Biomedical Signal Processing Speech Detection to Text

Anthony Huynh and Christopher Morales
4820, California State University Los Angeles, Los Angeles, CA 90032
(Dated: May 16, 2022)

The research relevance is based on the notion of the audio interpretation/understanding to grasp an internal representation of vibration that propagates as an acoustic wave. In response to consumers' demands and technology from the time of using Ear Trumpet to Cochlear Implants. These instruments are needed for multimodal that have been based on translating acoustic waves to speech. However, implementing a code to gather vibrations into speech recognition*, the process of the code will use Spectrogram to visually inspect the spectrum of frequencies of the given speech/phoneme.

I. INTRODUCTION

According to a Harvard American History study, during the 1700s, Martha Vineyard had a wide range of population of deaf people. Some areas were as high as 1/4 which created a human interaction burrier and created a difference in the community. Harvard study also mentioned that in the 1760s, the French Sign Language was created which consisted of the alphabet only. If the area was not educated well enough then they will be able to purchase an Ear Trumpet. A regular trumpet works by the individual creating vibration by buzzing on the lips to the mouth piece to produce sound, however the Ear Trumpet idea was to gather the vibration that is connected to one ear. The Ear Trumpet would amplify the sound to a degree but the speech would not be audible to the individual. In today's modern age, according to Charles Reilly and Sen Qi, from all ages about 600,000 people are deaf (roughly about 0.22% of the population of the US) [3] and about 15% of the US population have trouble hearing at the age of 18 or older [4]. The demographic of the population of losing or having trouble hearing is quite significantly large that could affect a spectrum of individuals that have not learned about the American Sign Language or could not afford a proper Cochlear Implants. According to the Mayo Clinic [2], the Cochlear Implants uses a sound processor that is connected to an array of electrodes that is able to pick up the sound. Also, the cost of a Cochlear Implant could range from $60,000 to $100,000 or more according to Duke Health [5]. Compare to an hearing aid its 1/10 of the cost of an Cochlear Implant however, people who have cochlear implant report to hear speech without visual cues such as reading lips and listen in noisy environment according to Mayo Clinic [2]. Our team would like to create an open source algorithm that is open to the public that does not require tensorflow or machine learning level that may give other willing participants to understand the process of Biomedical Signal Processing Speech Detection to Text algorithm. The project will take an audio file that will be able to convert the Short Time Fourier Transform of the audio signal.

II. TECHNICAL DESCRIPTION

By receiving the acoustic waves, we are able to imply a spectrogram onto the acoustic wave to get a frequency visual representation of the sound. Before we go in depth, to understand the acoustic wave, the American Language uses the enthesis of the pronunciation of a sound (acoustic wave) to give a distinction of a word or also known as phonemes. To gather data from an audio file we need to use an matlab function called, audioread, as demonstrate below:

[y, fs] = audioread(*filename*)

Where, y, will be represented as the audio data, while fs will be represented as the frequency sample rate of the audio data. If we plot one of the phonemes, we get the following,

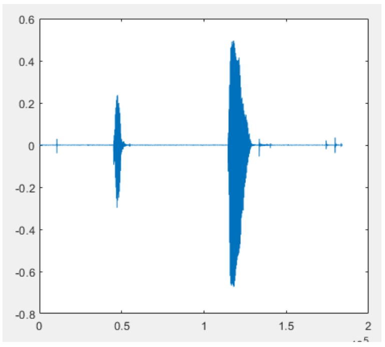


FIG. 1: Speech signal of the audio /b/ bug

With this information we will be able to use another matlab function called the spectrogram. The spectrogram will return the Short Time Fourier Transform of the audio data (positive side spectrum), a vector of frequencies, and a vector of time. In order to use the spectrogram we will need to create a window, a percentage overlap of the window effect, number points which will be called nfft, and the frequency sample rate. Note, the following constants are:

