Voice Activated Food Preparation Assistant

Aditya Harakare

Mentored By: Sudeshna Trilok

Rebel Foods (Formerly Faasos)
Summer Internship (Electrical Engineering)

Project-2, May 2020

Contents

1 Abstract

1.1 Problem Statement

The main aim of the project was:

- To input an audio clip from the user
- Process the audio clip and guess the voice command
- Play an audio instruction in response to the recognised request

1.2 Brief Description of the approach

- The problem statement was broken down to a main task which involved recognising the audio command from the input voice.
- This was done using a **speech recognition API** provided by **Google**. The input voice from the user was first recorded in a file which was saved locally. This file was then sent to Google API where it used to recognize the text using a pre-trained **neural network**. The text was then sent back to us. After receiving the text, the code used to identify some keywords in the text and then the request was recognised. After which the appropriate audio output was given to the user.
- The only drawback to this software was that it **required the internet** to work.
- Hence, most of my project was concentrated in bringing this functionality offline and recognising the voice using **Audio Signal Processing** techniques.
- First the tuning of parameters and basic understanding was done by trial and error in **MATLAB**.
- Later, this code was translated to python and finally a **python package** capable of recognising the command was made.
- All the bits of the code were merged together and finally the first software was published which checked the internet connectivity and automatically decided which method of Speech Recognition to use.

2 Introduction

- In a typical kitchen where the chef is busy with preparation of food with his hands involved, a voice assistant is extremely important for providing aid to the chef in making the food.
- Hence there was a need to make a custom voice based food preparation assistant.
- Also, since we can't guarantee connectivity and external resources, bringing this feature offline was a major task.
- Typically all speech recognition software use Neural Networks, but to train such a network we require a lot of data points which were not available and hence I tried using a purely signal processing approach to the problem.
- At the end the results obtained were extremely satisfactory.

3 Research and Literature Review

- Speech recognition is the ability of a machine or program to identify words and phrases in spoken language and convert them to a machine-readable format.
- Many speech recognition software are developed using pre-trained neural networks. I used the SpeechRecognition package in python and the Google Speech Recognition API for the online version of the software.
- To know more about Speech Recognition, I went through a course on Speech Recognition by Microsoft available on edX. Most of which was not useful for our application as it described high level speech recognition systems using acoustic and language modelling on top of neural networks.
- Since the wave forms of various words are similar, speech recognition is generally not done using signal processing.
- But since we had to choose only among a finite number of possibilities of words at a time, finding the best match using signal processing algorithms seemed reasonable.

- Being in the Electronics Sub-System of team AUV-IITB, I had some prior experience on audio signal processing. There we use similar techniques for the underwater acoustic based localization system.

4 Materials and Methods

The main problem statement can be broken down into three basic tasks as shown in Figure 1.

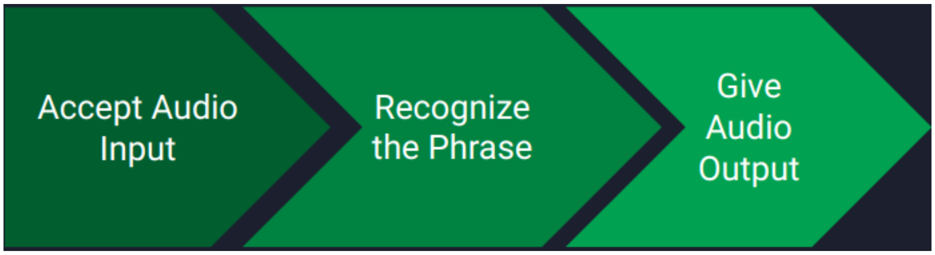


Figure 1: Division of Problem into sub-tasks

For recognising the phrase, firstly I started with the online mode and then proceeded for the offline one.

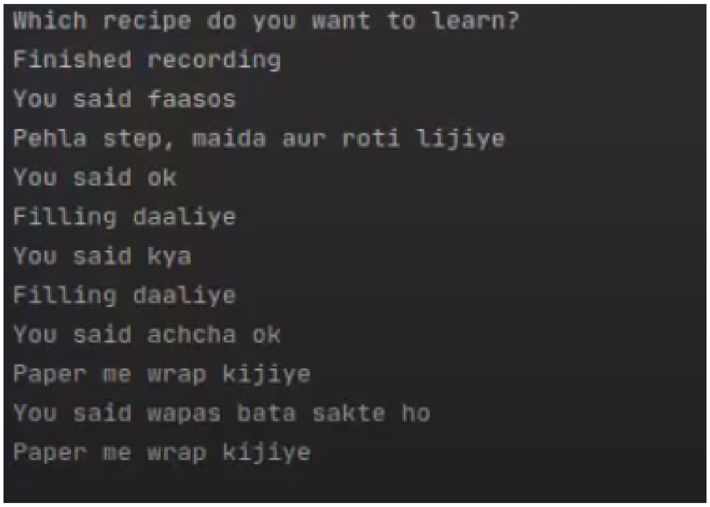
4.1 Online Version of the Software

- This version uses the Google Speech Recognition API to translate the audio input to text output.
- Following are the major steps involved in the code:
  - Record the audio Input
  - Save it locally
  - Send it to Google API
  - Google recognizes the speech using pre-trained Neural Networks
  - Apply conditions on the recognized voice and give the appropriate output.
- The advantages of this system are:

