“Look Who’s Talking”

18-551, Fall 2007, Group 2, Final Report

Han Joo Chae (hchae)
Matt Keagle (mkeagle)
Andrew Lam (aclam)

1. Project Details

In seeking to prevent more disastrous attacks on the United States, many different spying devices will be used in the field of battle. These bugging devices are very useful in monitoring the location and activities of important and dangerous terrorist cells. However, with the introduction of these monitoring devices there exists a need for individuals to pour over the information that is constantly being recorded. Without having knowledge of what type of information is being monitored, all of the continuous data must be scanned for useful information. Because of the large amount of raw information streaming from the bugging devices, it is possible for someone viewing or listening to the stream to overlook key aspects of the conversations that could lead to preventing a terrorist attack. Another possibility would be that once the moderator detects an important message, because it is nearly impossible to keep up with the real time stream of data, once the important speech has been heard and understood, there is a high possibility that the event has already occurred or is too late to be prevented. Overall, scanning the large amounts of data generate from bugging devices proves to be too time consuming and a waste of man-hours that could be devoted to more important tasks.

In order to solve this problem, we proposed that several algorithms be developed to process the data in real time so that the information can be parsed and deemed as appropriate or urgent. By marking the information as urgent, those people tasked with analyzing the data could work on other tasks while waiting for the algorithms we developed to highlight the necessary speech and when to search. This not only provides the analysts time to work on other tasks, but also allows more immediate access to critical and time sensitive information that might have otherwise been overlooked. We identified three characteristics of the information that bugs receive that would identify the incoming raw data as important speech. These three algorithms are voice activity detection (VAD), speaker identification (SID), and keyword spotting.

With voice activity detection, the user would never have to wait around for someone to speak in the conversation because the user would have the ability to filter out the non-voiced signals and be notified of the presence of speech. Speaker identification would be done in a non-text based fashion where we do not have the luxury of having the terrorists speak a particular set of phrases and rely on them to speak them again in their conversations. However, even though no specific words can be identified and trained upon, there would be a large amount of available speech from previous recordings of the wanted individual that can be used for training. Large amounts of training data allow non-text based speaker independent systems to become viable within the scope of the problem. The final algorithm to be developed is user independent keyword spotting which would be able to identify trained

keywords when they are spoken by any individual being bugged. This would allow the user of the system to characterize the information that they are attempting to monitor and greatly increase the likelihood that the information identified will be pertinent. All of these elements combined would greatly improve the practicality and efficiency of bugging devices.