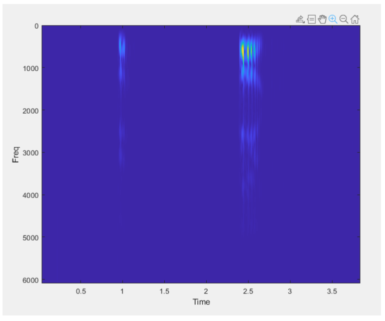


FIG. 2: Spectrogram of the speech signal /b/ bug

III. METHOD I

Our first method that we decided to try was to get a common single or multiple peak frequencies for each phoneme and have a threshold magnitude value that the audio file would have to pass for the phoneme to be detected. To get these peak frequencies we had to visually define each frequency and save it to create a phoneme bank. While this method somewhat worked with a few phonemes, Dr.Won had warned us that it may not be as well detected with all the phonemes due to overlapping frequencies. We saw this issue when we added all the phonemes. Our hypothesis why this would lead to an inconclusive data would be saying this one frequency will be this certain frequency for the phoneme rather than a band of frequencies. Meaning, if the phoneme relied on one frequency or the average of two voices, would be dependent on those two voices. If the two people or the person, had a low pitch voice and if we used the data speech detection then the frequency would be too low to be detected for the correct required phoneme. We determined this issue by recording the alphabet and we noticed that there was a pattern. An example would be the letter A, we would see a curvature and multiple frequencies being active but the pitch between our voices shifted the frequencies. We noted that our voices were in the low spectrum and if someone had a high pitch voice then the data would not be viable for the high pitch person.

IV. METHOD II

Our second method that we tried was to instead of getting only the peak frequency, we would get a bandwidth of the peak frequency and get an average magnitude. For the average magnitude we would sum up all the magnitudes within the frequency range. Then we would divide by how many frequencies that we summed up to get the average density. We would then normalize this data by dividing by the highest density value that is seen using the function Max. Note, for method 2 specifically, the frequency and time domain of the phoneme will be visually recorded rather than using an algorithm to determine the starting and ending points for the given domains. To gather up the phoneme frequency ranges we created a separate code called Project_textconfig.m to help us automate typing each variable out.

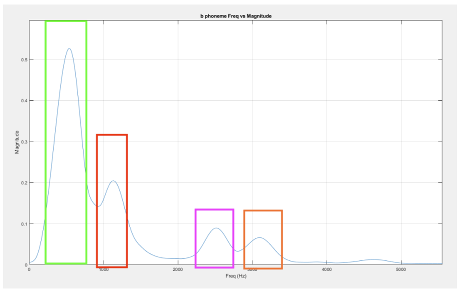
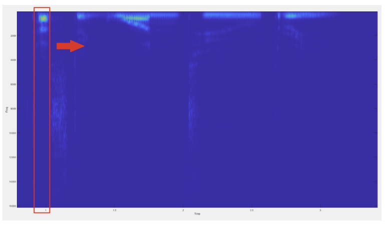
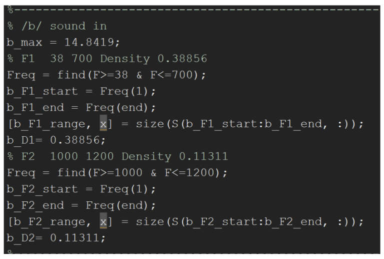


FIG. 3: Grouping different peak frequency ranges to get avg density.

If we look at the figure above, we can see on the green rectangle window getting the highest density in that window range. Then the window will find the next highest density in the specific frequency domain and so on. Then if we look at the figure below, we can see a visual representation of our window shifting across our audio signal which is compared to each of our phonemes.