  - Very high accuracy as the neural network is trained using millions of training data.
  - Multi-language input option available.
  - User has the freedom to coin any sentence. We just pick out the keywords from it.

- The disadvantage is:

  - It requires the internet.

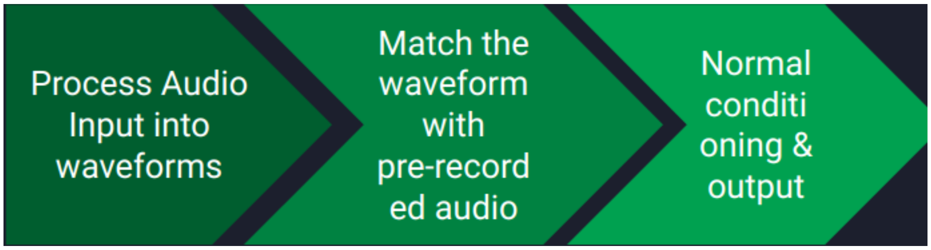


```
Which recipe do you want to learn?
Finished recording
You said faasos
Pehla step, maida aur roti lijiye
You said ok
Filling daaliye
You said kya
Filling daaliye
You said achcha ok
Paper me wrap kijiye
You said wapas bata sakte ho
Paper me wrap kijiye
```

Figure 2: Working Screenshot - Online Version

4.2 Offline Version of the Software - Signal Processing

The audio signal processing can be broken down to the following 3 major steps:



Process Audio Input into waveforms

Match the waveform with pre-record ed audio

Normal conditi oning & output

Figure 3: Audio Signal Processing - Outline

4.2.1 Recording Clean Audio - Audacity

- At the end, we need to compare between pre-recorded clean audio waveforms and pre-processed raw input voice from the user.
- For the generation of clean audio signals, I used the software called *'Audacity'*.
- Following Figure 4 shows the raw input waveform from the user.

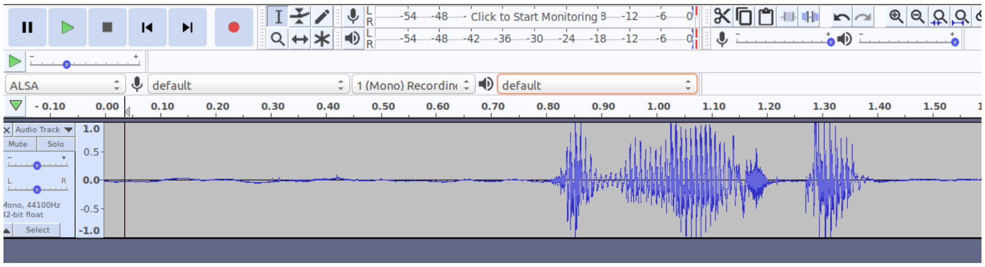


Figure 4: Audacity - Raw Input Waveform

- We can see that it contains a lot of noise and hence some processing is required to clean the waveform. Following process was used to process the audio and gave best results:
  - Noise Profiling

  - Noise Reduction
  - Normalization
  - Low Pass Filter
  - High Pass Filter
  - Normalization

- The details of each step are explained later. Following is the waveform after the processing is completed.

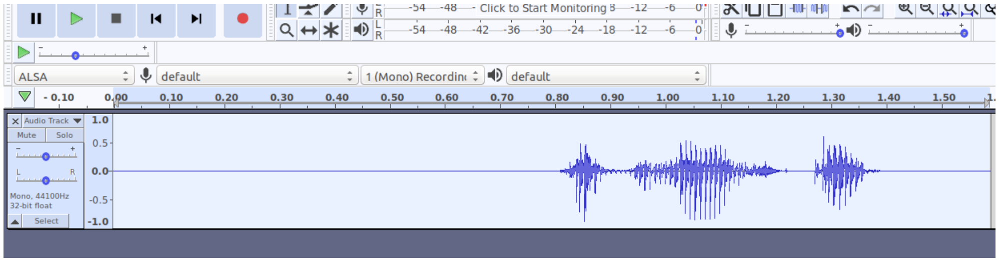


Figure 5: Audacity - Processed Input Waveform

- The waveform with the actual voice is cropped and saved for comparison later.