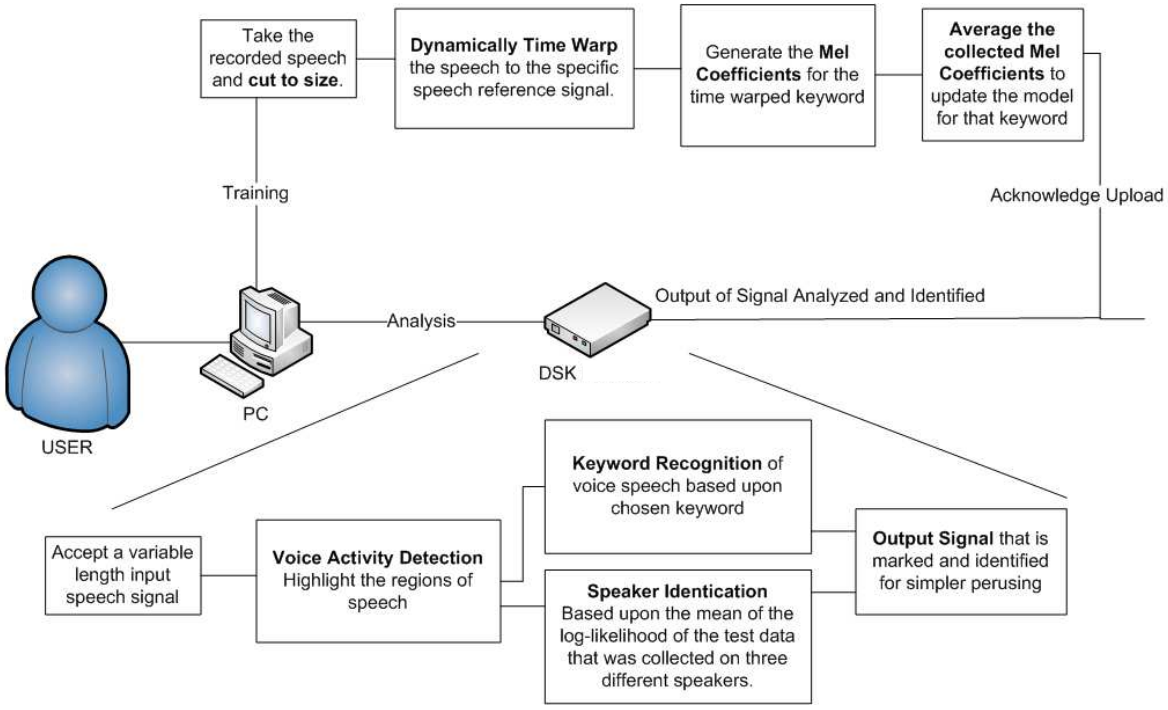
2. Previous Projects

Previous 18-551 projects have worked in the area of keyword recognition and have adapted some of the algorithms used in our project for different purposes. Here we will detail their work and how it differs from the work in our project. There have been prior projects that have done keyword recognition, specifically in Spring '00 when a group performed isolated keyword recognition in non-continuous speech using a text-based approach to model the speaker's in the training set. This work matched ours in that it matched the keywords; however, our case uses a non-text based approach and can be for any speaker of the keyword. Other than this group no group has specifically attempted to identify keywords or speakers from modeled set of users.

Groups in previous semesters have used some of the algorithms that we are using in our project such as Dynamic Time Warping (DTW) and our standard deviation approach to voice activity detection. The DTW performed by the other groups was used to morph different speaker's voices to others by helping to time align the data to the reference and not in an attempt to model the keyword. In Spring '04, Group 11 was able to code the DTW algorithm for their purposes on the DSK and in Fall '06, Group 1 used the DTW algorithm programmed in Matlab for help with training data. Neither of these groups used the sliding model method of the DTW algorithm, which will be explained later, in order to determine the presence of a specific keyword in speech. Standard deviation has been used by Group 2 of Spring '00 and Group 6 of Spring '02 as well as many others in methods of image processing to determine a noise threshold but has never been implemented with voice activity detection to determine the presence of speech. This development is new to our project and was developed and implemented entirely by our group.

The new material to 18-551 is the speaker and keyword recognition in continuous speech. Continuous speech introduces the problems of separation of words within speech and other issues that come with the fact that people do not speak clearly in day-to-day speech. Also, Gaussian Mixture Models and speaker and text-independent methods have not been tried in previous projects.

3. Overview of the System



The system which we designed uses Matlab and the DSK for two different modes of operation. The initial mode of operation which needs to be performed is the training. This method is performed completely on the PC in Matlab in order to generate the models for each specific keyword and speaker. The initial Matlab training step for the keyword recognition is to cut the recorded speech given by the user for the specific keyword to the size of solely the word and no non-speech. Once the word has been recorded and cut to size, the user would then add it to the training set through the training GUI commands. Behind the scenes, Matlab would be dynamically time warping the input speech based upon a reference template for the word. This process eliminates the difference between how fast each person speaks the word and provides more accuracy for matching when comparing the keywords together in the next step. As stated, the training process then generates Mel coefficients from the time warped input signal and averages the coefficients with the stored Mel coefficients for that word. By averaging the Mel coefficients over time with different speakers and many iterations of the same word, you achieve a model of Mel coefficients that ideally represents all speakers of the same word. By using that reference to model all of the speakers we hoped to match any different speaker of the same word.

The training path for the speaker identification takes a longer, 60 second, sample recording from a single speaker in order to perform Gaussian Mixture Model representation of the speaker. This procedure was performed three times for the different speakers in our group and could be repeated for

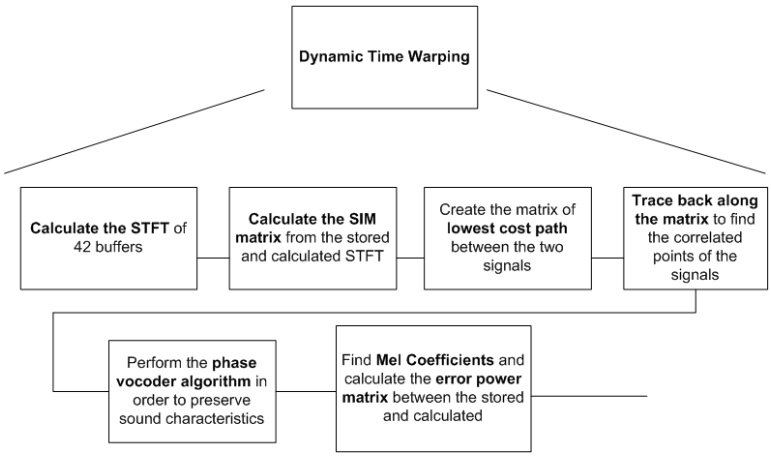
as many speakers as desired. The generated results can then be stored on the DSK and used to analyze which person the current speaker most resembles.