(a) STFT of the windowing effect shifting across

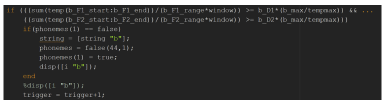


```
%-----------------------------------------------
% /b/ sound in
b_max = 14.8419;
% F1  38 700 Density 0.38856
Freq = find(F>=38 & F<=700);
b_F1_start = Freq(1);
b_F1_end = Freq(end);
[b_F1_range, x] = size(S(b_F1_start:b_F1_end, :));
b_D1= 0.38856;
% F2  1000 1200 Density 0.11311
Freq = find(F>=1000 & F<=1200);
b_F2_start = Freq(1);
b_F2_end = Freq(end);
[b_F2_range, x] = size(S(b_F2_start:b_F2_end, :));
b_D2= 0.11311;
%-----------------------------------------------
```

(b) Store Threshold Values for the Windowing Effect

FIG. 4: Threshold for the windowing effect

Here above we have the code section that stores the frequency ranges that we observed when using spectrogram on the audio file. Then we would have a for loop that would take a window that would have a small section of the whole audio file. And compare it to the predetermined values with an if statement seen below. We then have an array called phonemes that would trigger a true for our phoneme to prevent the phoneme being detected again until another phoneme is detected to reset it to false.



```
if (((sum(temp(b_F1_start:b_F1_end))/(b_F1_range*window)) >= b_D1*(b_max/tempmax)) && ...
    ((sum(temp(b_F2_start:b_F2_end))/(b_F2_range*window)) >= b_D2*(b_max/tempmax)))
    if(phonemes(1) == false)
        string = [string "b"];
        phonemes = false(44,1);
        phonemes(1) = true;
        disp([i "b"]);
    end
    %disp([i "b"]);
    trigger = trigger+1;
```

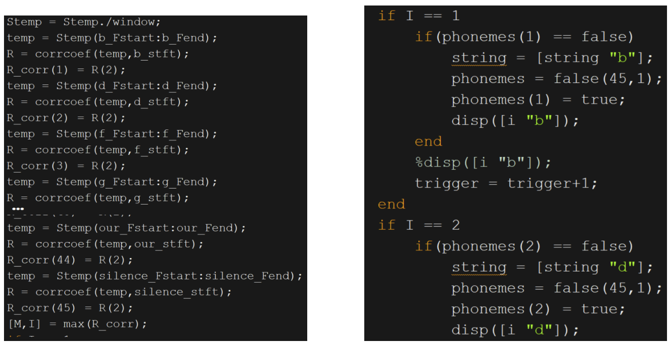
FIG. 5: Phoneme Detection above Threshold

The figure above shows how method II worked for a few phonemes about 15, but once we tried all 44, we had an issue with false triggers of the phonemes in the code. The false trigger was caused by the density threshold process. When looking at a spectrogram phoneme plot we see there will be activity throughout the frequency range. This false trigger was due to some phonemes having similar average densities. We tried to eliminate this by adding a normalization process of the phoneme and the window. However we noticed that normalizing our phoneme and normalizing our test signal, they had different max peaks. This caused an issue of some thresholds being too high even with normalization. We then tried to only normalize within our window by using the current max peak density in the window. But this brought up issues where we had silence and normalized that. It would have also falsely triggered a phoneme. To counter the density average issue, we implemented a Butterworth filter that was setup as an high pass filter however the false trigger persisted and another method was created to eliminate the common false trigger frequencies that were seen. This is when we thought of trying a different method where we would store the whole frequency of the phoneme and cross correlate it with the window's frequency.