4.2.2 Simulating in MATLAB

- Matlab was used to carry out various preprocessing simulations on the input audio waveform and finally finding the correlation with prerecorded waveforms.
- The detailed MATLAB report can be found here: Link to the Matlab Report.
- Reading and Normalizing the audio data:

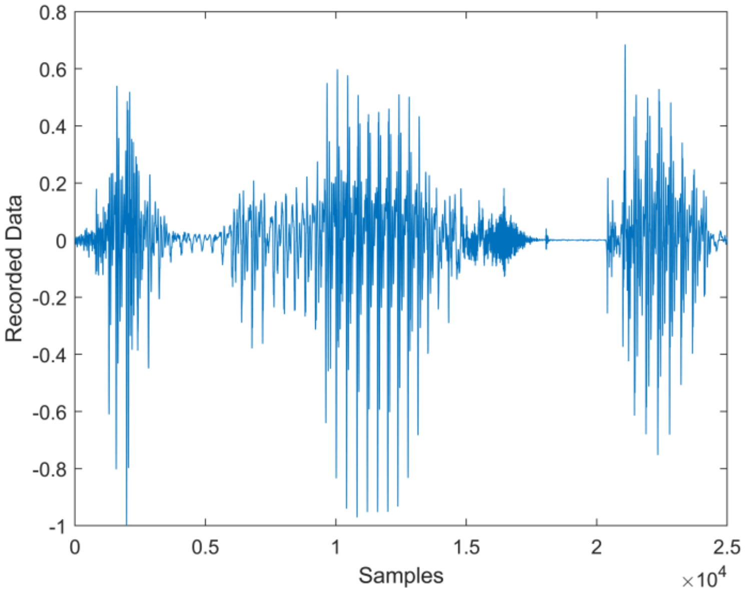


Figure 6: Pre-Recorded Audio Waveform

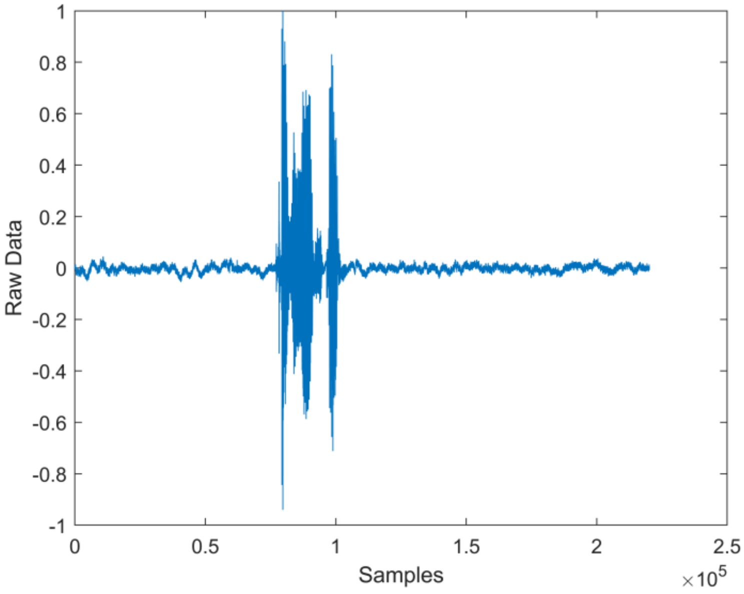


Figure 7: Raw Input Audio Waveform

- Pre-processing of raw data:
  - Clearly we can see that direct comparison of the waveforms is not possible and we need to pre-process the raw input first.
  - Hence, operations like High-Pass filter, Low-Pass filter and normalization were done and the following waveform was achieved:

