime from the out waveform plot. These operational points make it easy to switch between different modes of Dynamic Range Control. Also, there is a input waveform plot to help user to make a fair comparison between the input and the output audio. The applet gives an user an experience of combined audio visual effects of Dynamic Range Control system.

2.2.2 Implementation in MATLAB

There are already existing MATLAB implementations separately for each of the four modes of DRC. MATLAB can be downloaded in any system with the student credentials and those implementations can be used to visualize the audio effects.

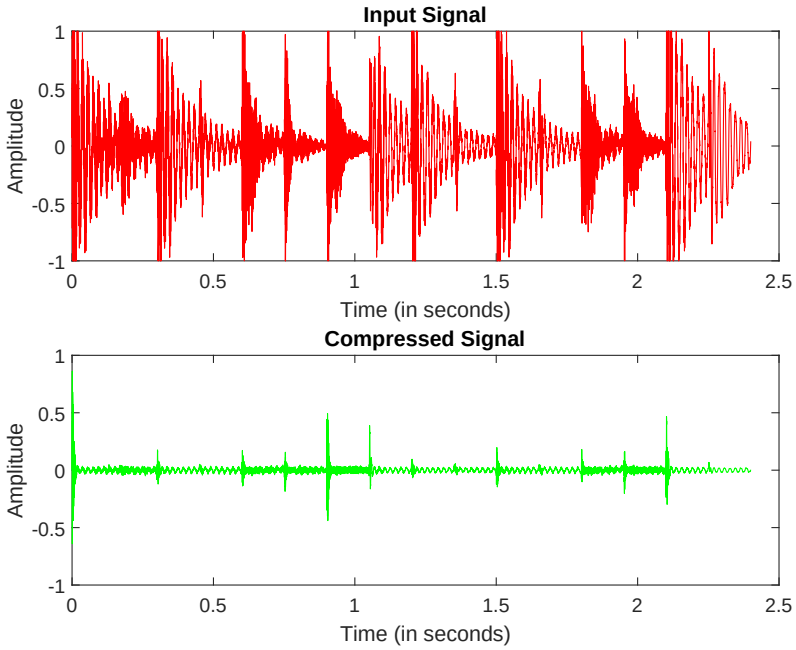

Figure 2.4: *Compression in MATLAB*

Description Figure 2.4 shows the waveform plot of input audio and that of the compressed one. The parameters used are: MATLAB lets the user to look into the individual sample values.

Parameter	Value
Ratio	20
Threshold	-40dB
Attack	0.01s
Release	0.0001s

MATLAB implementation gives user a more mathematical approach for doing an analysis on the sample values of audio before and after compression.

2.2.3 Web Implementation

Different Application Programming Interfaces (APIs) have implemented Compressors on web. Implementation in web has the advantage of accessing the same at any place and and any time without the need to install any additional software.

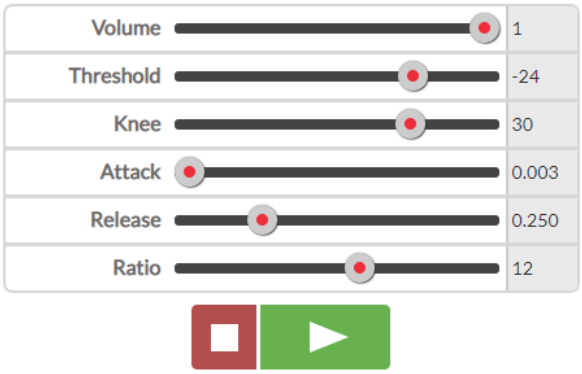

Figure 2.5: *Compression in Pizzicato [2]*

Description

Figure 2.5 shows implementation of Compressor using an API named as Pizzicato.js. The parameters can be adjusted dynamically with the help of the sliders.

2.2.4 Problems with the Existing Implementations

Each of the three implementations has some serious bottlenecks, as a result of which stemmed the need to design an application which will overcome the problems present in the existing implementations.

Applet

- Java needs to be installed on the system
- With newer versions of web browsers, there has been a compatibility issue between the version of browser and the version of Java used in the applet, thereby making it impossible to continue using the applet in future.

MATLAB

- MATLAB needs to be installed in the system.
- It is a heavy software, which takes a lot of time to download.
- It is not available as an open source.

API

- None of the APIs has all modes of Dynamic Range Control altogether.
- Audio APIs are not built properly and are not strong enough to perform advanced operations.

- Audio APIs do not support transfer of data from one API to another in an acceptable format for the latter, thereby making it impossible to implement a full fledged application in which different APIs need to communicate with each other.
- Output audio effects produced by APIs do not sound similar to the desired effect in MATLAB.
- Instead of working on the audio content and operating on the individual sample values, APIs act as a kind of equalizer to produce a similar effect. But when heard closely, there is quite a lot of difference.

The next chapter will narrate the entire course of implementation.

Chapter 3

Approach

This chapter will take you through to the actual journey, starting from scratch till the successful implementation of the application on web.

3.1 Initial Approaches

This subsection talks about all the initial approaches that were tried and the challenges faced in each of the approaches.

3.1.1 Using Web Audio API

The initial task that was set was to implement a compressor using both RMS and Peak ways of level measurement. The first approach that had been suggested for the implementation purpose was by using audio API based on JavaScript. Different APIs were tried in the process, namely Tone.js, Pizzicato.js, Wavesurfer.js and Loopslicer.js. Diving deep into each of the APIs will make the idea more clear.

Pizzicato.js

The first step taken towards building a compressor on web was to build an audio player that can load and play an audio on web. The first API that was used was Pizzicato.js [2].

The second step entailed a Compressor implementation with a single audio file and a static set of input parameters.

Both the steps were achieved using this API. The Figure 3.1 shows the first look of the application which can perform these two basic operations.

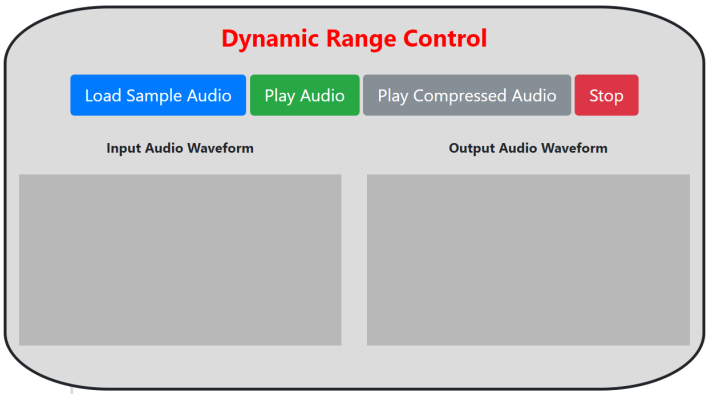

Figure 3.1: *Compressor using Pizzicato*

There were four buttons in this version of implementation. The first button loaded only one sample audio, name of which been was hardcoded. The second button played the audio. The third button played the compressed audio according to the hardcoded static set of parameters. The fourth button stops playing the audio.

The third step involved implementation of waveform visualization for both input and compressed audio. The absence of an inbuilt visualizer or the inflexibility of the API to allow building an own visualizer made it evident about the incapability of the API to perform advanced operations. So a more well built API was searched for, which can do waveform visualization and that leads us to the next subsection which talks about implementation using Tone.js.

Tone.js

In this method of implementation, everything started again from the scratch. Similar to Pizzicato, the first and the second steps were implemented using the API, Tone.js [3]. In this API also, there was no inbuilt visualizer. But a visualizer was successfully built, which could visualize both the input and the compressed audio. The second version of implementation looked as shown in Figure 3.2.

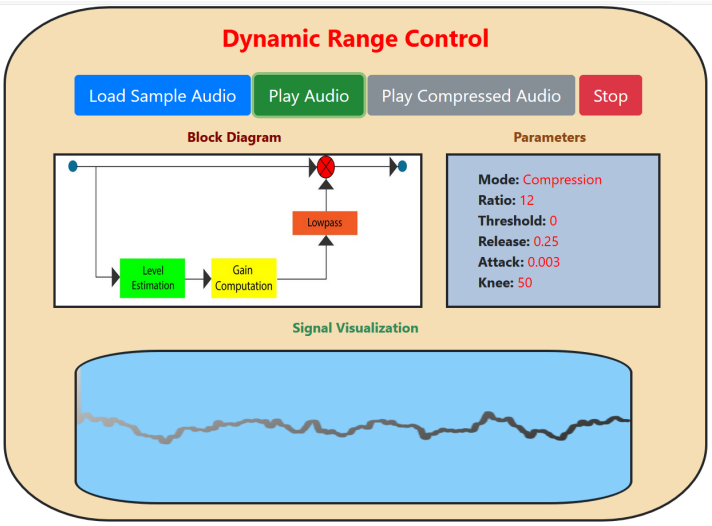

Figure 3.2: *Compressor using Tone*

There were again four buttons. Additionally, validations had been bound to each of these buttons, which are described in detail below.

- **First button:** Loaded the sample audio. If the audio was loaded again, it notified the user that audio had already been loaded.
- **Second button:** Played the loaded audio. If the sample audio had not been loaded, it notified the user to load the audio first.
- **Third Button:** Played the compressed audio. If the sample audio had not been loaded, it again notified the user similarly. Unlike in Pizzicato, in Tone, if this button was clicked more than once, it would compress the already compressed audio as many times as the

button was pressed. So an additional validation was applied to compress the audio only once, in spite of repeated clicks, and play the same.

