# DIGITAL SIGNAL PROCESSING: VOICE RECOGNITION

EEC 201 Final Project

Winter Quarter

Prof. Zhi Ding

Victor Madrid, George Martin

gtmartin@ucdavis.edu, vfmadrid@ucdavis.edu

## Contents

## Abstract

For this project, we sought to isolate the fundamental features unique to a person's voice and use this information to identify the speaker out of a set of potential speakers. To accomplish this, a set of audio files for testing and training were analyzed using frequency transformation, warping to a natural scale akin to the human perceptible audio range, and grouping by power density within this range. A standard algorithm was constructed to process data and reduce its complexity to a set of fundamental values, which were then compared to determine which known speaker's characteristics had the highest similarity. The code generated for this project was able to achieve a 100% accuracy from: the test data provided, self-recorded data, and data found online. This proves to be much better than a human benchmark which averaged around 54%.

## Background

### Pre-Processing

#### *Normalization*

The initial requirement of this project was to process the voice data in order identify distinct features and make the comparison problem possible when the test and training set are generated separately for the same speaker. This is because a person's voice is not perfectly reproduced each time that they say the same word or phrase. Audio files in .wav format were provided for both a training and testing set, which provided a time series of audio amplitude data along with a sampling frequency of speakers saying the word 'zero.' This data has variance in amplitude as well as some consistent distortion and must be normalized. To do this, the data is normalized to the range [-1,1] and the average value of this data is subtracted to produce audio date which only has frequency change included.

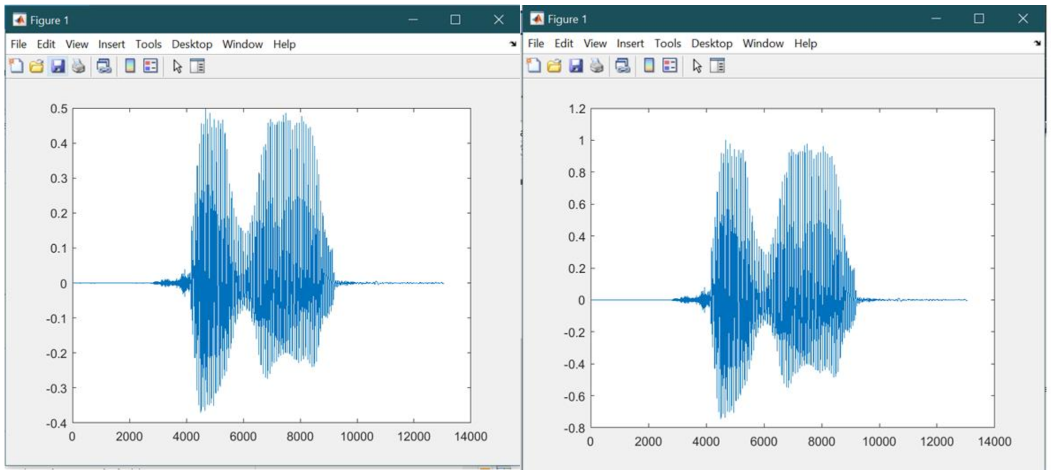


*Figure 1 – Data Normalization*

#### *Decomposition and Vectorization*

The process of processing a voice file is sequential and reproduceable for any .wav file. First, the data is imported from the file, and frames of the file are generated. This is because in short intervals, a voice can be considered constant and frequency decomposition can be performed. Next, a Hamming window is applied to each frame to prevent side-lobe issues, and the Fourier transform is used to translate this information into the frequency domain. This data is still in a linear frequency scale, which is not how audio is perceived by human ears, so the data is wrapped using Mel frequency distortion, which is linear up to 1000 Hz and logarithmic beyond that. Finally, the data is filtered through a Mel frequency bank and the power of each resulting filter is computed. This provides and analogous set of values for the respective voice power in each frequency range. The full process is described as shown below.