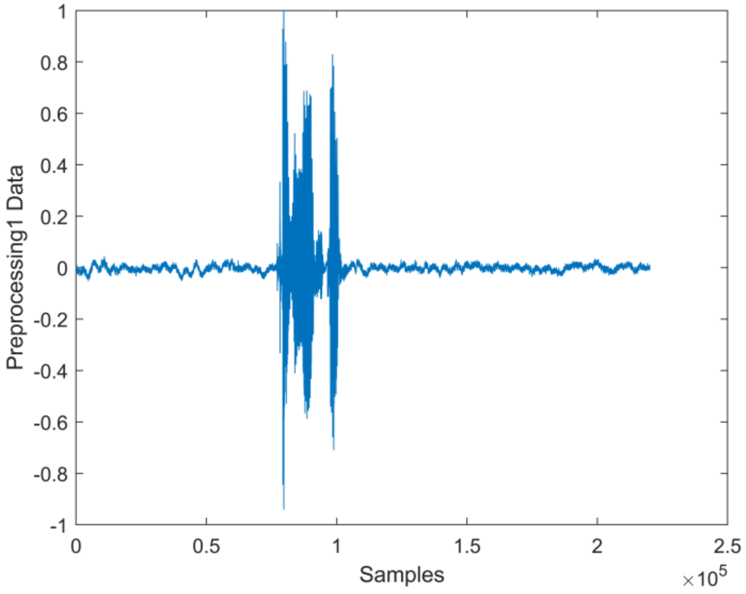


Figure 8: Pre-Processed Raw Audio Waveform

- We then resample the data to make sure the sampling frequency of the recorded and raw voice is same. This step is not required at this point and hence was removed later in the python code.
- Finding the bin where the voice command is present:
  - The next main challenge was to find the location of the start and end point of the voice in the total data sample. (Silence Removal).
  - For this, I used a method of two sliding windows.

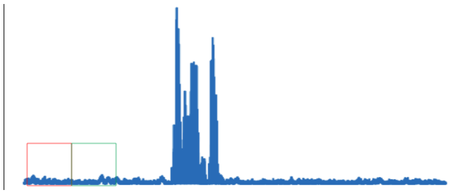


Figure 9: Create two Sliding Windows

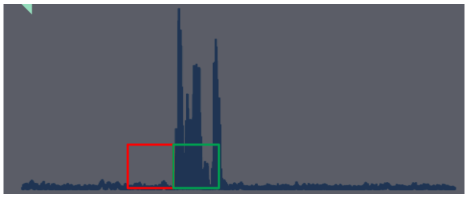


Figure 10: Point of Max BinRatio-1

- First we take the absolute value of the signal and then create two windows of suitable width and place them at the start.
- We define bin ratios as:

$$BinRatio_1 = \frac{\sum_{GreenBox} SignalData}{\sum_{RedBox} SignalData}$$

$$BinRatio_2 = \frac{\sum_{RedBox} SignalData}{\sum_{GreenBox} SignalData}$$

- Then we slide these windows across the complete signal and plot the BinRatio vs WindowStartIndex

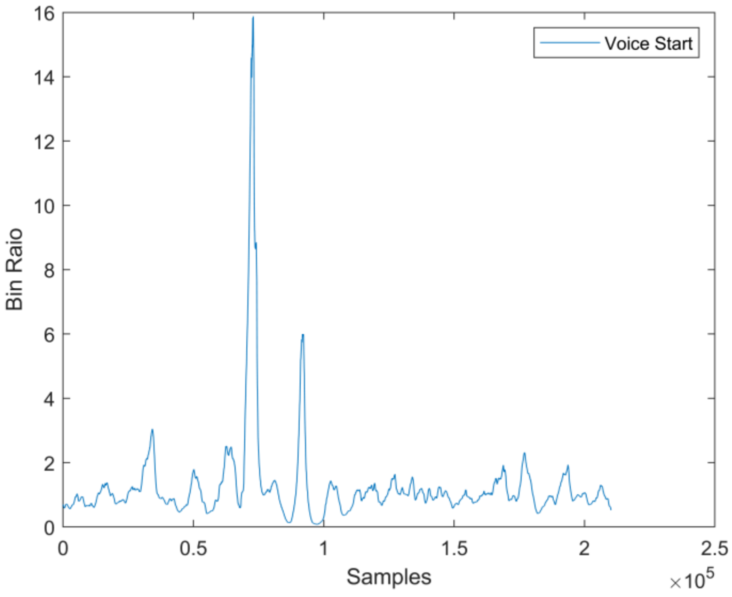


Figure 11: BinRatio1

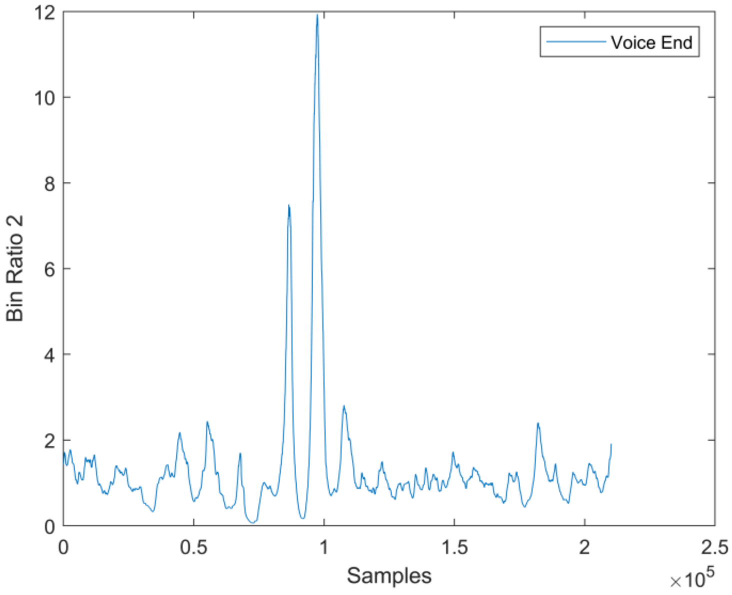


Figure 12: BinRatio2

- We then find the peaks of the above graph with a minimum prominance and calculate the voice start and voice end index.