- **Fourth Button:** Stopped playing the audio and also notified the user about no audio being played, if that was the case.

There were two static blocks, one of which showed the parameters used for compression, namely Ratio, Threshold, Release, Attack and Knee and the other one showed the block diagram behind the implementation. Knee was a new parameter present in this earlier version which implies how smoothly the Static Curve bends near the Threshold value.

There is a dynamic visualizer, which could be toggled between input audio and compressed audio visualization, accordingly as and when 'Play Audio' and 'Play Compressed Audio' were clicked.

The fourth step required plotting of static waveforms of input and compressed audio. There was neither an inbuilt plotting function in this API nor could it be built using this API. So the only feasible way was to look for an API that could extract the audio data from the Tone API object and plot the same in that API. That is when the third API, Wavesurfer.js, came into picture, which will be described in detail in the next subsection.

Wavesurfer.js

Implementation using this API, Wavesurfer.js [4], was an extension of the previous implementation using Tone.js. Wavesurfer.js has inbuilt function to plot static audio waveforms. The thrid version of the application using Wavesurfer.js looked like as is shown in Figure 3.3.

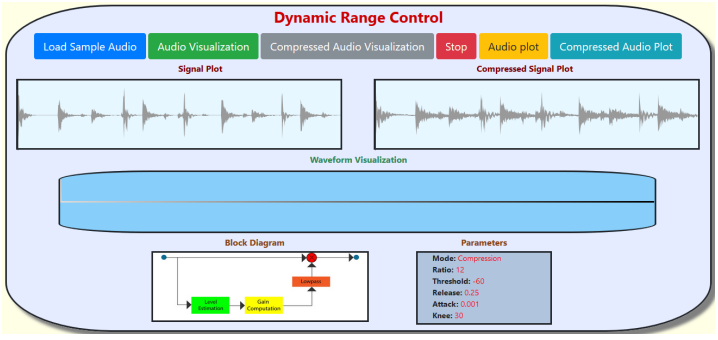

Figure 3.3: *Compressor using Tone and Wavesurfer*

The only new components in this version of the application are the 'Audio Plot' and 'Compressed Audio Plot' buttons and plotting canvases. The canvases were used to plot the static waveforms of input and compressed audio respectively.

The issue that came up with Wavesurfer.js was that, to plot a waveform, it accepted audio in either .wav format or from an URL or as a Blob (large binary object) or as a file object. After compression using Tone.js, the audio gets converted into an API object which could not neither be converted into Blob nor can it be loaded from an URL. Efforts were made to save the API object in a .wav file, an acceptable format that can be fed to Wavesurfer.js to plot the audio, but did not succeed. In the process, it was discovered that none of the APIs used above operates on the audio sample values. Instead all act like audio equalizers for any

uploaded audio. The plots shown in the image below had not been generated by feeding input and compressed audio from Tone.js to Wavesurfer.js. Instead Wavesurfer.js was used to plot the waveform of the input audio and the same audio, compressed externally, to show what was targeted to achieve. This takes us to the last trial made using an API, namely Loopslicer.js, described in next subsection.

Loopslicer.js

The reason why different types of APIs were tried for implementation, is that it depends on the API solely whether it allows the API object to convert into another API object or a generic JavaScript object. This implementation was not done from the scratch. Only the necessary part was taken out from the available implementation [5]. The look of the fourth version of implementation is shown in the Figure 3.4.

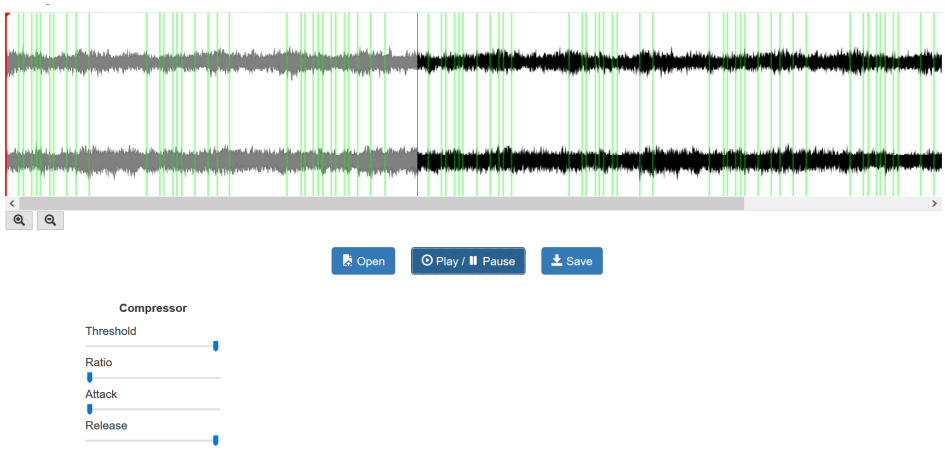

Figure 3.4: *Compressor using Loopslicer*

This API had an advantage of choosing input parameters dynamically by using sliders. With reference to the diagram, the second plot, which was thought of as the plot of output audio, turned out to be the plot for the second channel, present in stereo sounds.

Also, it was realized that with different input parameters, the audio effect is not at all same as that in MATLAB. When cross-checked for the previously used APIs, it turned out to be the same for them as well.

Finally, the plan of implementing Compressor using API was discarded due to the compatibility issue of object types across different APIs and also due to impossibility of having a complete control over the audio processing in sample by sample manner. This brings us to the next way of implementation, using plain Javascript, without using any API.

3.1.2 JavaScript

The new target of implementation, using JavaScript, was to implement compression on individual sample values of the audio instead of faulty real-time compression using an API. Visualizer was dropped for this version and instead static plot of waveforms was given more priority. Implementation again started from the beginning.

To implement a compressor in JavaScript, an object of type AudioContext [6] was created. AudioContext represents the sound system of computer and is the main object used for creating and managing audio. Following that, an audio buffer was created for that AudioContext object. This audio buffer was used to load a sample audio from the computer. To implement compression on the selected audio, audio buffer was required to be converted into an array buffer, which stored the sample values in its indices. After implementing compression on the individual sample values, array buffer was required to be converted back to audio buffer, so as to be saved back into a .wav file, which is needed to plot using Wavesurfer.js. But this was not made possible, as conversion from audio buffer into array buffer was a one way track. Hence, instead of trying to plot the audio, an easier target of first implementing a Vu Meter was set. This fifth version of implementation looked like as shown in Figure 3.5

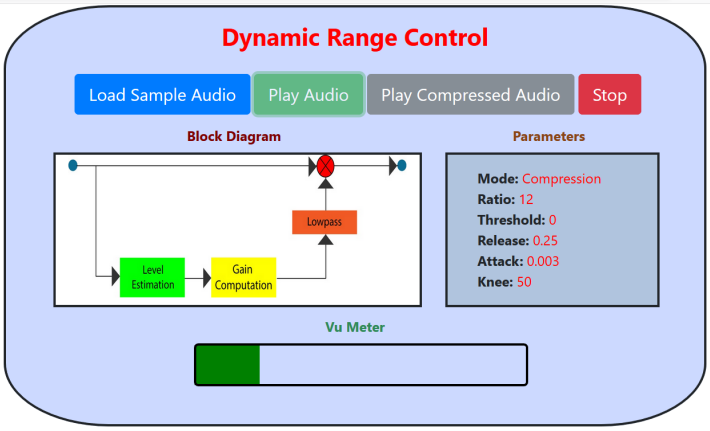

Figure 3.5: *Compressor using Vanilla Javascript*

The only new component in this version was the VU Meter. Rest components remained the same. When "Play Audio" or "Play Compressed Audio" buttons were pressed, the Vu Meter was supposed to dynamically visualize the sound level of individual samples. The Vu Meter could visualize both the input and the compressed audio by fetching the values from the array buffer before and after compression. But it could only play the input audio, by reading the audio buffer before compression, but not the compressed one, as the problem of conversion, from array buffer back to audio buffer, after compression, was still there.

It was subsequently realized that implementation of a complex audio operation like compression with an effect, similar to that in MATLAB, could not be achieved through only client side scripting languages like JavaScript. Scripting languages are strong enough to handle a normal website request from an user, like navigation to a different page or opening up a dropdown, but not robust enough to deal with heavy audio operations with the available resources. Hence, it was decided to involve both client side and server side in the project to achieve the goal. For server side, the first thing that comes in mind is Java Web, which was used in the sixth version of implementation and is described in detail in the next subsection.

3.1.3 Using Java Web

The process of implementation of compression using both client and server can be classified in four broad steps

- **Step 1:** Implementing compression in server, using Java Web
- **Step 2:** Receiving audio and input parameters from user in client side, using JavaScript
- **Step 3:** Setting up a framework to enable communication between client and server
- **Step 4:** Passing the audio and the input parameters from client to server by virtue of the framework, implementing the compression in server and feeding the compressed audio back to the client for plotting and playing the compressed audio along with the input audio in the browser

The process of implementation started with the first step. But implementation using Java Web met with the same bottleneck as that in JavaScript. The audio could be converted into array of sample values and operations could be executed on the sample values, but the array of compressed sample values could not be converted back into a .wav audio file. Hence this approach also did not materialize.