V. METHOD III

Our third and final method that we tried was to store each frequency vs magnitude vector of each phoneme into a .mat file and then retrieve that data to cross correlate it with the current time windows frequency vs magnitude vector. We also had created a silent phoneme to test correlation with the window when there are no phonemes present. We then had an array to store all the correlation percents for each phoneme to the window. And we would then see which one correlated the most. We then gathered all the data using a separate code called Project_textconfigV2.m which allowed us to save the workspace of it as our phoneme_bank.mat

The correlation was done with the method below. The code below we can see we have done a correlation of the current window with each of the phonemes. We then store this correlation to use the function max to find the highest correlation phoneme that occurred at this time.



```
Stemp = Stemp./window;
temp = Stemp(b_Fstart:b_Fend);
R = corrcoef(temp,b_stft);
R_corr(1) = R(2);
temp = Stemp(d_Fstart:d_Fend);
R = corrcoef(temp,d_stft);
R_corr(2) = R(2);
temp = Stemp(f_Fstart:f_Fend);
R = corrcoef(temp,f_stft);
R_corr(3) = R(2);
temp = Stemp(g_Fstart:g_Fend);
R = corrcoef(temp,g_stft);
...
temp = Stemp(our_Fstart:our_Fend);
R = corrcoef(temp,our_stft);
R_corr(44) = R(2);
temp = Stemp(silence_Fstart:silence_Fend);
R = corrcoef(temp,silence_stft);
R_corr(45) = R(2);
[M,I] = max(R_corr);
```

(a) Correlation of all phonemes

```
if I == 1
    if(phonemes(1) == false)
        string = [string "b"];
        phonemes = false(45,1);
        phonemes(1) = true;
        disp([i "b"]);
    end
    %disp([i "b"]);
    trigger = trigger+1;
end
if I == 2
    if(phonemes(2) == false)
        string = [string "d"];
        phonemes = false(45,1);
        phonemes(2) = true;
        disp([i "d"]);
```

(b) Outputs the correlated phoneme if index is trigger

FIG. 6: 2 Figures side by side

We then noticed that we still had an issue with a lot of the phonemes having similar correlation. This is due to the majority of the frequency spectrum being a flat line. This can be seen below. In the red rectangle you can see the main activity of the phoneme 'b' but then we have a majority of the spectrum being a flat line. This can cause issues because corrcoef sees this as a high correlation with alot of frequency spectrums that have a flat line.

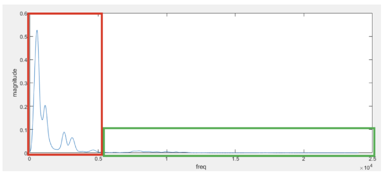


FIG. 7: Showing the big deadspace in green box

To resolve this issue, we also limited the frequency range to best fit the phoneme where we would cut off the frequency if we noticed little to no magnitude. This meant that we would need to store the start and end frequencies for each phoneme which we did in Project_textconfigV3.m we also had to change the correlation to have the same length now so we would use the start and end freq to snip the window size to match our stored phoneme. We then save the workspace into a mat file called phoneme_bankV4.mat

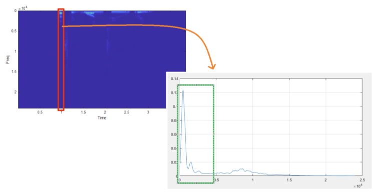


FIG. 8: Shows us taking a window time segment and getting a Frequency vs avg magnitude

Looking at figure 6, we have a range for time and frequency then plot the signal to determine which peaks are active from the phoneme.

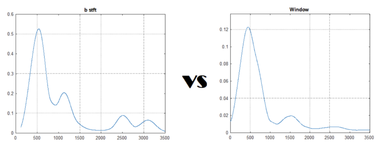


FIG. 9: Shows us comparing the green segmented window in Fig 6 vs our stored phoneme

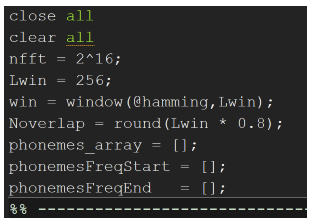
While narrowing down the window's frequency seemed to help by reducing the unnecessary spikes that could be meant for another phoneme. We still seemed to have issues with noise or ending sounds being detected as a phoneme.