  $$VoiceStartIndex = FirstPeakIndex(BinRatio1) + len(RedWindow)$$

  $$VoiceEndIndex = LastPeakIndex(BinRatio2) + len(RedWindow)$$

- Once we have the Voice start and Voice end index, we crop the input audio and only plot the part where the voice is present as seen in Figure 13.

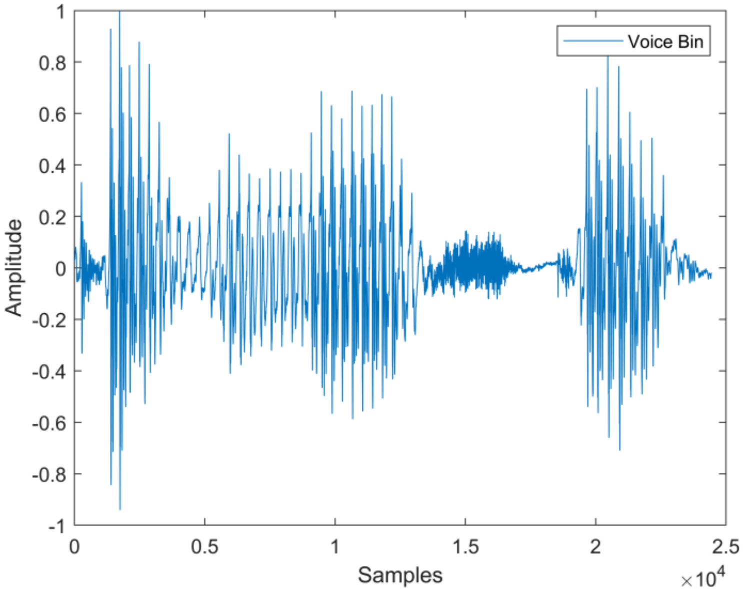


Figure 13: Cropped and Processed Raw Data

- Now we strech/squeeze the input data to account for slow/fast speaker. In reality we are making the number of sample points of input data and pre-recorded data same.

- Just before comparing, some more processing on the data was done like taking the absolute value of the signal and smothening the curve to some extent.

- Finding the Correlation:

- The correlation coefficient of two random variables is a measure of their linear dependence and captures how similar the two waveforms are.
- Correlation is mathematically defined as:

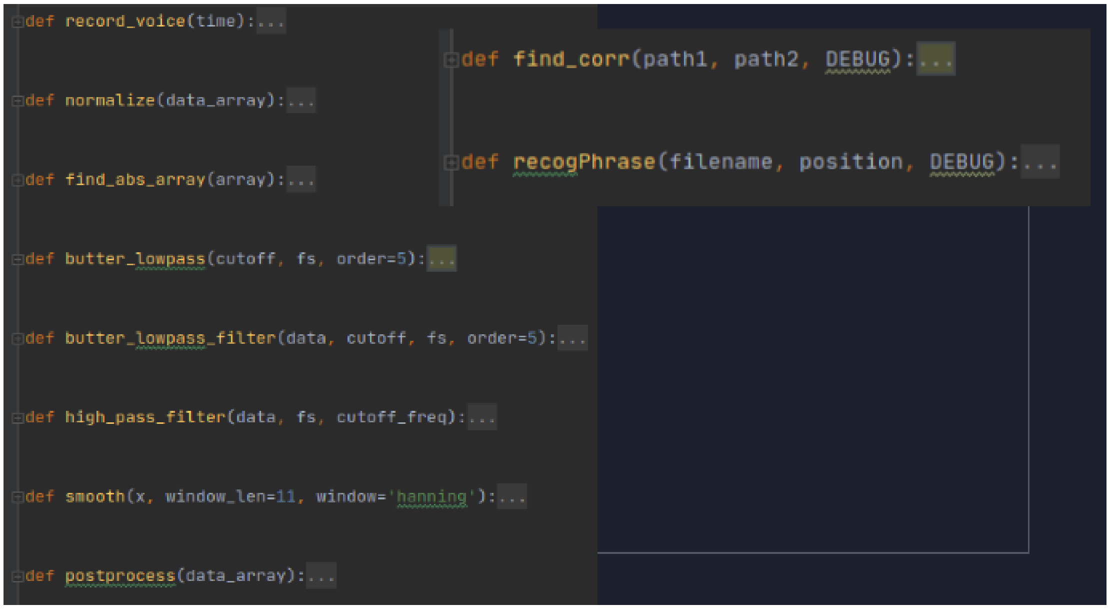
$$r_{xy} = \frac{\sum(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum(X_i - \overline{X})^2 \sum(Y_i - \overline{Y})^2}}$$

