Summer Project `Documentation`

Harshit Jain

harshitjain1371999@gmail.com

Kunal Narayan

kunalnarayan@gmail.com

Abstract

A system for adaptively processing a telephonic speech signal performs modification in either the spectral domain or the time domain to bring the power in each frequency above the hearing threshold of the listener but below the upper limit of the listener's dynamic range.

1. Introduction

This project aims to develop a system for adaptive processing of speech signals for hearing impaired listeners, and has particular utility in adaptively processing telephonic speech signals to compensate the signal for hearing impaired listeners.

2. Background

As much as twenty percent of the population has some sort of hearing difficulty. It is typical for persons over 50 years of age to experience progressive loss in their aural perception in the high frequency part of the audio spectrum. A large percentage of those who have hearing impairment are aided in their understanding of speech in face-to-face communications by their familiarity with visual cues, and because the other persons speaking to them will adjust the loudness of their voices. However, visual cues are not available to the hearing impaired listener in a telephone conversation, and non-verbal interaction between communicants on the telephone is not possible. Also, there is from time-to-time the added problem of telephone noise and speech signal distortion which will add to the problems of the hearing impaired. Moreover, many of those with hearing impairments do not have

hearing aids. Even those hearing impaired persons who have hearing aids may have problems when attempting to use the hearing aid with a telephone due to feedback occurring because of the close proximity of the telephone receiver and hearing aid microphone, and difficulty in maintaining the optimum position of the telephone receiver. It is not uncommon for someone to have a hearing aid fitted to their best ear, but because of the problem of hearing aid–receiver interaction, the person uses the other ear for telephone communications.

It is known that the speech spectrum exists mainly in the band below 8,000 Hz, and that the most important region lies below 5000 Hz. Most of the power of the signal is contained in the band 100 to 1000 Hz, while the middle to higher frequencies contribute significantly to the intelligibility of the signal. The speech signal has a great deal of redundancy, in fact the band below 1500 Hz has about the same amount of intelligibility as the band above 1500 Hz. The telephone signal capitalizes on this redundancy and uses a band of 300 to 3200 Hz for voice signals.

While for the average person the telephone signal typically gives an intelligibility of better than 90 percent, for a significant minority of the population who have hearing impairments the telephone signal can present varying degrees of intelligibility.

At each frequency level within the telephonic bandwidth, the hearing characteristics of a particular listener may be measured by two parameters. First, is the threshold value (T) which indicates the power level that each frequency point must have for the listener to be able to hear that particular frequency. Second, is the limit (S) on the listener's dynamic range at each frequency point, which indicates when the listener will experience pain or discomfort when the power level at the frequency point is increased. The T and S values constitute a hearing profile which characterizes an individual listener. These profiles may commonly grouped or classified to match typical hearing impairment problems. Alternatively, the hearing profile of any particular listener may be unique to the aural impairment, disorder or disease suffered by that listener.

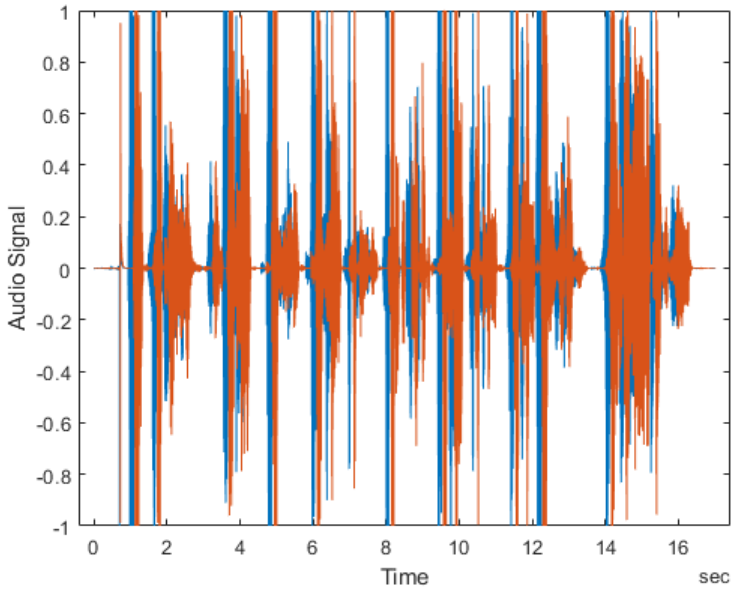
3. Algorithmic Steps

The preferred algorithmic steps for adaptive speech processing are generally described as follows. **First, the analog speech signal is converted into digital form, or if already in a digital form it is converted into a linear 16-bit integer representation. The digital signal is then filtered to remove noise. The filtered digital signal then undergoes**