VI. SPECIAL CASES

For previous and current methods, we noticed that a few phonemes are unique and will be considered special cases. Each method had their own problem with the special cases. The list of phonemes that are considered special cases are s, t, th, v, oo as in look, oo as in boom, and noise. What makes these phonemes special cases is the characteristic it is used in a word rather than saying the phoneme itself. An example would be the /t/ phoneme, when saying the /t/ it is quite distinctive but when saying the word “fat”. We noticed that the /t/ is more silent than the /t/ itself. Which presented an issue of t not being detected but /f/ and /a/ were detected for our current method, 3. Having the idea in mind, for method 1 the issue was the magnitude being lower or using the spectrogram to manually see the visual frequencies but in doing so led the special cases where barely visual when inspecting using the spectrogram or the magnitude was low enough to be considered noise. As for method 2, we noticed that when we got the densities for all phonemes, the special cases were either out of range, below the threshold or exceeding the threshold. Making the subtle special cases not being detected with. As for method 3,instead of a threshold, we decided to use corrcoef to see the correlation between the current window and all the different phonemes. This would allow us to just select the highest percentage correlation as the correct phonemes. To overcome the special cases we needed to create a new threshold that would consider the frequency range, time range, limit the activity of the stft range, and create the threshold using these new sets of boundaries. The time domain would limit the length of the phoneme since some are very short, the frequency range would define the subtle activities such as /t/ or the word “fat”, and the stft would let the code know the activity of the audio file or phoneme. With the current method, most if not all phonemes are able to be detected but noise is still prevalent. Furthermore, I have not mentioned noise for either methods due to method 1 and 2 where not stable enough to determine if the system had noise or not. With method 3, we were able to distinguish if there was noise on the system. Furthermore, we created another phoneme that will be called noise. When applying the spectrogram, noise was constant throughout the time and frequency domain. We would get the average of the noise and on top we used a butterworth high pass filter to eliminate some of the lower end of the spectrum noise. Keep in mind, noise is still prevalent for method 3 but we are able to detect relevant data with the added procedures.

VII. PERFORMANCE OF THE PROGRAM

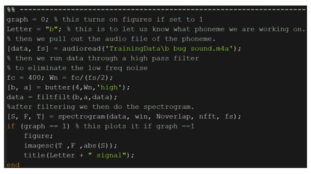
To start our code, we had to first gather all the phoneme data points. To do this, we created a Project_textconfigV3.m where it would run and after it finished. We could save the workspace as a .mat file to call later.



```
close all
clear all
nfft = 2^16;
Lwin = 256;
win = window(@hamming,Lwin);
Noverlap = round(Lwin * 0.8);
phonemes_array = [];
phonemesFreqStart = [];
phonemesFreqEnd   = [];
%% -----------------------------
```

FIG. 10: pulling out audio file and filtering using butterworth and plotting stft

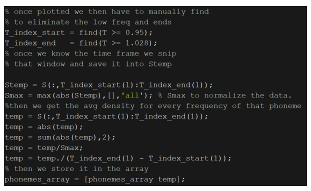
To start off, we first had to initialize our parameters such as nfft, Lwin, win, Noverlap, and create our arrays.



```
%% ------------------------------------------------------------------
graph = 0; % this turns on figures if set to 1
Letter = "b"; % this is to let us know what phoneme we are working on.
% then we pull out the audio file of the phoneme.
[data, fs] = audioread('TrainingData\b bug sound.m4a');
% then we run data through a high pass filter
% to eliminate the low freq noise
fc = 400; Wn = fc/(fs/2);
[b, a] = butter(4,Wn,'high');
data = filtfilt(b,a,data);
%after filtering we then do the spectrogram.
[S, F, T] = spectrogram(data, win, Noverlap, nfft, fs);
if (graph == 1) % this plots it if graph ==1
    figure;
    imagesc(T ,F ,abs(S));
    title(Letter + " signal");
end
```

FIG. 11: pulling out audio file and filtering using butterworth and plotting stft