The analysis side of our system is performed on the DSK with all of the steps (VAD, DTW, and SID) all running in series to determine whether there is speech present, what speaker can be identified if they remain in the training set, and what words any of the unknown individuals are saying. First the raw data is processed by testing the standard deviation and determining whether or not it crosses the threshold to characterize it as speech. Once speech has been determined, the DSK will begin processing to determine the presence of the keyword and the identity of the speaker. The raw data is converted into the Short Term Fourier Transform (STFT) for the length of one keyword. A similarity matrix is generated from the STFT of the raw data and the STFT of the stored reference. This matrix is used to perform the dynamic time warping and generate the Mel coefficients that represent the time warped representation of the incoming signal. If the incoming signal matches that of the word that you are trying to identify then the time warped version of the signal should have Mel coefficients that are similar to the stored data, no matter the speaker. Once the Mel coefficients are generated the power difference from the stored Mel coefficients can be calculated and used to represent the similarity between the input word and the stored word. The smaller the power difference is equivalent to a higher similarity. The input to log likelihood calculations for SID is the resulting Mel coefficients from the DTW and the results of the log likelihood can be used to determine who out of the set of trained speakers the current speech represents. Processing continues every 24 ms for the incoming data repeating the same calculations, assuming that speech has been detected.

The results of the calculations would be the number representing the error power of the Mel coefficient matrices and the identified speaker from the set of trained speakers. The small amount of data would be transferred back to the PC where the results can be shown within a GUI. Although there are too many calculations to be performed for real time performance, given a faster processor with more internal memory, one day the process could be completed in real time.

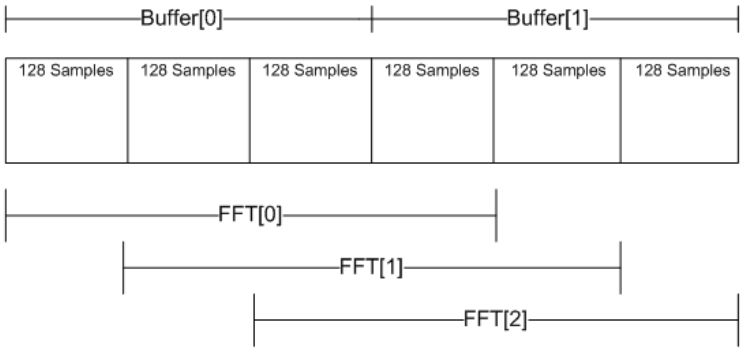
4. Description of the Algorithms

Keyword Spotting with Dynamic Time Warping



The keyword spotting method employs a sliding model comparison method utilizing Dynamic Time Warping to reliably characterize the difference between a word length chunk of incoming data and a reference template. [1] At a sampling rate of 16kHz every 24ms, the time for one block, 384 samples of raw input data are collected from the PC and sent to the DSK using a TCP/IP transfer. Three 512 length Fast Fourier Transforms (FFT) are performed every time a new block is received on the current block plus the past block with an entire block length of overlap between each FFT. This means that the step size of the FFT is 128 samples. These numbers were chosen for the step size and block length in order to simplify the calculations to allow integer multiples of FFTs to be performed every new block. Each of the three results of the FFT are stored into a circular buffer, which stores around 42 blocks worth of FFTs depending on the length of the word. The three least recently used FFTs are discarded because they are no longer need in the keyword spotting algorithm on the DSK.

The FFTs have overlap to counteract the lack of overlap in the storing of raw data. Because only the FFTs are used in the calculations there is no need to overlap the raw data and re-store the information. Using a 512 length FFT, only the past block data can affect the current data and therefore the amount of raw block data that need to be saved remains at two. An example of how the FFT works on the incoming block is shown below:



Once the data has been successfully converted into the STFT, the necessary calculations can be performed to time warp the incoming signal. Dynamic time warping uses a dynamic programming technique to calculate the lowest cost path between the stored signal and the incoming signal. From this lowest cost path you can determine which sections of the incoming speech correspond to the sections of the stored data. The matching parts of the incoming signal can then be aligned, skipping or deleting different parts so that you can determine the likelihood of matching the original signal without compromising the quality of the match from speaking the word at different speeds. The dynamic programming calculations listed in the following equation calculate the minimum of the surrounding matrices and add it to the diagonal.