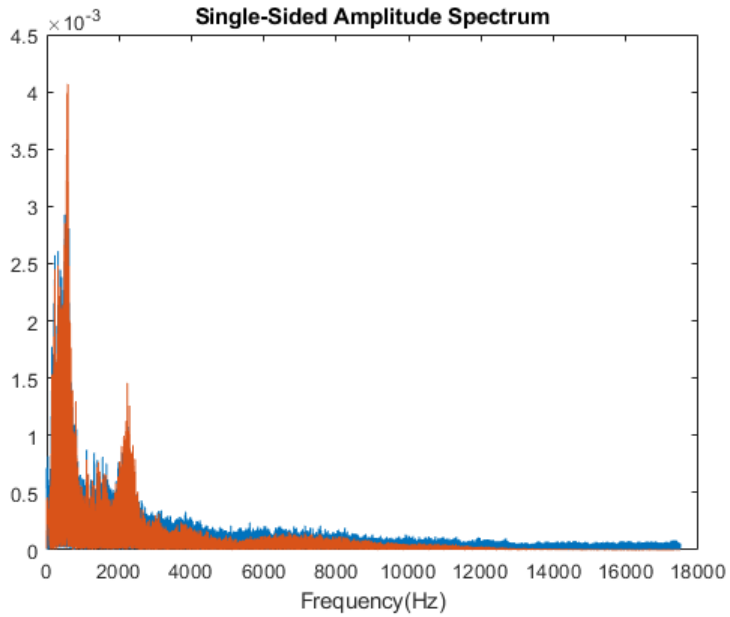
a Fourier transformation into the frequency domain, and each frequency component of the speech signal is represented by a point value (represented by real and imaginary coordinate values in the complex spectrum). A spectral modification is then performed by multiplying each point value based on the particular adjustment needed at that frequency level according to the requirements of the particular hearing impaired listener. The multiplication of the frequency point value is intended to modulate the power in that frequency to be within the range defined by the sensory threshold (T) at the low end and the dynamic limit (S) at the high end. The modulated frequency point values are then inversely transformed from the frequency domain to a digital representation of the speech signal. The re-digitalized signal is then further reconstructed by using an overlap and add method to prevent aliasing effects and to optimize its intelligibility to the hearing impaired listener. Finally, the digitized signal is re-converted to analog form for transmittal to the telephone receiver and improved perception by the hearing impaired listener.

In an alternative manner, the algorithmic steps may be implemented in a time domain processing method. In this method, signal compression at selected frequencies is implemented by adjusting the gain of frequency specific filters. Each filter has a different center frequency, and the center frequencies are octave-spaced within the telephone bandwidth.

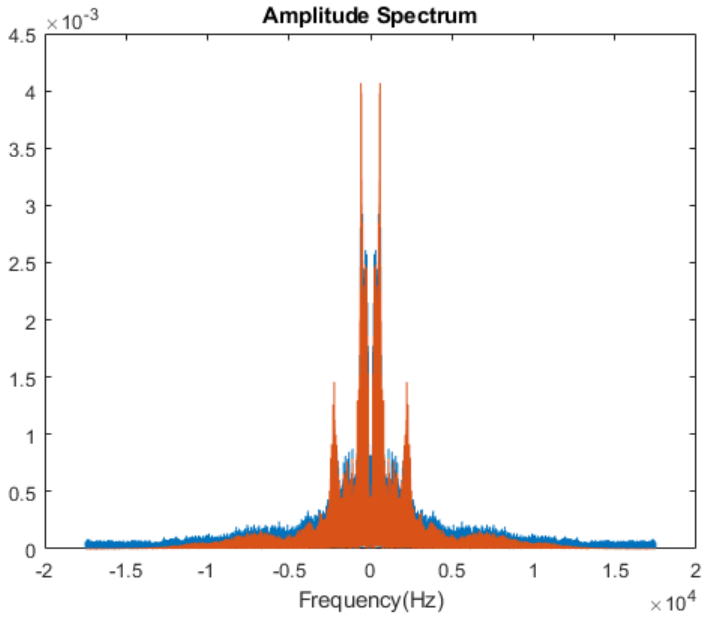
4. Audio Processing figures



Blue: Input Signal, Orange: Output Signal



Orange: Input Signal, Blue: Output Signal



Orange: Input Signal, Blue: Output Signal

5. Description

5.1. Libraries used

- RPi.GPIO: To take the audio input and output from Raspberry pi.
- threading: Since, the requirement was to perform the processing in real time, hence we used threads to do the tasks of taking the input and processing it parallely.
- pyaudio: To construct audio streams
- matplotlib: To plot the temporal and frequency plots of the signal