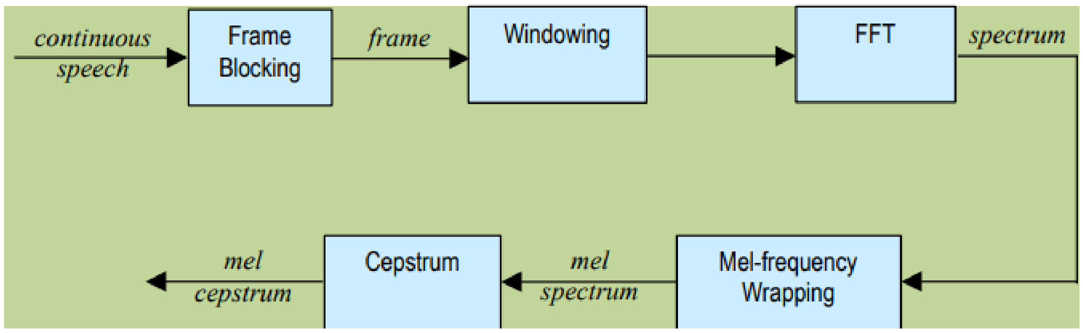


*Figure 2 - Preprocessing Flowchart*

## Processing

### *Training*

After each frame is converted into a vector of respective power in the Mel frequency domain, these vectors can be analyzed to find the average value of their features. This is achieved by creating a set of centroids which correspond to clusters of the generated vectors for use in comparison to a test set. The algorithm which achieves this is described as follows:

1. **Centroid Identification**
   - 1.1. For each voice file, the centroid of each vector-quantized frame cepstrum is calculated
   - 1.2. This centroid is doubled, by creating two new vectors multiplied by a fixed scalar value
   - 1.3. For each new centroid, all vectors are compared and sorted into 'bins' according to their closest centroid
   - 1.4. Step (iii) is repeated 100 times (answers converge sooner but this was found to not be computationally expensive, so additional repetition was determined as acceptable to verify centroid location)
   - 1.5. Steps (ii) through (iv) are repeated until a codebook of 8 vectors is generated for each voice file.
2. **Codebook Generation**
   - 2.1. The codebook is exported as a CSV file and saved so that users can efficiently run the test program without being required to re-train each time MATLAB is opened.
   - 2.2. The codebook vectors are listed sequentially, using the knowledge that each vector is 20 terms in length and each voice has 8 vectors to reorganize the data later.

### *Testing*

Once training is complete and a codebook is created with vectors corresponding to cluster centroids, this data can be used to identify similarities from a test file. The process is performed using the following algorithm.

1. Centroid Matching
   1.1. For each quantized frame, calculate the Euclidean distance to each codebook vector of a voice set
   1.2. The minimum distance and corresponding voice is identified and recorded
   1.3. Steps (i) and (ii) are repeated for each frame
2. Speaker Similarity Comparison
   2.1. After generating a match for each frame, this data is used to make a statistical decision of the matching speaker.
   2.2. The minimums of each frame are logged in a matrix assigned to each voice file and an average Euclidean distance is calculated.
   2.3. The minimum column of this matrix is stored outside the loop while the process repeats for the remaining voices.
   2.4. The minimum of these stored mean distances corresponds to the speaker.

## Testing

A series of prescribed tests were performed to monitor progress of the algorithm's development as well as validate the quality of the results. Each test builds upon the progress of previous tests and results in the completion of a robust voice recognition software.

### Test 1

To create a benchmark, the voice recognition ability of two human listeners was recorded. To do this, all training audio voice files were played twice so that the listeners could become familiar with each voice. Next, test files were played in a random order and the listener was asked to guess which of the training voices had spoken. The first listener (George) was able to guess five of the 9 voice samples, and the second listener (Victor) was able to correctly guess 6 of the speakers. Since these values are similar and well above any threshold of random guessing, they were determined to be a good baseline as to human performance.

### Test 2

Sampling rate for all recordings is 12500 samples/second. There are 20.46 milliseconds per block of 256 samples. After running the STFT on all samples we can see that most of the samples contain the most energy under 2KHz. Which is expected since the samples are human voices that are normally lower in frequency. After analyzing the data, it is easier to see the dispersion of energy with a lower N.

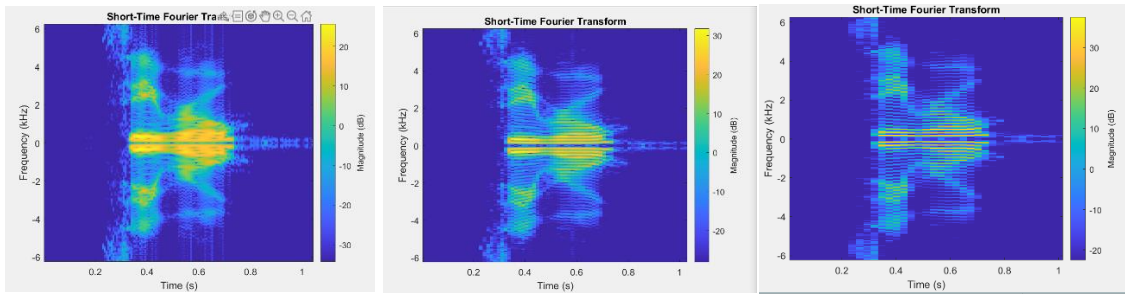


*Figure 3 - STFT using frames of 128, 256, and 512 data points, respectively*

## Test 3

Mel filter bank responses were calculated and plotted to compare to theoretical values. This is in good agreement with theoretical values, as the filters are linearly spaced and equally sized in lower frequencies but grow in larger frequencies corresponding to the logarithmic compression of the Mel frequency mapping.