A last try was still left to be made with the most powerful server side programming language, Python, which has mathematical libraries suitable for audio. This brings us to the next section of this chapter which talks about the successful approach using a combination of JavaScript and Python.

3.2 Successful Approach

This subsection narrates the process followed in implementing the final version of the application.

3.2.1 Application Details

The seventh and the final version of the application looks as shown in the Figure 3.6.

GUI

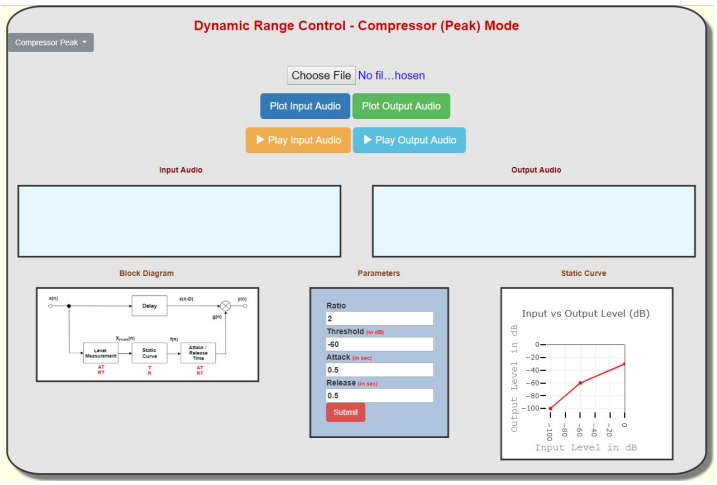

Figure 3.6: *Dynamic Range Control*

Design

The Figure 3.7 below represents the complete design of the application in terms of User Modelling Language (UML).

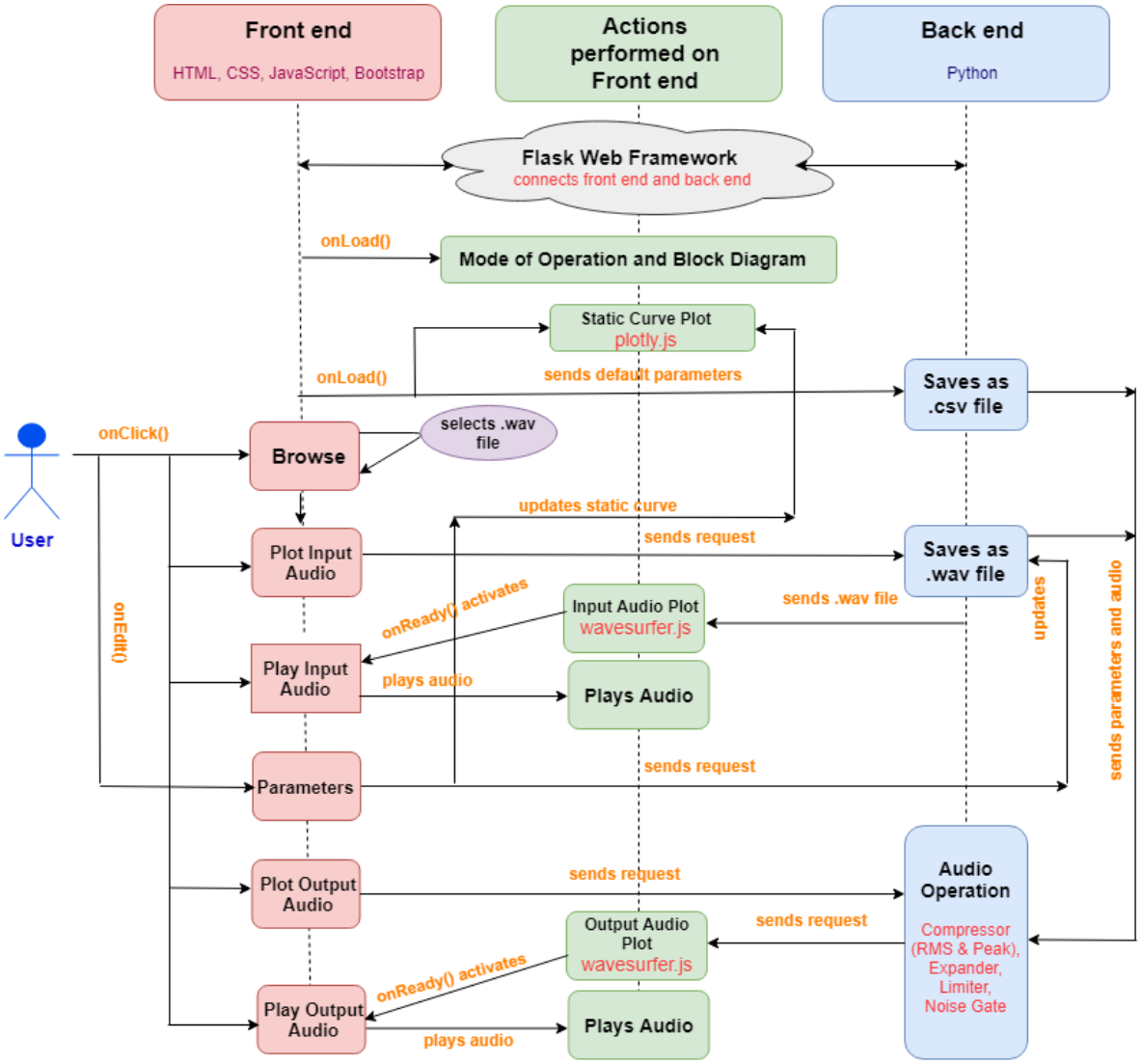

Figure 3.7: *UML Diagram*

Description

The UML diagram is split into three sections, functions available in the client, also termed as front end and the operations happening in the server, also termed as back end.

The Programming Languages, libraries, APIs and Frameworks used in the final version of the application are listed below

- **HTML** - for structuring the application [7]
- **CSS** - for the look and feel [7]

- **JavaScript** - for the front end logic [7]
    1. **jQuery** - JavaScript library for simplifying front end scripting of HTML [7]
    2. **Ajax** - to enable the application to send and receive information from the back end [7]
- **Bootstrap** - front end framework, containing HTML and CSS templates, for more polished look and feel [8]
- **Wavesurfer.js** - API for plotting input and audio waveforms on canvas [4]
- **Plotly.js** - API for plotting the Static Curve [9]
- **Python** - for back end logic to implement the desired audio operations [10] [11] [12]
- **Flask** - micro web framework which facilitates to and fro communication between front end and back end [13]

With reference to the UML Diagram in Figure 3.7, it is clear that Flask has been used as a framework for communication between client side and server side. The whole working principle will be expressed in steps below.

- **Step 1:** On loading the application, one out of the five modes of operation of DRC gets loaded. It is possible to switch between the modes with the help of a dropdown menu present in the application and can be seen in the GUI.
- **Step 2:** The block diagram also gets loaded according to the mode of operation.
- **Step 3:** While the webpage is loaded, the default set of parameters are sent to the back end in JavaScript Object Notation (JSON) format as an HTTP-POST request and is saved in a .csv file in the python server. The type of input parameters for this application differ from one mode to another.
- **Step 4:** A section of the parameters also sets the default Static Curve for the default parameters.
- **Step 5:** An audio .wav file is browsed from the computer.
- **Step 6:** User clicks on "Plot Input Audio" which sends the audio in JSON format as an HTTP-POST request and gets saved in the server as a .wav file. The reason for saving the .wav file in the Python server is that the API Wavesurfer.js can load the input .wav file from the URL of the stored location in the server by an HTTP-GET request to show its static plot in the canvas.
- **Step 7:** Once the input waveform plot is generated in the canvas, the same can be played by clicking on "Play Input Audio". A slider moves along when the audio gets played along the whole length of audio.
- **Step 8:** Other than the default parameters that are being sent in the beginning, user can edit the parameters as required and when clicked on submit, the parameters get updated in the .csv in the same way as mentioned above.

- **Step 9:** On clicking on "Plot Output Audio", it sends an HTTP-GET request to the Python server, which starts the required audio operations, according to the set mode, on the audio file based on the sent parameters. After the completion of the audio operations, the file is saved in the server as a .wav file and Wavesurfer.js accesses it in the front end to plot it in the canvas in the same way as mentioned before.
- **Step 10:** The output audio can be played in the same way as mentioned before, by clicking on the "Play Output Audio" button.

The basic difference between an HTTP-GET and an HTTP-POST request, in this context, is that the former is used to request data from the server whereas the latter is used to send or update data in the server.

These 10 steps explain the operations performed by the application sequentially. In the next subsection, each of the components present in the application will be explained in detail.

Building Blocks

Each and every building block in the GUI, as seen in Figure 3.7, has front end functionalities or validations associated with it.