5.2. Working

There are two main things in the code - first being how actually the audio signal is being processed and the second is how to ensure that the input and the output are almost real time. Now, to process the audio such that the output comes almost at the same time of the input, we take the audio from the input stream in chunks of 1024 values. And while this is being processed another thread stores the next 1024 signal values in a buffer and prepares that buffer to be processed next. We do audio processing by converting the

signal first in to the frequency domain and then varying the respecting gains in different frequency ranges as specified by the user and then we reconstruct the audio to stream into the output. This mainly constitutes the working of the code.

5.3. Android app

In addition, we have also developed a basic android app which is used to modify the gain values as and when demanded by the user. For communication between the pi and the client's mobile, we have to ensure that both of these are connected to the same network and the server on the pi side is up and running. Now, whenever the app request for a change in gain values these get reflected in the audio processing code.

5.4. Python code

For the task of real time audio processing, libraries to be installed are `wave, pyaudio, numpy, scipy, matplotlib`. At present, the code takes the input stream in the form of small chunks, performs gain amplification on the chunks and adds them to the output stream. There is some code written to perform noise cancellation on the input stream, but as of now, the code doesn't use that part. Otherwise, the code as shown below has been well commented for the purpose of understanding the working.

```
import RPi.GPIO as GPIO
import os
import wave
import time
import datetime
from threading import Thread
import pyaudio
import numpy as np
from matplotlib import pyplot as plt
from scipy.signal import butter, sosfilt, lfilter, buttord

buffer_1 = []
buffer_2 = []
output = np.array([])
rate = 36000
flag = 1

# Default gain values
gain_range_0_500 = 1.2;
gain_range_500_1000 = 1.1;
```

```
gain_range_1000_1500 = 1.3;
gain_range_1500_2000 = 1.2;
gain_range_2000_2500 = 1;
gain_range_2500_3000 = 1.1;
gain_range_3000_3500 = 1.2;
gain_range_3500_4000 = 1;

# function to apply the set gains to respective frequency ranges
def gain_filter(y):
   global gain_range_0_500, gain_range_500_1000,
       gain_range_1000_1500, gain_range_1500_2000,
       gain_range_2000_2500, gain_range_2500_3000,
       gain_range_3000_3500, gain_range_3500_4000
   y = gain_correction(y, 1, 500, gain_range_0_500)
   y = gain_correction(y, 500, 1000, gain_range_500_1000)
   y = gain_correction(y, 1000, 1500, gain_range_1000_1500)
   y = gain_correction(y, 1500, 2000, gain_range_1500_2000)
   y = gain_correction(y, 2000, 2500, gain_range_2000_2500)
   y = gain_correction(y, 2500, 3000, gain_range_2500_3000)
   y = gain_correction(y, 3000, 3500, gain_range_3000_3500)
   y = gain_correction(y, 3500, 3999, gain_range_3500_4000)
   return y

# Amplification in frequency domain of signal y within frequency
    range (a, b)
def gain_correction(y, a, b, gain):
   N = 1024;
   Fs = 36000;
   out = y;
   out[-(N*b)/Fs + N/2 : -(N*a)/Fs + N/2] = gain * y[-(N*b)/Fs +
       N/2 : -(N*a)/Fs + N/2]
   out[(N*a)/Fs + N/2 : (N*b)/Fs + N/2] = gain * y[(N*a)/Fs + N/2 :
       (N*b)/Fs + N/2]
   return out

def butter_bandpass(lowcut, highcut, fs, order=3):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a


def butter_bandpass_filter(data, lowcut, highcut, fs, order=3):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
```

```
    y = lfilter(b, a, data)
    return y


def process_audio():
    global buffer_1, buffer_2, output, rate
    while(True):
        if(flag == 2 and buffer_1 != []):
            # print("processing buffer 1 started")
            # output = butter_bandpass_filter(buffer_1, 100, 4000,
                rate, order=3)
            output = gain_filter(buffer_1)
            buffer_1 = []
            # print("processing buffer 1 ended")
        elif(flag == 1 and buffer_2 != []):
            # print("processing buffer 2 started")
            # output = butter_bandpass_filter(buffer_2, 100, 4000,
                rate, order=3)
            output = gain_filter(buffer_2)
            buffer_2 = []
            # print("processing buffer 2 ended")

        output = output.astype(np.int16)

    return


def record_audio():
    global buffer_1, buffer_2, output, rate, flag
    CHUNK = 1024
    FORMAT = pyaudio.paInt16
    CHANNELS = 1

    p = pyaudio.PyAudio()

    p.get_default_input_device_info()
    # input audio stream
    stream = p.open(format=FORMAT,
                    channels=CHANNELS,
                    rate=rate,
                    input=True,
                    frames_per_buffer=CHUNK)

    # output stream
    output_stream = p.open(format=FORMAT,
```

```
                        channels=CHANNELS,
                        rate=35000,
                        output=True)

    # Chunk read from the input stream
    raw_data = stream.read(CHUNK)
    # filling buffer 1
    buffer_1 = np.fromstring(raw_data, dtype=np.int16)
    flag = 2
    t2.start()
    # writing processed input chunk to the output stream
    output_stream.write(output.tostring())

    # This loop alternatively fills buffer 1 and 2 for processing
        by a
    # parallel thread and writing the processed buffer to the
        output stream.
    while(True):
        raw_data = stream.read(CHUNK)
        if(flag == 1):
            print("filling buffer 1")
            buffer_1 = np.fromstring(raw_data, dtype=np.int16)
            flag = 2
        elif (flag == 2):
            print("filling buffer 2")
            buffer_2 = np.fromstring(raw_data, dtype=np.int16)
            flag = 1
        else:
            print("buffers blocked !!")
            continue
        output_stream.write(output.tostring())
        #frames.append(output.tostring())

    stream.stop_stream()
    output_stream.stop_stream()
    stream.close()
    output_stream.close()
    p.terminate()
    exit(0)

# t1: thread to take input chunks and store into buffers.
t1 = Thread(target=record_audio, args=())

# t2: thread to process the input chunks stored in buffers by
    preceding thread.
```

```
t2 = Thread(target=process_audio, args=())
t1.start()
```