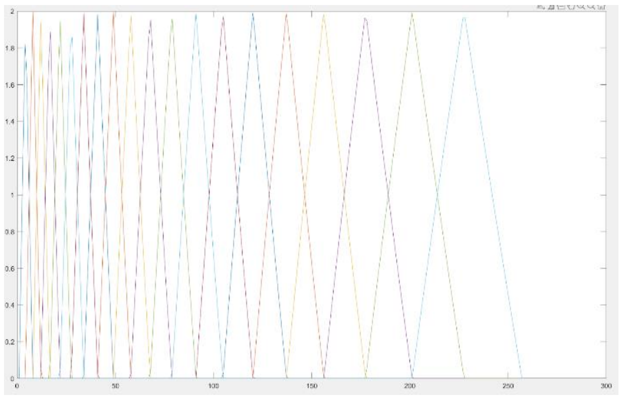


*Figure 4 - Mel Filter Bank*

The data is frequency domain, when mapped to the Mel domain, has a more consistent power spectral density as shown below. This is useful as it implies the coefficients computed using the Mel filter bank will have values of equal relevancy in voice recognition analysis. This is to say that the coefficients, while having a varying frequency band, correlate to a similar amount of significance in the human voice, and changes to each coefficient will be of similar significance.

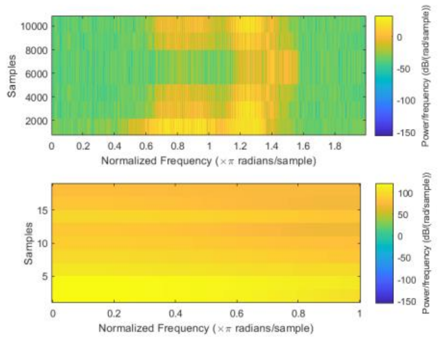


*Figure 5 - Spectrogram before and after Mel Frequency Wrapping*

## Test 4

All processing steps were compiled into one Matlab file which processes the data from .wav file to completed vectorization. This file is available at the project GitHub repository.

## Test 5 and 6

Tests five and six involved plotting each of the vectors created via processing, as well as the centroids generated. As the system is has nineteen dimensions, two coefficients were selected to produce a legible graph of the centroids. As shown below, the generated centroids have appeared to be properly placed near clusters of data points. Any variance can be attributed to the positions of the centroids in the other dimensions, as this is only a two-dimensional slice of a much higher order vector mapping.

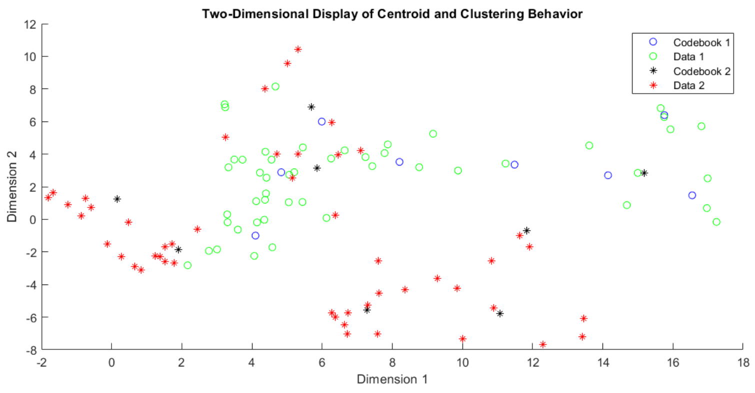
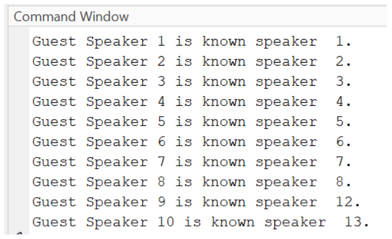


*Figure 6 - Frame Vectors and Centroids of Two Voice Samples*

## Test 7

The algorithm was able to achieve a 100% success rate, as shown below.