- The correlation of voices of same voice commands was found to be higher (close to 1) than that of different commands. However, the accuracy of this code was not high enough (60-70%) and hence it needed more ammendments and improvisations.

4.2.3 Making of the Python Package

The MATLAB code couldn't be used directly due to license issues with different machines and hence needed to be translated to Python/C++. I tried exporting the code as a python package/C++ library but as there is no one-to-one translation for various MATLAB functions, it raised many issues and hence manual translation was inevitable.

- Finally I created a python package with a number of improvements in the previous code.
- Following are the member functions of the package as shown in Figure 14. The description of each function along with the input parameters and the output is specified in the definition of that function.



```
def record_voice(time):...

def normalize(data_array):...

def find_abs_array(array):...

def butter_lowpass(cutoff, fs, order=5):...

def butter_lowpass_filter(data, cutoff, fs, order=5):...

def high_pass_filter(data, fs, cutoff_freq):...

def smooth(x, window_len=11, window='hanning'):...

def postprocess(data_array):...
```

```
def find_corr(path1, path2, DEBUG):...

def recogPhrase(filename, position, DEBUG):...
```

Figure 14: Member functions of the Python Package

- Here *find_corr* is the main function which returns the correlation between two audio files.
- We will only discuss some major additions to the MATLAB code here, rest of the code is similar.
- Check whether the voice is present:
  - We simply threshold on the maximum value of the input signal and check whether the voice is present in the audio file.
  - If not we immediately return -1000
- Hamming Filter:

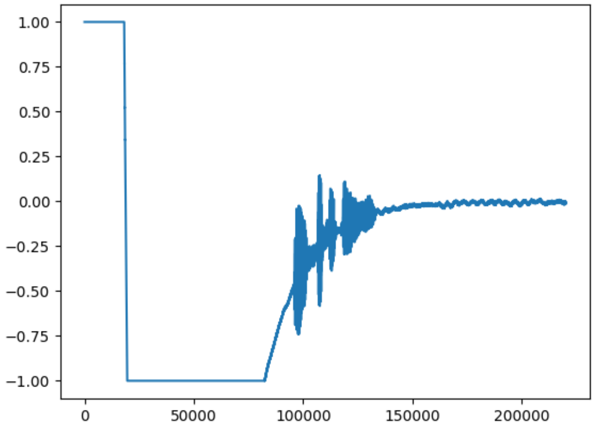


Figure 15: Uninitialized Microphone Input

- The waveform in Figure 15 is when the microphone is uninitialized and the audio input is taken. Basically it has a DC bias and needs to be 'detrended' before we move on to the further processing.
- A high-pass filter does the necessary work but might lead to unwanted spikes at the start as seen in the Blue waveform in Figure 16

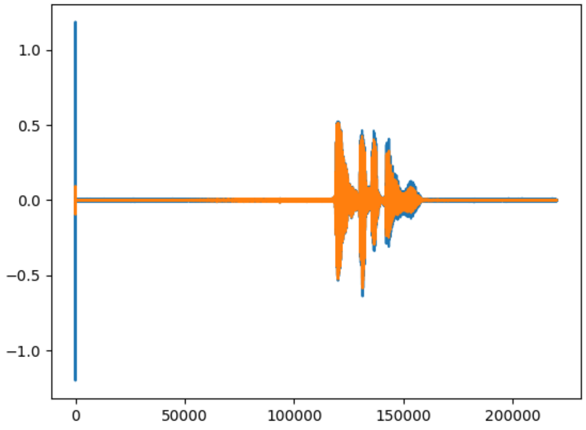


Figure 16: Hamming Filter

- Hence we multiply this signal with a Hamming Window of appropriate size as shown in Figure 17.