- **Browse:**
  1. This field allows uploading only .wav files and gives an alert message to the user stating "Invalid File type! Only .wav files are allowed.", if any file with a different extension, including .mp3, is uploaded.
  2. There is a validation regarding name of audio that gets uploaded. Allowed name of audio can have alphabets from a to z in both upper and lower cases, numbers, round brackets, hyphen, underscore and dot.
  3. There is a validation for the size of the upload file as well. It is 1MB for all modes except Noise Gate where it is only 500KB. The reason for this file size validation is to cut down the operation time while generating the output plot or to avoid the Python server to go to a state of no response due to overload.
- **Plot Input Audio:** If the button is pressed before uploading an audio, it gives an alert message to the user stating "Please browse an audio first.", in order to make the application more intuitive.
- **Play Input Audio:** If the button is pressed before plotting the input audio, then user gets a message saying "Please plot input audio first.". The reason for this is that a slider appears after the plot is generated for the input audio and it moves along the plot length when the audio is played, which is not possible without plot generation.
- **Plot Output Audio:**
  1. If this button is pressed before plotting the input audio, then user again gets the same message "Please plot input audio first.". The reason for this is that both input audio file and parameters are required for audio operations. The default parameters are sent to back end onload. So only when the "Plot Input Audio" button is pressed first, the audio file is sent from JavaScript to Python.

2. If the "Plot Input Audio" button has been pressed before pressing this button, it gives a popover message saying that "This might take a few seconds! Please wait.", to indicate the user to wait patiently till the audio operations get completed.

- **Play Input Audio:** If the button is pressed before plotting the output audio, then user gets a message saying "Please plot input audio first." because of the same reason mentioned in **Play Input Audio**.

- **Block Diagram:**

  1. On switching between different modes, the block diagram gets loaded with different sets of parameters.
  2. It has zoom in and out feature on hover in and out to give user a better view of the parameters present in diagram.

- **Parameters:**

  1. **Ratio:**
     (a) For both modes of Compressor, Ratio has a range from 2 to 1000. A Ratio of 1 will not have any influence on the audio and a Ratio beyond 1000 will have similar effect as that of Limiter. For Expander, Ratio ranges from .001 to 1. Ratio does not influence the other two modes and is not an input parameter in those modes.
     (b) If a number is entered which is below or above the limits, the entry is reset to the lower and higher limits respectively along with alerting user with a suitable message.
     (c) Only allowed type of entries in this field are numbers from 0 to 9 and dot for fractional numbers.
     (d) There is also a validation for comparison of fractional numbers. e.g., Range of Expander is from .001 to 1. So if .0001 is entered as ratio, it will be reset to .001.
     (e) If Ratio is entered in a wrong way, user gets an alert message saying 'Please enter all the Parameters correctly!!', when user tries to generate the output audio plot. e.g., a Ratio of 0.00.1 will give this alert message.
     (f) This field cannot be kept empty. It gives an alert message to the user stating the same and also generates an error message, if kept empty while trying to plot output audio. '
  2. **Threshold:**
     (a) The range for this field is from 0dB to -100dB. This field exists for all modes. In Noise Gate it is split into Lower and Upper Thresholds.
     (b) Validations mentioned in points b, d, e and f for Ratio also exist in this field. With reference to point c from Ratio, additionally negative sign has been allowed for threshold.
  3. **Lower and Upper Threshold:**
     (a) These 2 fields exist in only Noise Gate mode. All the validations which are there for the field Threshold, exist for both of these fields in here.
     (b) Apart from the already mentioned out of range validations, in range validations also exist. If the Lower Threshold is set to a value higher than Upper Threshold, it is set to the same value as that of Upper Threshold along with giving an indicative message to the user.

(c) If the Upper Threshold is set to a value lower than Lower Threshold, it is set to the same value as that of Upper Threshold along with giving a similar message to the user.

4. **Attack and Release:**
   (a) These 2 fields exist in all the modes and is expressed in seconds. Validations mentioned in points c, e and f in Ratio have also been implemented for this field.
5. **Holding Time:**
   (a) This field exists for Noise Gate mode only. All the validations mentioned in Ratio are also present in this field. The only difference is its range is from 0 to 0.5 and is expressed in seconds.
6. **Pole Placement:**
   (a) This field is present only in Noise Gate and has a set range from 0 to 1. This coefficient represents the range (0-1) of z value for a stable filter, beyond which filter becomes unstable. All the validations mentioned in Ratio are also present in this field.

- **Static Curve:**
  1. The API Plotly.js is used to generate the plot of Static Curve. Components of the plot like legends and axes have been customized as required.
  2. The parameters influencing the plot differ from one mode to another. Those are Ratio and Threshold for Compressor and Expander modes, only Threshold for Limiter mode and Lower and Upper threshold for Noise Gate mode.

This final approach worked out eventually. But this approach did not come off in one go. There were many hurdles to it as well which will be described in the next subsection.

3.2.2 Approach

This subsection will narrate different phases of the successful approach from the beginning till the end.

- **Phase 1:** In this phase, a compressor was implemented in Python with RMS way of level measurement.
  1. Python and Pycharm 64 bit were installed. To import a .wav file, to read and write the file from and back into an array and to do mathematical audio operations on the array elements, important Python libraries, mainly scipy.io and numpy.io, were installed.
  2. At first a .wav file was loaded into an array of sample values and writen back the same into a .wav file.
  3. Following that, compressor was implemented in Python in RMS way in a similar way as implemented in MATLAB [14].
  4. When the audio file, compressed in Python, was analyzed in MATLAB, the compressed audio from Python had twice the number of channels and samples compared to that of the input audio.

5. A stereo audio is converted into a mono audio and the above mentioned process is repeated. This time the signals were found to be similar.

- **Phase 2:** In this phase, connection was established between JavaScript and Python.

  1. To set up a connection between client and server, a framework in Python, called Flask, was installed in the server.
  2. Then .wav audio was sent and uploaded in the Python server by using `https://www.baqend.com/`.
  3. The next step was to send the audio from client to server. The required front end code was implemented in Visual Studio. A complete new framework, Angular JavaScript, was used at first, but it failed due to incompatibility in port number between Angular JS and Python server.

- **Phase 3:** Compression was implemented using JavaScript and Python.

  1. A new approach was used using jQuery call to make an HTTP-POST request to send a .wav file in JavaScript Object Notation (JSON) format to the Python server.
  2. Then file was read from the user and the compressed audio was saved in the server.
  3. By making an HTTP-GET request through jQuery call, Wavesurfer.js accessed the saved file from the server through the URL of the stored location and could plot the compressed audio on the front end.

- **Phase 4:** Remaining aspects of compression were also implemented.

  1. The next target was to plot the original audio as well. It was achieved in a similar manner, used for output audio, by saving the input audio file in the Python server.
  2. Buttons and their functionalities were re-aligned. On pressing the "Plot Original Audio" button, the audio would be sent to the server and then plotted in the canvas. On pressing the "Plot Compressed Audio" button, the compressed audio will be plotted in the canvas.
  3. Then the Parameters section was modified, to enable entering dynamically by the user and to be sent to the Python server and saved as a .csv file, when clicked on the "Submit" button.
  4. In this step, instead of using fixed sampling frequency, it was dynamically read from the browsed audio.
  5. Till now stereo audio was being converted to mono before performing compression. Now a logic was implemented in Python to compress both stereo and mono audios in different ways.

- **Phase 5:** Major show stoppers in the compression process were detected and taken care of in this phase.

  1. On clicking on plot buttons for either original and compressed audio a problem of caching in the browser was spotted. Cache control was implemented in both client and server which solved the problem.
  2. When the audio compressed in Python was compared with the audio compressed in MATLAB in logarithmic scale, an issue of scaling was noticed. The array of input audio samples were scaled accordingly and the plots were found to be almost identical but not same.

3. A difference in trail was observed and analyzed to be due to initialization and due to the delay included in the input buffer in MATLAB. It was fixed by treating the samples in the beginning differently and by adjusting the delay in Python with the help of a function called enumerate respectively.

- **Phase 6:** GUI was completed for RMS Compression.

  1. Block Diagram was drawn in Adobe Illustrator for this particular mode with required set of coefficients. Zoom, on hover, effect was implemented.
  2. Static Curve was implemented using Plotly.js. Initially the logic for the plot of Static Curve was implemented using fixed values and eventually using values sent by user. Front end validations for all buttons, parameters and Static Curve were implemented.
  3. Static Curve was modified from the default one in terms of legends and axes. Initially two traces were plotted, one for Compressor and the other as default. Later on fixed axes visualization was implemented which ended the need for the default trace.
  4. Buttons and plots were renamed.

- **Phase 7:** In this phase, Compressor in Peak way, Expander and Limiter were implemented using a similar approach of that in MATLAB [14].

  1. In this step, Compressor, in Peak Mode, was implemented in Python, block diagram was updated and a new logic for Static Curve was implemented in JavaScript. Rest of the implementation setup remained same. The compressed audio was compared again with the one from MATLAB for the same set of parameters. There was an initialization issue in this mode as well. Later it was realized that it is due to the difference in starting index. Needful changes were done in the code and the plots were similar.
  2. In this step, Expander was implemented in Python, range of parameters were modified, block diagram was updated and a new logic for Static Curve was implemented in JavaScript. Rest of the implementation setup remained same. There was an initialization issue, which was again due to the difference in starting index, as mentioned before and which was fixed similarly.
  3. Similar set of changes were implemented also for Limiter. Then the Limiter code was modified in Python with reference to the code present in the paperback version which is different form the one available online. When the output audio plots from Python and MATLAB were checked there was an initialization error in this mode as well similar to Compressor in RMS way and was fixed in the same way as that in RMS mode.

- **Phase 8:** This marks the last phase of implementation. In this phase, Noise Gate was implemented with reference to MATLAB implementation [14].