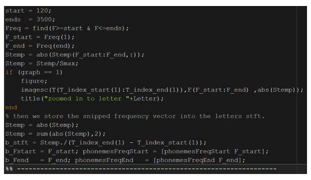
Then we have to load the signal and plot the spectrogram to see where the phoneme is in the time domain.



```
% once plotted we then have to manually find
% to eliminate the low freq and ends
T_index_start = find(T >= 0.95);
T_index_end   = find(T >= 1.028);
% once we know the time frame we snip
% that window and save it into Stemp

Stemp = S(:,T_index_start(1):T_index_end(1));
Smax = max(abs(Stemp),[],'all'); % Smax to normalize the data.
%then we get the avg density for every frequency of that phoneme
temp = S(:,T_index_start(1):T_index_end(1));
temp = abs(temp);
temp = sum(abs(temp),2);
temp = temp/Smax;
temp = temp./(T_index_end(1) - T_index_start(1));
% then we store it in the array
phonemes_array = [phonemes_array temp];
```

FIG. 12: snipping time window and storing it in Stemp

Once we find the time domain start and end of the phoneme we put that value in to find the time index start and end. With this we can now snip a window of time and store that matrix in Stemp. Once we do that we can then sum up the frequencies to have a magnitude vs freq vector. We then store this vector in phoneme_array.



```
start = 120;
ends  = 3500;
Freq = find(F>=start & F<=ends);
F_start = Freq(1);
F_end = Freq(end);
Stemp = abs(Stemp(F_start:F_end,:));
Stemp = Stemp/Smax;
if (graph == 1)
    figure;
    imagesc(T(T_index_start(1):T_index_end(1)),F(F_start:F_end) ,abs(Stemp));
    title("zoomed in to letter "+Letter);
end
% then we store the snipped frequency vector into the letters stft.
Stemp = abs(Stemp);
Stemp = sum(abs(Stemp),2);
b_stft = Stemp./(T_index_end(1) - T_index_start(1));
b_Fstart = F_start; phonemesFreqStart = [phonemesFreqStart F_start];
b_Fend   = F_end; phonemesFreqEnd   = [phonemesFreqEnd F_end];
%% ------------------------------------------------------------
```

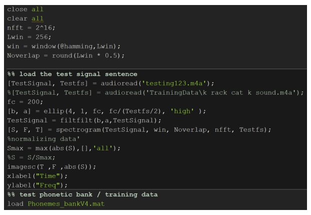
FIG. 13: Snips active frequency range and stores it

Then we also stored a snippet of the frequency vector where we would cut off the frequency when there was no activity. So for example, the ‘b’ sound only had activity from 120-3500 Hz. We then store this value inside b_stft. We then repeat this process for all 44 phonemes.

Then we have our main code that is supposed to detect phonemes for speech to text. Here you can see that we are initializing the variables and pulling out the audio file for our test. We then plot this spectrogram to see what it looks like. Then we load our phonemes bank.

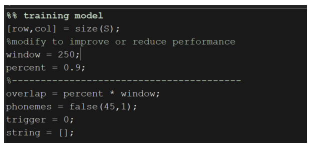
Once we load the data we then have to set parameters for our window that is shifting across our speech signal.

Then we step into our for loop which is moving our window across our speech signal. In each window, we are resetting the correlation array and creating our avg density for each frequency. This is stored in Stemp.