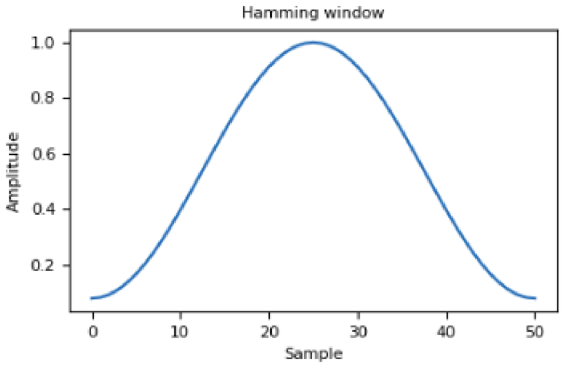


Figure 17: Hamming Window

  - This accounts for the end effects and we get the resultant waveform as seen in the Orange signal of Figure 16

- Noise Profiling and Noise Reduction:
  - We input some part of the start of the audio tract to the noise profiling function which generates a model for the noise in our audio sample.
  - We then reduce this noise from the rest of the sample.

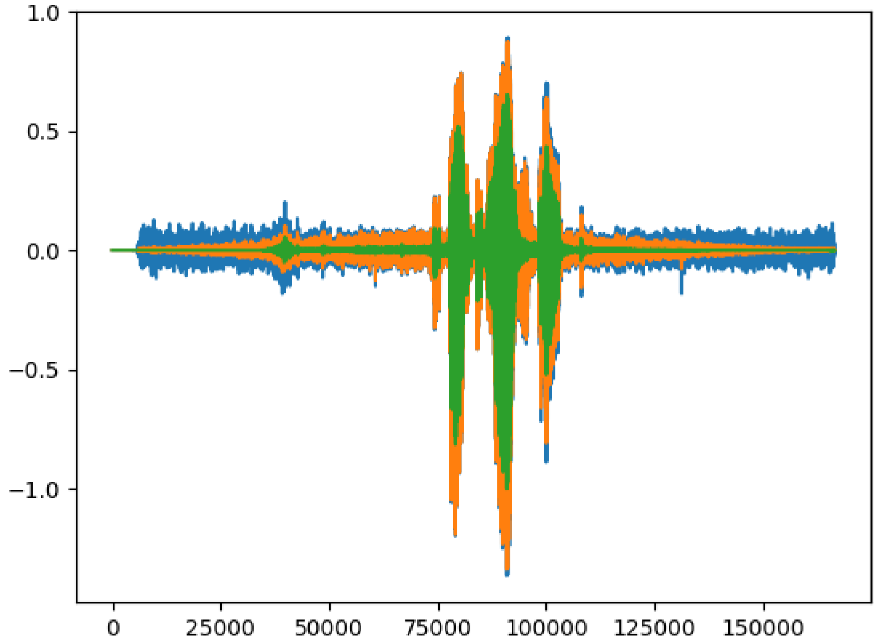


Figure 18: Noise before Voice

- In Figure 18, the blue waveform indicates the raw audio signal, orange is after applying the hamming filter and green is after applying noise reduction.

- Finding the bin ratio:
  - If the signal after noise reduction remains very close to zero for some time and has a noise spike later, this may lead to a very high bin ratio value and an unwanted peak in the plot.
  - To account for this, I slightly changed the bin ratio formula to:

$$BinRatio_1 = \frac{\sum_{GreenBox} SignalData + 0.5}{\sum_{RedBox} SignalData + 0.5}$$

$$BinRatio_2 = \frac{\sum_{RedBox} SignalData + 0.5}{\sum_{GreenBox} SignalData + 0.5}$$

  - After this the plot of the bin ratio was scaled such that the highest peaks of the BinRaio1 and BinRatio2 are of the same height

- Correcting for unwanted noise pulses:

  - As seen in Figure 18, there was some noise at sample no 40000 and hence the voice start which was detected by our algorithm was wrong.
  - This was corrected by thresholding distance between the voice start and the maximum peak index.

```
while voice_peak - voice_start > 40000:
    if n_peaks1 == index1 + 1:
        break
    voice_start = peaks1[0][index1 + 1] + bin1
    index1 = index1 + 1
    if DEBUG:
        print("Bad Noise Detected before start")
```

  - As seen in Figure 19, there is some noise detected just before the voice and hence the voice start turns out to be wrong.