  1. Similar set of changes were implemented also for Noise Gate. Also, additional parameters were implemented.
  2. While implementing this mode in Python, there was a need to access a previous index of sample values of input audio and at the same time index was required to be initialized from an index, different from the first index. It was fixed using append technique in almost sections of the code. On comparing the audio plots, an initialization error was spotted. It was fixed by treating a variable differently as it was not getting incremented in regular way. Then the audio plots were found identical. This brings an end to the design story from beginning to completion.

The next subsection is about how both the client and python server have been hosted on free servers and how connection between the same has been modified from a local to local to server to server.

3.2.3 Hosting

Hosting is process in which a website, constructed locally on a computer, can be put on the internet so as to be accessed globally by anyone. This final step was also taken up out of own interest so as to materialize access anytime and anywhere goal of the project. Hosting of the complete website was achieved in 3 major phases which has been described in detail below.

- **Phase 1:** Initially to host the complete website, a profile was created in a website, `https://www.ecowebhosting.co.uk/`. A month's subscription was purchased. Then the rest process will be explained in the subsequent steps mentioned below.

  1. A hosting package was created by selecting a domain name.
  2. In order to transfer, add or delete files on the server, a protocol named File Transfer Protocol (FTP) was used. There are several programs which support FTP, out of which Fire FTP was chosen. Fire FTP plugin was added in Mozilla Firefox.
  3. Following creation of hosting package, a mail containing login details of the package, was received.
  4. Using the credentials, the hosting package was accessed through Manage Hosting Package option. To activate the add-on, the browser was restarted and FTP credentials were to be used.
  5. After a month the hosting package expired. It was realized that extending subscription every month is not a good idea and also a user friendly and free hosting environment will be much easier to deal with.

- **Phase 2:** It was decided to use separate hosting environments from front end and back end codes. At first a free hosting environment for back end was found. The following steps were taken to host the back end code.

  1. A service called pythonanywhere, virtual host based on Python, was used. By setting up an account in the website, `https://www.pythonanywhere.com`, the utilities available could be accessed. Different operations could be done under the available tabs of Consoles, Files, Web, Schedule and Databases.
  2. In the next step, a virtual environment was created through the Consoles tab. Flask framework and all the required dependencies in the Flask app were installed through the bash console.
  3. All the files needed to run the Flask app on the server were uploaded in the Files Tab.
  4. A new web app was created and a framework was selected from the list of available frameworks under the web tab. For this project, a manual configuration option and a Python version of 3.6 were selected.
  5. After submitting, a page appeared which displayed configuration for the website. In there, a virtual environment path had to be added and a WSGI configuration file had to be modified so that the flask app file can be accessed by the front end.

This completes hosting of the back end code. Now front end code needs to be hosted to make the complete application available on the internet. The steps to do so have been explained in the next phase.

- **Phase 3:** In the last phase the front end was hosted using a free hosting website, `https://www.000webhost.com/` . Unlike the sequence of steps mentioned in Phase 1, hosting the front end code was much easier this time. The steps mentioned below describes how it was done.

  1. At first, an account was created in the above mentioned website with required credentials.
  2. Using the credentials, a hosting environment with name of dynamicrangecontrol was created. Once a hosting environment was created, there was no need to download or install any thing else, as this JavaScript friendly hosting environment had provided a complete and necessary structure to host the frontend files.
  3. The JavaScript files, image files and JavaScript libraries were uploaded in the environment, created under the above mentioned hosting name.
  4. Small changes had to be made in code wherever a file or a response was needed to be accessed from front end to back end. The reason for this is that now the setup has moved from local to the internet, thereby changing the access path.
  5. Finally, the website can be accessed anywhere and anytime and operations can be performed, using the URL, `http://dynamicrangecontrol.000webhostapp.com/`. Some additional latency will prevail due to both the hosting environments being free and hence being less maintained.

This brings an end in terms of implementation of the project. To understand how much satisfying the results are, a thorough investigation of the outputs need to be done. The results of investigation have been presented in the next section of the report.

Chapter 4

Results

4.1 Output

The Figure 3.6 showed how the application looks like in general. The Figure 4.1 shows how the application looks like after having plotted the input waveform and the corresponding output waveform for the given set of parameters.

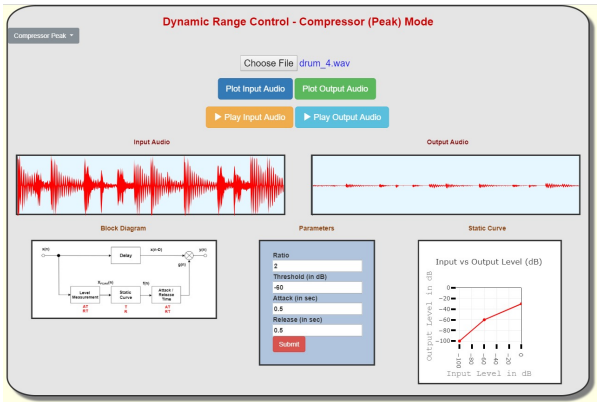

Figure 4.1: *GUI with Signal Plots*

In the next subsection, output signals will be generated for each and every mode of operation and compared with the same generated in MATLAB.

4.2 Evaluation

A way to perform analysis of the generated results is to compare the output audio signals generated in MATLAB and in Python in dB. In order to do so, at first the audio operation is performed in Python for a set of input of parameters and afterwards saved as a .wav file. Now, the corresponding audio operation for similar input parameters is performed in MATLAB. Finally, envelope of the difference signal is plotted in dB with respect to time. There are differences in between MATLAB and Python in terms of number format representation and variable storage. This creates a minute difference between the output signals, obtained from the two different platforms, although same parameters have been used to generate the output. The difference is not visible in linear scale, but in logarithmic scale. A difference value of -80dB or less on logarithmic scale has been set as a benchmark for correct implementation, as information contained below that level is not perceptible or even audible. The above mentioned process of analysis has been performed for different set of input parameters and for every mode of operation. The sample audio that has been used is `drum_4.wav`.

4.2.1 Compressor (Peak) Mode

The parameters used for the generation of the plots are:

Parameter	Value
Ratio	20
Threshold	-40dB
Attack	0.1s
Release	0.001s

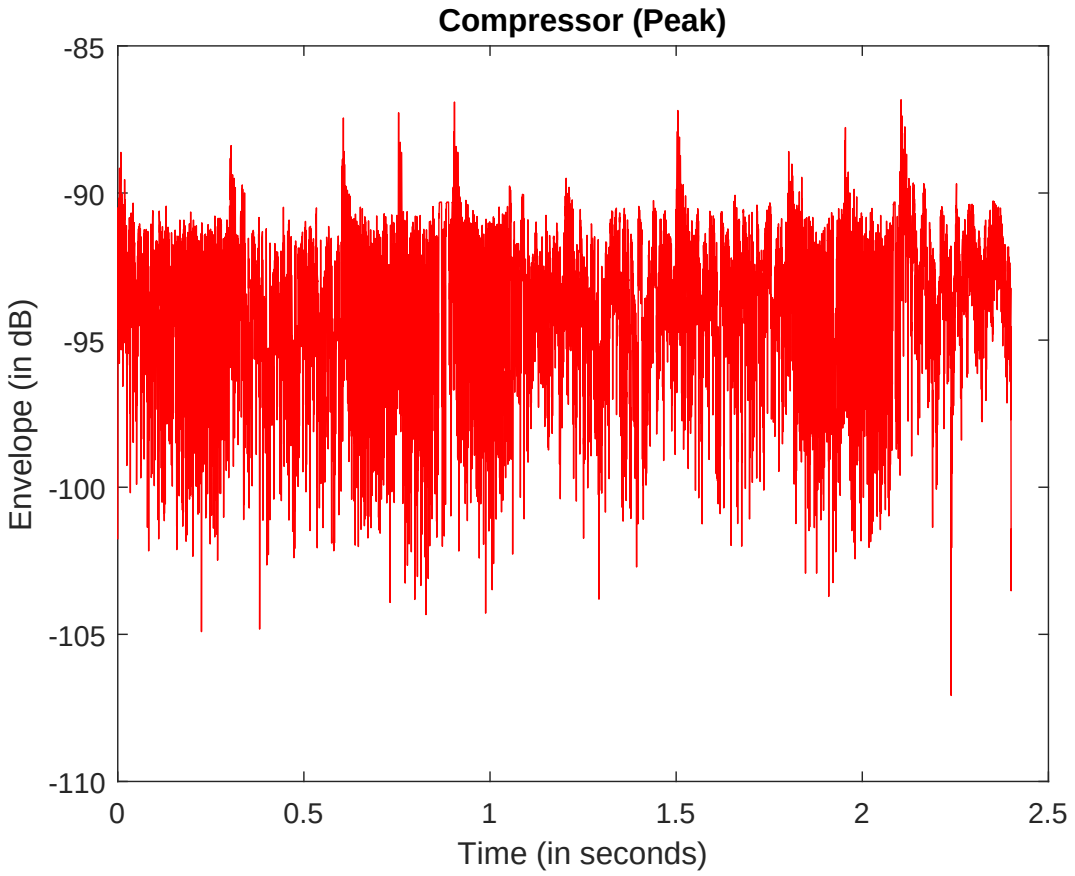

Figure 4.2: *Compressor Peak Plot in dB*

- **Observation:** Figure 4.2 shows that envelope of difference signal stays below -80dB for the complete signal and hence passes the set benchmark.