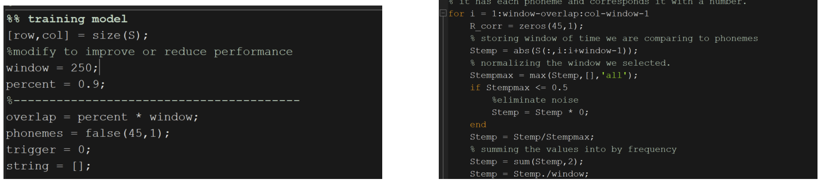
```
close all
clear all
nfft = 2^16;
Lwin = 256;
win = window(@hamming,Lwin);
Noverlap = round(Lwin * 0.5);

%% load the test signal sentence
[TestSignal, Testfs] = audioread('testing123.m4a');
%[TestSignal, Testfs] = audioread('TrainingData\k rack cat k sound.m4a');
fc = 200;
[b, a] = ellip(4, 1, fc, fc/(Testfs/2), 'high' );
TestSignal = filtfilt(b,a,TestSignal);
[S, F, T] = spectrogram(TestSignal, win, Noverlap, nfft, Testfs);
%normalizing data'
Smax = max(abs(S),[],'all');
%S = S/Smax;
imagesc(T ,F ,abs(S));
xlabel("Time");
ylabel("Freq");
%% test phonetic bank / training data
load Phonemes_bankV4.mat
```

FIG. 14: Initializing parameters and pulling test audio out to filter and stft



```
%% training model
[row,col] = size(S);
%modify to improve or reduce performance
window = 250;
percent = 0.9;
%----------------------------------------
overlap = percent * window;
phonemes = false(45,1);
trigger = 0;
string = [];
```

FIG. 15: Initializing window shifting parameters

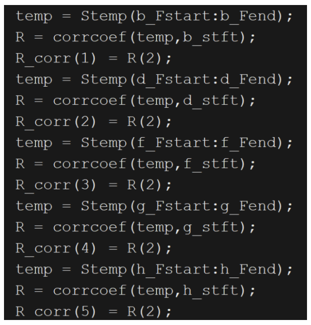


```
%% training model
[row,col] = size(S);
%modify to improve or reduce performance
window = 250;
percent = 0.9;
%----------------------------------------
overlap = percent * window;
phonemes = false(45,1);
trigger = 0;
string = [];
```

(a) Grouping different peak frequency ranges to get avg density

```
% phoneme list order is from this site we used
% https://www.dyslexia-reading-well.com/44-phonemes-in-english.html
% it has each phoneme and corresponds it with a number.
for i = 1:window-overlap:col-window-1
    R_corr = zeros(45,1);
    % storing window of time we are comparing to phonemes
    Stemp = abs(S(:,i:i+window-1));
    % normalizing the window we selected.
    Stempmax = max(Stemp,[],'all');
    if Stempmax <= 0.5
        %eliminate noise
        Stemp = Stemp * 0;
    end
    Stemp = Stemp/Stempmax;
    % summing the values into by frequency
    Stemp = sum(Stemp,2);
    Stemp = Stemp./window;
```

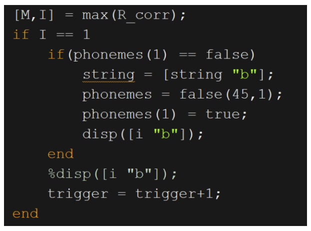
(b) For loop that shifts window



```
temp = Stemp(b_Fstart:b_Fend);
R = corrcoef(temp,b_stft);
R_corr(1) = R(2);
temp = Stemp(d_Fstart:d_Fend);
R = corrcoef(temp,d_stft);
R_corr(2) = R(2);
temp = Stemp(f_Fstart:f_Fend);
R = corrcoef(temp,f_stft);
R_corr(3) = R(2);
temp = Stemp(g_Fstart:g_Fend);
R = corrcoef(temp,g_stft);
R_corr(4) = R(2);
temp = Stemp(h_Fstart:h_Fend);
R = corrcoef(temp,h_stft);
R_corr(5) = R(2);
```

FIG. 16: correlation of all phonemes with current window

This is then referenced to create the different frequency ranges for each phoneme. Then we cross correlate each phoneme with the current window.