5.5. Android studio code

The Gain Controller app is made using the software android studio, so to view the code, open the folder using android studio. It consists of the main java file and an xml file. The java file consists of the actual code and the xml file consists the layout i.e. UI that we see while running the app. To transfer the gain values from mobile phone to the pi, we are using simple socket, creating a data output stream and then flushing it to the pi.

```
Socket s = new Socket("192.168.43.99", 7000);
DataOutputStream dos = new DataOutputStream(s.getOutputStream());
while (true) {
    while (flag) {
        // connect to the device and change the gain values.
        // get progress value from the Seek bar
        int seekBarValue1 = simpleSeekBar1.getProgress();
        int seekBarValue2 = simpleSeekBar2.getProgress();
        int seekBarValue3 = simpleSeekBar3.getProgress();
        int seekBarValue4 = simpleSeekBar4.getProgress();
        int seekBarValue5 = simpleSeekBar5.getProgress();
        int seekBarValue6 = simpleSeekBar6.getProgress();
        int seekBarValue7 = simpleSeekBar7.getProgress();
        int seekBarValue8 = simpleSeekBar8.getProgress();

        JSONArray arr = new JSONArray();
        arr.put(modified_gain(seekBarValue1));
        arr.put(modified_gain(seekBarValue2));
        arr.put(modified_gain(seekBarValue3));
        arr.put(modified_gain(seekBarValue4));
        arr.put(modified_gain(seekBarValue7));
        arr.put(modified_gain(seekBarValue6));
        arr.put(modified_gain(seekBarValue5));
        arr.put(modified_gain(seekBarValue8));
        System.out.println(arr.toString());

        dos.writeUTF(arr.toString());
        dos.flush();
        flag = false;
    }
    sleep(500);
}
```

6. User Manual

1. Setup the raspberry pi. Then go into the code directory and type the following commands in two seperate terminals.

   ```
   python server.py
   python real_time_gain_controller.py
   ```

2. Ensure that the phone and the pi are connected to the same network.

3. Using android studio open the gain controller app, change the IP value in the java file to the ip of the pi and then install it into any mobile phone.

4. Open the app and mention the specific gain values for each frequency range and press the set button to set the gain values in the raspberry pi3. As soon as, you open the app it'll get connected to the python server running on the raspberry pi.

5. Connect the audio device from which the input has to be provided for gain correction and you will be able to listen the gain amplified output using the earphones in real time.

7. References

[1] Alvin M Terry and Thomas P Krauss. System for adaptive processing of telephone voice signals, February 7 1995. US Patent 5,388,185.

[2] Wikipedia contributors. Fourier transform — Wikipedia, the free encyclopedia, 2019. [Online; accessed 13-April-2019].