- **Conclusion:** Output signals, obtained from MATLAB and Python, are similar. Hence, Compressor (Peak Mode) implementation is successful.

4.2.2 Compressor (RMS) Mode

The parameters used for the generation of the plots are:

Parameter	Value
Ratio	10
Threshold	-50dB
Attack	0.01s
Release	0.0001s

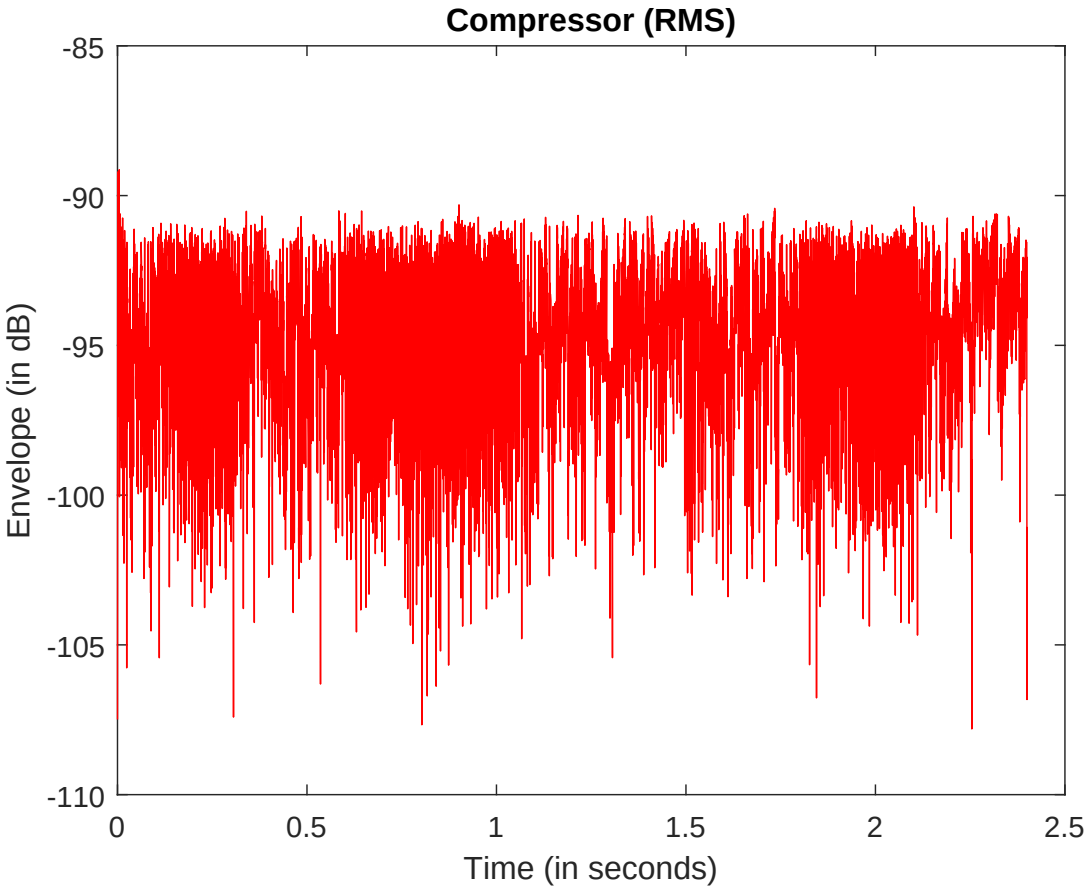

Figure 4.3: *Compressor RMS Plot in dB*

- **Observation:** Figure 4.3 shows that envelope of difference signal stays below -80dB for the complete signal and hence passes the set benchmark.

- **Conclusion:** Output signals, obtained from MATLAB and Python, are similar. Hence, Compressor (RMS Mode) implementation is successful.

4.2.3 Expander Mode

The parameters used for the generation of the plots are:

Parameter	Value
Ratio	0.5
Threshold	-10dB
Attack	0.5s
Release	0.5s

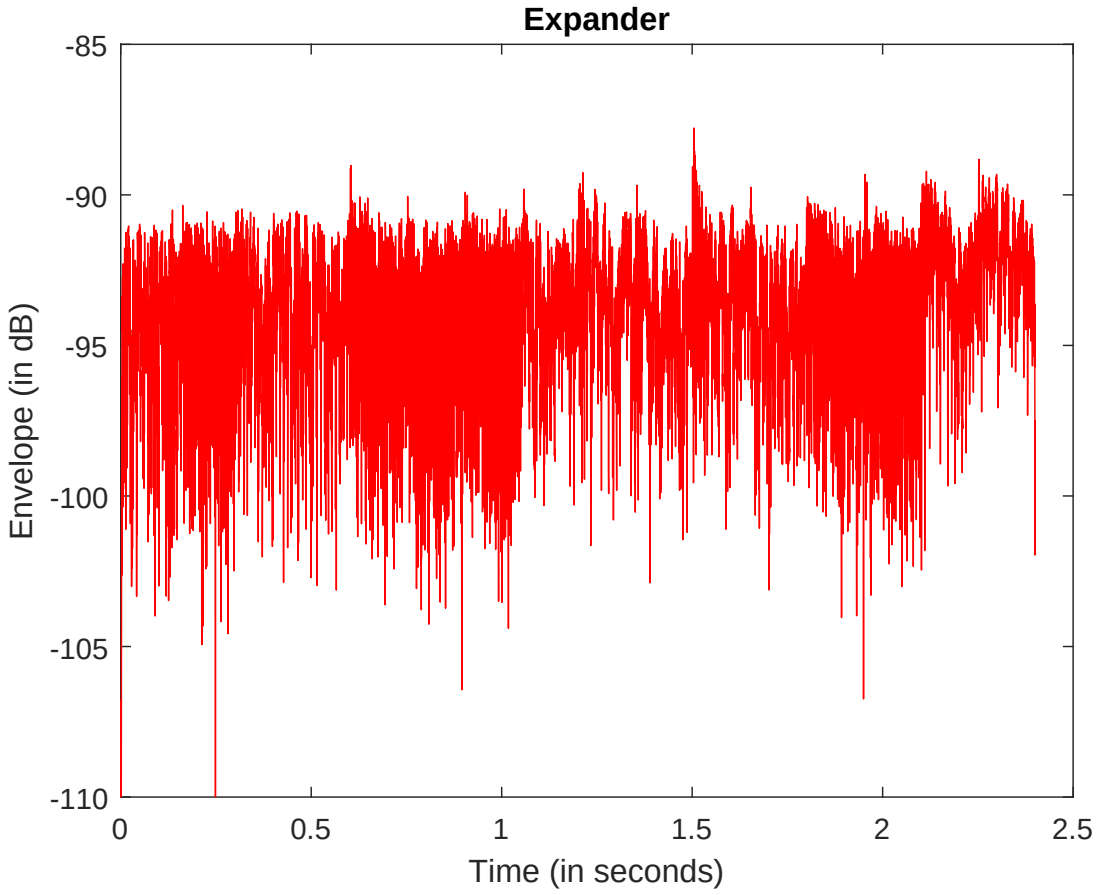

Figure 4.4: *Expander Plot in dB*

- **Observation:** Figure 4.4 shows that envelope of difference signal stays below -80dB for the complete signal and hence passes the set benchmark.

- **Conclusion:** Output signals, obtained from MATLAB and Python, are similar. Hence, Expander implementation is successful.

4.2.4 Limiter Mode

The parameters used for the generation of the plots are:

Parameter	Value
Threshold	-55dB
Attack	0.001s
Release	0.0005s

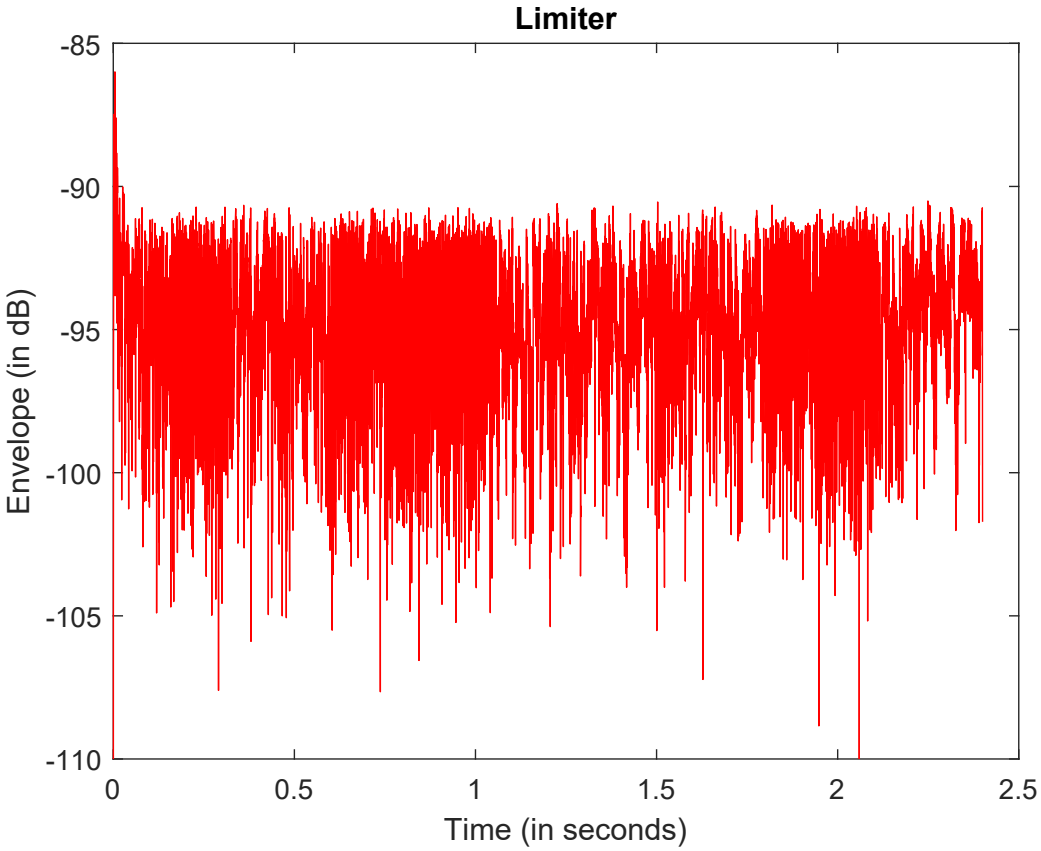

Figure 4.5: *Limiter Plot in dB*

- **Observation:** Figure 4.5 shows that envelope of difference signal stays below -80dB for the complete signal and hence passes the set benchmark.

- **Conclusion:** Output signals, obtained from MATLAB and Python, are similar. Hence, Limiter implementation is successful.

4.2.5 Noise Gate Mode

The parameters used for the generation of the plots are:

Parameter	Value
Lower Threshold	-30dB
Upper Threshold	-25dB
Attack	0.05s
Release	0.00001s
Hold Time	0.0001s
Pole Placement	0.3

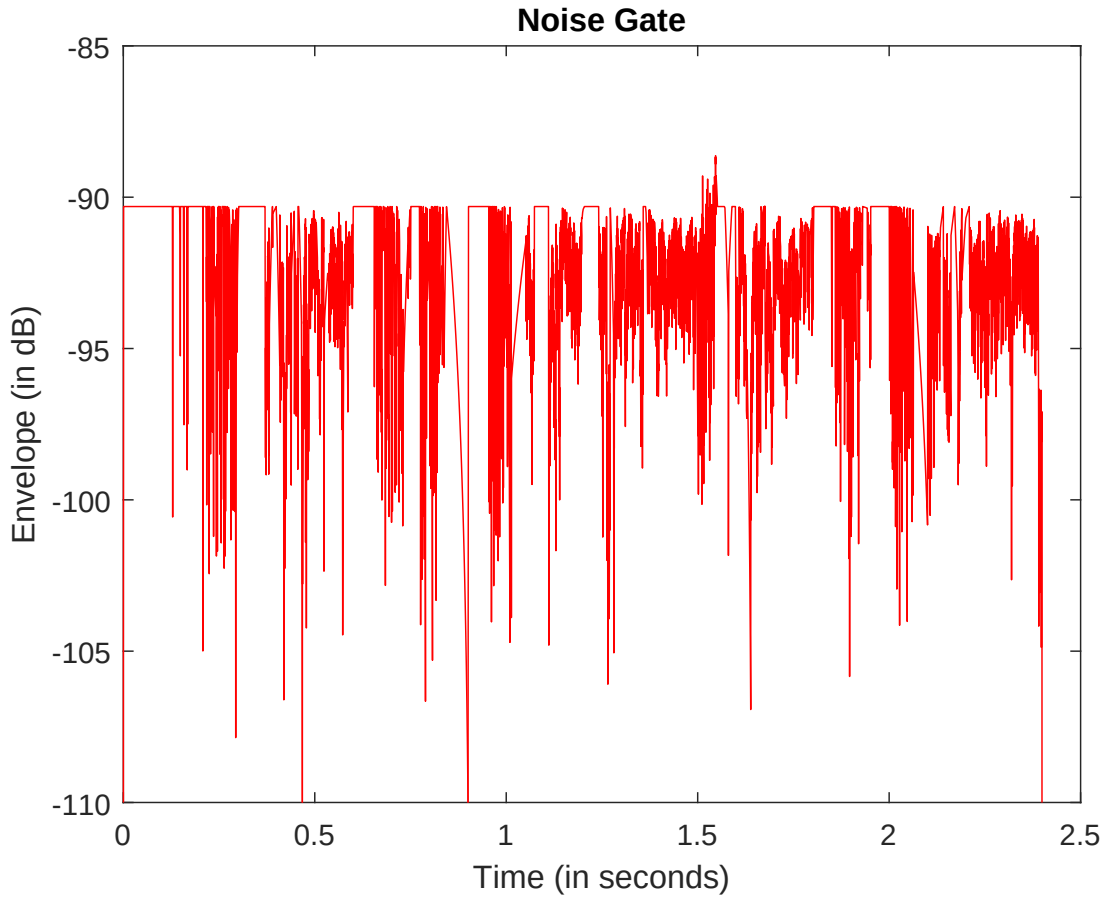

Figure 4.6: *Noise Gate Plot in dB*

- **Observation:** Figure 4.6 shows that envelope of difference signal stays below -80dB for the complete signal and hence passes the set benchmark.
- **Conclusion:** Output signals, obtained from MATLAB and Python, are similar. Hence, Noise Gate implementation is successful.

The results clearly indicate that what was expected from the audio operations in MATLAB has been achieved through similar operations in combination of JavaScript and Python.

4.2.6 Summary

A summary of evaluation is presented in this subsection. The previous section provided results based on envelope plots. This subsection provides results based on mathematical calculations with respect to a different metric, Root Mean Square Error.

Mode	Parameters	RMS Error
Compressor (Peak)	**Ratio:** 20, **Threshold:** -40dB, **Attack:** 0.1s, **Release:** 0.001s	4.0030e-07
Compressor (RMS)	**Ratio:** 10, **Threshold:** -50dB, **Attack:** 0.01s, **Release:** 0.0001s	2.6528e-07
Expander	**Ratio:** 0.5, **Threshold:** -10dB, **Attack:** 0.5s, **Release:** 0.5s	2.6540e-07
Limiter	**Threshold:** -55dB, **Attack:** 0.001s, **Release:** 0.0005s,	2.3602e-07
Noise Gate	**Ratio:** 10, **Lower Threshold:** -30dB, **Upper Threshold:** -25dB, **Attack:** 0.05s, **Release:** 0.00001s **Hold Time:** 0.0001s **Pole Placement:** 0.3s	1.3804e-07

This concludes this chapter. Few important things will be discussed in the next chapter which will fill in the the knowledge gaps in the complete understanding of the project.

Chapter 5

Discussions

Before concluding the report, it is necessary to discuss a few essential points which will give some additional information to the reader and can act as future reference, while developing the application further.

5.1 Software Development Life Cycle

Usually when an application is built by an organization, it follows all the steps [15] as mentioned in this subsection. Each and every step is documented for clarity and for comparison between what was expected and what has been delivered and also for future reference. In this case, a slightly different approach, within the framework of student project, has been followed.

5.1.1 Planning

This is the first phase of the cycle. In this phase, project manager and development team lead decide on the resources in terms of project budget, scheduling and manpower. In this case, project plan was set only in terms of scheduling.

5.1.2 Requirements Gathering

Usually, a project goal and supporting set of requirements are set and well documented in a document called Business Requirement Document (BRD). This document acts as a reference for the user, the developing team, the testing team and also the enhancement or the maintenance team. In this case, through informal discussion, the project goal was set to build an application which performs similar to the applet was set as the project goal. Little variations can be allowed depending on the feasibility of implementation on a platform different from the applet.

5.1.3 Design

It is a task of the development team to decide on the complete architecture for the project and get it approved by the user before starting the development process. In this case, instead of setting up the whole project plan in the beginning, development happened bit by bit and the plan kept improvising throughout the whole process of implementation, depending on the situation. On the design bit, as mentioned in the journey of implementation, API was decided to be used for easier and quicker implementation.

5.1.4 Development

This is the sole responsibility of the development team, who follows the Business Requirement Document through the process of development. In this case, the journey of development has been explained in detail in third chapter. As mentioned in the journey of implementation, use of APIs or JavaScript or Java did not work out in the process of implementation. Hence, design decision was changed to using both JavaScript for client and Python in server, communicating by virtue of a framework named Flask.

5.1.5 Testing

This phase goes in hands and gloves with the development phase. This phase brings in the role of the testing team. Each and every sub phase in this phase follows after the corresponding development phase. Usually, a defect logging tool is used where testers can log defects with reference to the BRD. Developers also use this tool to access the logged defects and fix the same. In this case, defect logging tool was avoided as the tester and the developer were the same. Hence any defect that was spotted was at first fixed and then other processes moved forward. The sub phases of testing are:

Unit Testing

This phase includes testing each and every individual module. This project has several modules. Each and every module was tested solely, before integration happened. One example of a defect at this stage is when it was realized that plain JavaScript can covert a .wav audio file into an array buffer, but it cannot convert the array buffer back into an audio file. Next phase is Integration Testing.