Command Window

```
Guest Speaker 1 is known speaker  1.
Guest Speaker 2 is known speaker  2.
Guest Speaker 3 is known speaker  3.
Guest Speaker 4 is known speaker  4.
Guest Speaker 5 is known speaker  5.
Guest Speaker 6 is known speaker  6.
Guest Speaker 7 is known speaker  7.
Guest Speaker 8 is known speaker  8.
Guest Speaker 9 is known speaker  12.
Guest Speaker 10 is known speaker  13.
```

*Figure 7 - Test File Output*

Since there are 3 unused training set speakers, the numbers of respective test and train speakers are mismatched above the 8th speaker. This is due to file organization and not the performance of the code. The 9th test file is made by the same speaker as the 12th training file, and the same is true for the 10th test file and 13th training file.

## Test 8

Notch filters tested from a normalized range of .05 to 1 do not reduce the performance of the system. This is most likely due to the 19-dimensional centroid set used for analysis. When testing with a band-stop filter, one can generate a filter which can diminish the performance of the program. When filtering, a fixed size stopband in the high frequency range has a larger impact than the low frequency range, due to the logarithmic mapping done above 1000 Hz. This results in the distortion of fewer coefficients when removing the 5000-5500 Hz band compared to the 500-1000 Hz band, for example.

### Test 9

As shown in Test 7, with three additional speakers the system achieves 100% accuracy.

### Test 10

Finding data online was proven to not be practical due to the fact that the team was looking for two different recordings of someone saying the same thing. Instead of testing our algorithm on the one same training/training data found online, we decided it would be best to incorporate something different. After recording ourselves utilizing the microphones on our computers, as seen in training data 12,13 and testing data s9 and s10, it was decided that the best way to test the robustness of our algorithm was to incorporate a different method of voice recording. This method moved from the teams' computers to the teams phones. Since the phone has a much different quality of recording because of its smaller processor, different compression, etc. the team believed it could introduce a new edge case for the system. The team added the recording s11 and train14 that was recorded from a person outside of the team with an google pixel phone. In conclusion it didn't make a difference and the algorithm was able to still match the two together.

## Results and Discussion

### Findings

Several key features of the algorithm were tested to illuminate the performance of underlying mathematical structures. Important variables include frame width, window type, Mel frequency mapping properties, centroid search parameters and codebook size. Additionally, the algorithm used to determine the speaker required several iterations to determine the optimal method to process the data.

Frame width was found to be a key component of the algorithm, with a width of 512 being the optimal size. Powers of 2 were selected for frame testing as this is results in the best performance of the FFT from a computational cost standpoint, and from 64 to 512 incremental gains are shown each time the frame size is doubled. Above this, the frame is too wide and key features cannot be found, resulting in poor performance barely above random chance. The Hamming window was tested as well as Kaiser and Hann windows. The hamming window was selected as the best performing considering the ease of implementation and slight improvement of performance. At one point in development, the hamming window was able to produce codebooks with 1 additional correct guess.

Codebook generation was tested rigorously to find optimal parameters as well. Performance gains were again shown when doubling the codebook up to a size of eight per voice, which when presented graphically makes intuitive sense as the centroids moved closer to the centers of clusters. Below eight, centroids were occasionally found between two clusters due to the lack of ability to have a centroid within each cluster. Above eight, centroids remained within clusters but occasionally an over-fitted centroid was detected which reintroduced error in the performance of the code. Centroid iteration was test from a range of one to 100 iterations before doubling, and around five to ten iterations proved to be sufficient with the given test set. However, as it is not computationally taxing to recalculate centroids without doubling, 100 iterations were chosen to eliminate any unforeseen solver issues given by insufficient iterations.

The testing algorithm was revised several times to find the successful version presented here. Features such as total average distortion by voice, weighted guessing based on the minimum of each feature (i.e. low frequencies being considered more important as well as high frequencies). Ultimately the algorithm that determined the nearest codebook vector, stored this information, and then found the corresponding voice with the lowest overall distortion was found to work the best of all methods attempted.

## Future Work

The algorithm used in codebook generation is not refined, and future work can be done to analyze and dynamically replace ill-fitting centroids during the doubling process which should further improve performance. Even though efforts were made to make the code as efficient as possible, there is plenty of room for optimization improvement as well as the addition of code to make operation of the software more seamless and user friendly, such as integrating a hash map to the train and test files which would remove the naming restrictions and allow for more self-explanatory naming conventions. Above all, however, work can be done to find the minimum requirements that achieve a satisfactory performance based on the application of the software, such as identifying which vectors of the filter bank have the most significant impact. During testing it was determined that all vector values are of similar significance, but the Mel frequency warping can be tuned to more exactly the distribution of power density on an average voice.