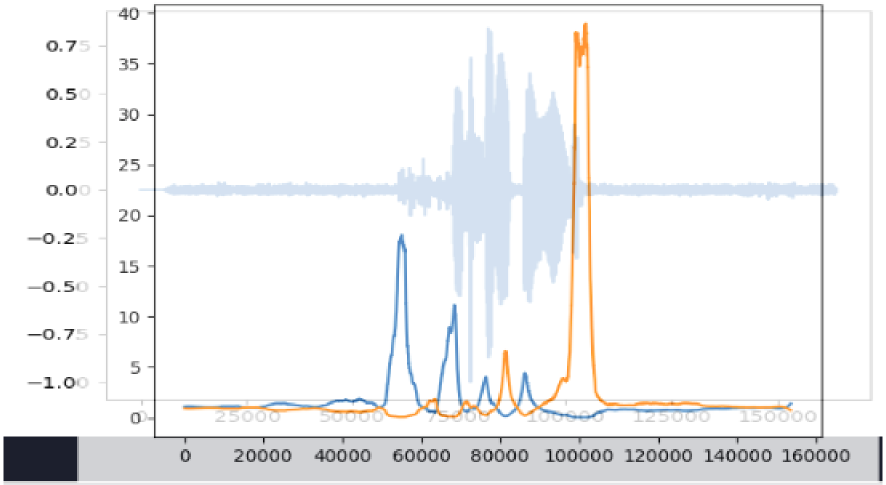


Figure 19: Noise just before Voice

  - This anomaly was corrected by detecting that the signal is below a threshold for quite a long time after voice start.

```
max_val1 = max(y_reduced_noise[voice_start:voice_start + 10000])
while max_val1 < 0.2:
    if index1 + 1 == n_peaks1:
        break
    if DEBUG:
        print("Noise Detected just before voice")
    voice_start = peaks1[0][index1 + 1] + bin1
    index1 = index1 + 1
    max_val1 = max(y_reduced_noise[voice_start:voice_start + 10000])
```

- After finding the voice start and voice end correctly, the input signal is cropped and then some further processing like taking the absolute and normalizing is done. After which the correlation is taken with the recorded voice and 100*(correlationCoeff) is returned.

- The recogPhrase function is a custom made function for our use case where the input file and the type of voice command (Title/Instruction) is specified. The function returns the most probable phrase or -1 if no voice is detected in the file.

4.2.4 Python Programs Developed on the Package

Various programs can be developed which use the functions of this package. Few of the programs useful for us are:

- Record_Audio.py
  We can now record clean voice samples using this program and our microphone so there is no need for the Audacity software.

- Record_Raw.py
  This program records your voice using the default microphone and saves it in a file. These raw recordings can be used to test/train any software for voice recognition.

- Recog_Microphone.py
  This programs records the voice command using the default microphone and recognizes it according to the parameters specified.

- Test_to_Speech.py
  This program converts the text into voice instructions and saves it in a file at the location specified.

4.2.5 Voice_Assist_v1.py

- This is the final program developed for our application.

- First it checks for internet connectivity and then switches between online and offline mode automatically.

- Currently the offline mode is capable of distinguishing between the following titles of the recipes:
  - Faasos
  - Behrouz Biryani
  - Oven Story Pizza

- It can distinguish between the following commands at each step:
  - Agla Step
  - Fir Se Bataiye

- The instructions for preparation of any recipe can be fed into the respective functions. A demo for Faasos is provided in the code.

- Demonstration Video for the software can be found at the the following links:
  - Online Mode
  - Offline Mode

5 Results and Inferences

5.1 Speech Recognition API

- This online method gave the most promising results as expected.
- It can recognize words and phrases of vaious languages and we can extract and condition on the keywords of the phrase.

5.2 Audacity

- This software generated extremely clean audio files and removed noise to the maximum extent possible. The cut-off frequencies of the filters were tuned for voice range.
- However after the development of a python script for recording clean audio, the need of this software has vanished.

5.3 MATLAB

- The pre-processing was first tried and tested on Matlab.
- However fine tuning and accounting for edge cases was not done on Matlab and hence outcomes were sometimes erroneous.

5.4 Audio Signal Processing in Python

- After tuning all the parameters, this gave excellent results with accuracy $> 95\%$.
- Around 60 Data samples were collected of 5-6 different voices and the model was trained and tested on this data.
- However, extensive testing is not done yet.

6 Summary

Speech recognition is an interdisciplinary subfield of computer science and computational linguistics that develops methodologies and technologies that enable the recognition and translation of spoken language into text by computers. This is usually done by using Machine Learning approaches and complex probabilistic models.
This project needed very limited speech recognition and hence we decided to use Signal Processing to solve the problem. Firstly, I developed the online version using Google Speech Recognition API and the results were extremely delightful. After a lot of trial and error and tuning various parameters a promising offline version was also developed which completely relied on Audio Signal Processing.
However, as waveforms of many words look similar, scaling this approach to a very large scale might be difficult and an easier way out would be trying to train our own neural network.

References

[1] *Underwater Acoustic Localization and Positioning* , AUV (Autonomous Underwater Vehicle) - IIT Bombay

[2] *Speech and Audio Signal Processing using Matlab* , JCBRO Labs

[3] *Audio Signal Processing in Matlab*, Circuits DIY

[4] *Speech Recognition*, Microsoft (edX Course)