Integration Testing

In this phase, successful integration among every module pair is tested, which happened also in this project and every defect of this phase had been taken care of. One small example of a defect in this step is when it was realized that Tone.js, which is used to compress an audio, cannot communicate with Wavesurfer.js, which is used to plot an audio, thereby making it impossible to plot a compressed audio using the combination of Tone and Wavesurfer APIs. Next comes the phase of System Testing.

System Testing

When the application has been built completely, the development team tests the overall system for any discontinuity before handing it out to the user. So has been done in this case. One small example of a defect in this stage was when it was realized that browser is caching data and hence cache control needs to be done. Finally comes the User Acceptance Testing phase.

User Acceptance Testing

This is done by the actual user, to whom the project will be finally handed over. This is basically a non technical testing, done to find out whether the system responds in a way as it was expected and whether it matches the requirements, set in the BRD. In this case, the application was handed over to Tutor and to a few of my colleagues in the University and was

suggested to try to break the same in each and every way possible and also to test for its look and feel. Feasible suggestions were incorporated. One small example includes removal of the default trace from the Static Curve, as suggested by Tutor. This phase concludes the testing process and the application is ready to be deployed.

5.1.6 Deployment

In this phase, the application is handed over to the user by deploying it on the web. Now it can be accessed by the target users from anywhere and at anytime. 3 different combinations of deployment have been prepared. The first one has both client and Python server on local system. This limits the access of application only to one system. The second one has only client on local system and Python sever on the web server. This also limits accessing the application from one system but saves the effort of setting up the Python server. The third one has both client and Python server deployed on the web servers. Any user can access the application anywhere and anytime through the required URL. Any of the three deployment measures can be taken depending on the requirement. Once this phase is over, the technicalities of project, as of now, comes to an end.

5.1.7 Maintenance and Enhancement

This phase follows post deployment. This includes maintaining the application to avoid server overloading and performing similar tasks and also looking for enhancements that can be undertaken for the improvement of the application. This phase, in this case, refers to the scopes of future work that have been mentioned in the next section.

5.2 Pros and Cons

This section lists out the pros and cons of the implementation with respect to all the available implementations.

5.2.1 Pros

This section describes the advantages of the application with respect to other three systems.

- There is no compatibility issue with browser version as it exists in the Java Applet. Any audio can be browsed for audio operations, whereas in the applet only two sample audios are used.
- If both client and Python server are deployed on the Web server, there is no need for installation of any additional software or plugin in the system. User can access the application by browsing the URL.
- All four modes of operation are available in the application. Results of operation also match with the one in MATLAB. Any kind of customization can be made in terms of

implementation. Neither all the four modes were available, nor it was giving same audio effect as that in MATLAB. Operations were limited. Only a single audio name can be hard coded for audio operations.

5.2.2 Cons

This section describes the disadvantages of the application with respect to other three systems.

- User has to manually enter the parameters in the input fields, as opposed to selecting the parameters by sliding two operational points in the Static Curve in the original applet. Also, the audio operations take some time for execution which was happening real time in the applet.
- Implementation is easier in MATLAB as it provides user friendly functions. Also, it is suitable for comparison of variables, plotting of waveforms without any additional library. There is no need for to and fro response for audio operations or plotting. All these come at a cost in this application, which is again a bit of latency.
- Waveform visualization along with audio operations were happening realtime using Tone API, which is not the case with the final version of the application.

5.3 User Guidelines

The application is already intuitive. But a few user guidelines will be helpful to the user.

- Audio file of size up to 1Mb or 500KB (for Noise Gate) can be uploaded, as bigger sized file entails more time in generating the output plot.
- An audio file should be browsed before trying to plot the original audio or user gets a message stating the same.
- User should plot the input audio first before trying to play it as the slider slides through the waveform when played which is not possible without the input audio plot generation.
- User should plot the output audio first before trying to play it because of the same logic mentioned above.
- User should follow the alert messages shown on screen, which pops up, if parameters are entered beyond range or are not suitable.

This section comes to an end and has given insight about different stages in software development, about the positives and the negatives of this implementation and also notified a few guidelines for the user. The next section will take the reader to the conclusion of the report.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

As mentioned in the beginning, the project demanded implementation of compressor on Web. After checking the feasibility of implementation, it was found to be not possible to implement within the available resources by using only client side. Hence, a full fledged project architecture was needed which included having client, server, a framework to enable communication between client and server and also finally a hosting environment so as to enable users to access the application from anywhere and at anytime. Additionally, the rest three modes of operation, namely Expander, Limiter and Noise Gate, have been implemented so that user can experience a complete Dynamic Range Control system in one application only.

After going through the previous chapters, reader should have formed an idea about the context and the project goal, developed a clear understanding about the current system and the need for the new one, design of the new system, the journey of implementation and also the challenges that came across. By looking at the results and comparing the same with the ones, generated in MATLAB, the reader can conclude about its successful implementation. Also, reader can have a basic understanding of how to advance through different stages of software development life cycle to implement a web application completely and user can make use of the user tips while using the application. It brings us to the end of the report. We can finally have an insight into possible future work that can done on this project to improve it even more.

6.2 Future Work

There is barely any resource on the internet which explains audio operations in a manner in which this application has been built. Even the already existing audio APIs are not strong enough to do advanced operations. That restricts usage of those APIs to implement customized audio operations. That is the reason, the architecture associated with the project has been thought of to enable freedom in terms of customization in implementation. Since this project marks the inception of building audio application in a way that has never been followed before, there are ample areas of future investigation.

There are some key areas which have been identified as future area of work for further improvement, following the foundation laid by this project. Those are

- GUI can be improved further to enhance the look and feel
- GUI can be made responsive so that the application adjusts its size automatically, when opened in cell phone or other differently sized smart devices.

- Audio operations can be made faster by operating on fewer samples, as explained in the DAFX book, so that it takes less time to generate the output. Hence, it will allow the application to operate on larger audio files.
- An API can be built which can plot more than one audio signal, which are obtained through the different steps of gain calculation, other than the output on the same canvas, which has markers for both axes.
- Instead of switching between different modes in the application, all four modes can be implemented in one window. This needs implementing sliding operational points in Static Curve which can take inputs from the user dynamically.
- Other applications like Quantizer and FFT can also be built in the similar structure used to build the Dynamic Range Control.
- A complete webpage for the Department of Signal Processing and Communications in HSU can be built which will consist of all the applications required. The webpage can be made password protected. It can have a dropdown or a list of the available applications. User can navigate to the one he/she chooses by clicking on the same.

There can be some other areas of work following a different and futuristic approach. Those are

- There is a latency in the application majorly due to the time taken for compilation in Python and minorly due to the time required for a to and fro response between the client and the server. The array of input audio can be split into multiple smaller frames in server, audio operations can be executed on each frame and the same can be sent back to client, and by the time the processed audio gets played in the client, the next frame can be processed and sent to client. This can cut down the latency and can be more close to being real time.
- Audio operations can be done real time, if a more audio specific web friendly API develops in future.

This lists out the possible future work in this area and subsequently brings us to the end of the report. I wish it was a pleasant experience going through it.

List of Abbreviations

A	
API	Application Programming Interface
C	
CSS	Cascading Style Sheets
D	
DRC	Dynamic Range Control
dB	Decibel
F	
FTP	File Transfer Protocol
FFT	Fast Fourier Transform
G	
GUI	Graphical User Interface
H	
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer protocol
J	
js	JavaScript
JSON	JavaScript Notation Object
K	
KB	Kilo byte
K	
MB	Mega byte
R	
RMS	Root Mean Square
S	
s	seconds
U	
UML	User Modelling Language
URL	Uniform Resource Locator

List of Software

Name	Version	URL	Comment
MATLAB	R2018a	https://de.mathworks.com	Used to write MATLAB code
Visual Studio	15.8	https://visualstudio.microsoft.com	Used to write frontend code
PyCharm	218.2.2	https://www.jetbrains.com/	Used to write backend code
Adobe Illustrator	CC 2018	https://creative.adobe.com	Image manipulation tool

Table 6.2: *Softwares used*

Bibliography

[1] U. Zölzer, “Digital audio signal processing.”

[2] “Pizicato documentation.” [Online]. Available: https://alemangui.github.io/pizzicato/

[3] “Tone documentation.” [Online]. Available: https://tonejs.github.io/docs/

[4] “Wavesurfer documentation.” [Online]. Available: https://wavesurfer-js.org/docs/

[5] “Loopslicer documentation.” [Online]. Available: http://kernow.me/loopslicer/

[6] “Audiocontext documentation.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/AudioContext

[7] “Html, css and javascript documentation.” [Online]. Available: https://www.w3schools.com/

[8] “Bootstrap documentation.” [Online]. Available: https://getbootstrap.com/

[9] “Plotly documentation.” [Online]. Available: https://plot.ly/python/

[10] “Python documentation.” [Online]. Available: https://www.python.org/

[11] “Scipy documentation.” [Online]. Available: https://www.scipy.org/

[12] “Numpy documentation.” [Online]. Available: http://www.numpy.org/

[13] “Flask documentation.” [Online]. Available: http://flask.pocoo.org/

[14] U. Zölzer, “Dafx: Digital audio effects.”

[15] “Software development life cycle documentation.” [Online]. Available: https://raygun.com/blog/software-development-life-cycle/