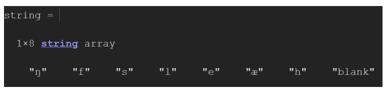


```
[M,I] = max(R_corr);
if I == 1
    if(phonemes(1) == false)
        string = [string "b"];
        phonemes = false(45,1);
        phonemes(1) = true;
        disp([i "b"]);
    end
    %disp([i "b"]);
    trigger = trigger+1;
end
```

FIG. 17: finds highest correlation phoneme and sets it as current phoneme of current window

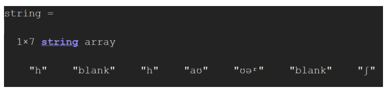
Once we have each phoneme we use the max function to find the phoneme that correlates the most with the current window. With the max function we can then correlate the index with each phoneme and have that phoneme be printed into our string array. We also have an if statement for phonemes where if a phoneme was detected in the previous window, it would not be called again and be skipped this iteration until a different phoneme is detected.

Below we can see the results that are outputted for our audio file /f/, fat, we can see that we are detecting other phonemes besides /f/ and /a/ and we also did not detect the /t/ sound. We also see that for the h hop file where the /h/ and /h/ and /o/ are detected but the /p/ sound was not detected. As for our final test you can see it did not perform well with the /ch/ chip watch audio file where we can see false phonemes being picked up and the /t/ and /w/ sound were not detected at all.



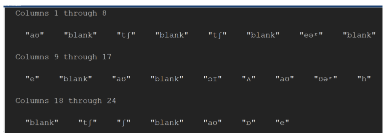
```
string =

  1×8 string array

    "ŋ"    "f"    "s"    "l"    "e"    "æ"    "h"    "blank"
```

(a) Result of audio f fat



```
string =

  1×7 string array

    "h"    "blank"    "h"    "aʊ"    "ʊəʳ"    "blank"    "ʃ"
```

(b) Result of h hop



```
  Columns 1 through 8

    "aʊ"    "blank"    "tʃ"    "blank"    "tʃ"    "blank"    "eəʳ"    "blank"

  Columns 9 through 17

    "e"    "blank"    "aʊ"    "blank"    "ɔɪ"    "ʌ"    "aʊ"    "ʊəʳ"    "h"

  Columns 18 through 24

    "blank"    "tʃ"    "ʃ"    "blank"    "aʊ"    "ɒ"    "e"
```

(c) Result of ch chip watch

FIG. 18: All audio test results

VIII. CONCLUSIONS

In conclusion, we noted that we still had issues with our signal where there seemed to always be background noise. This can be due to the mic quality or just noise from the pc fan or etc. While we think the corrcoef method seems to be good, we think that to further improve it. We would need better pronunciation of the phonemes and a bigger sample data to get the average with. We also saw the filters helped us greatly to reduce the noise such as the butterworth to eliminate low frequency noise and high frequency noise.

IX. ACKNOWLEDGMENTS

We thank Dr. Won (California State University, Los Angeles) for providing information for Biomedical Devices.

[1] “Deaf history timeline,” Deaf History Timeline. [Online]. Available: https://projects.iq.harvard.edu/asl/deaf-history-timeline. [Accessed: 15-May-2022].
[2] “Cochlear implants,” Mayo Clinic, 10-May-2022. [Online]. Available: https://mayocl.in/37MiPnd. [Accessed: 15-May-2022].
[3] “Deaf employment reports,” Gallaudet University, 12-Jan-2021. [Online]. Available: https://bit.ly/3NfQaWJ [Accessed: 15-May-2022].
[4] “Quick statistics about hearing,” National Institute of Deafness and Other Communication Disorders. [Online]. Available: https://www.nidcd.nih.gov/health/statistics/quick-statistics-hearing. [Accessed: 13-May-2022].
[5] Morgan deBlecourt and M. deBlecourt, “Hearing aids vs. Cochlear implants,” Duke Health. [Online]. Available: https://www.dukehealth.org/blog/hearing-aids-vs-cochlear-implants. [Accessed: 14-May-2022].