$$DP(i+1, j+1) = DP(i+1, j+1) + \min(DP(i,j), DP(i+1,j), DP(i,j+1))$$

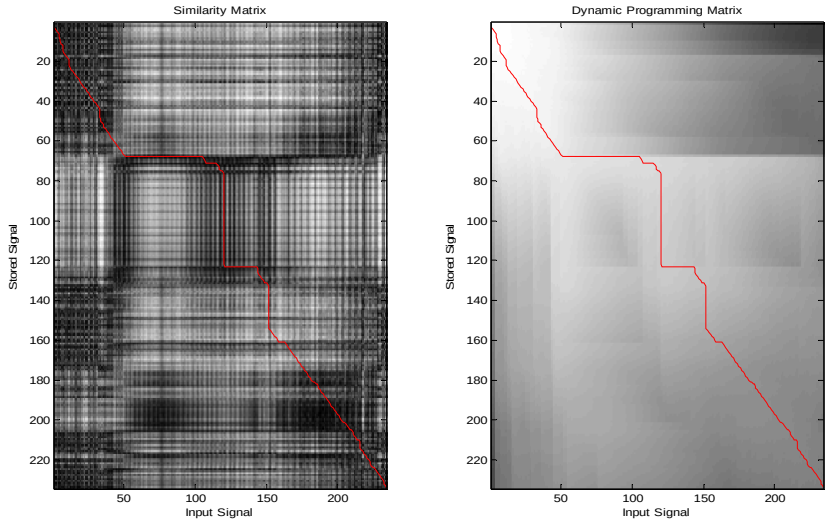


Figure: Lowest cost path through similarity matrix and DP matrix

This would result in the lowest path lying along the exact diagonal if the matrices were completely aligned. The two methods that we considered using in order to determine the match between the two signals was the error power matrix and the bottom right of the dynamic programming matrix which represented the lowest cost total distance between the two matrices. Throughout our implementations we reported both of the numbers in order to determine which of the two results functioned with more accuracy and stability amongst the many different users. The lowest cost path theoretically should be much lower for any word that is much closer to matching the original signal; however, the input signal STFT does not change from user to user only serves as a reference for the time warping, so when separate users begin to use the system, the validity of the match begins to fail. The averaged Mel coefficients on the other hand are able to account for

many different users and should theoretically increase in accuracy with more trained users and multiple training signals from each user. We witnessed these results in our tests and found that the averaged Mel coefficients from the different users had a much higher rate of success than the lowest cost number in predicting the presence of the word in continuous speech of multiple users.

After the dynamic programming matrix completes, following along the path of least cost allows the user to reconstruct the time aligned signal. This signal is now warped in magnitude to match the template data and has phase distortions that associate any warping in magnitude without the same warping in the phase characteristics. As it turns out, in our project the final result only comes from the magnitude characteristics that generate the Mel coefficients so the warping in the phase space does not affect our results. However, we did implement a phase vocoder algorithm, which will be briefly explained in the next section, to recalculate the correct phase of the time aligned signals. Our design flow continues with the phase realigned signal being entered into the Mel coefficient generator which converts the frequency characteristics into the Mel spectrum.

The Mel coefficients are generated by multiplying the frequency of the signal by weighted triangles of logarithmically increasing centered frequencies that are likely to match the harmonics of characterized speech. The Mel domain closely resembles the human ear in characteristics and it is for this reason that these coefficients are frequently used in speech processing for characterizing a speaker. Our group had considered implementing Linear Predictive Coefficients (LPCs); however, previous groups and various textbooks highly recommended Mel cepstrum coefficients due to their superior representation of speech. Once the realigned signal has been converted into the Mel frequency domain and filtered to represent the speech in coefficients, the result can be compared with the stored representation of the word from the training data by calculating the power between the two matrices. This number can serve as a result of the current input signal matching the reference.

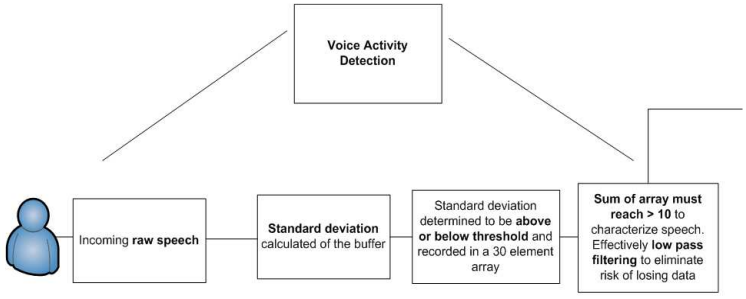
Once a long enough signal of approximately 10 seconds has elapsed, then the standard deviation of the error can be calculated to estimate the noise with the farther peaks portraying a positive match against the target keyword. After testing a threshold can be tested and mapped to determine when the keyword is officially detected in speech. Depending on the situation and the level of false alarms that are desired, the threshold can be tweaked. Given our story and project aims, more overall false alarms than misses are preferable in order to avoid not being alarmed for important and potentially life-threatening information. Ideally the peaks of the signal analyzed would be low enough to detect all speakers voicing the specified keyword and if the computations could be done fast enough then the process could be repeated

for multiple keywords in real time in order to even more accurately predict the conversation being held by the monitored suspects.

Phase Vocoder (a note for future projects)

After working on the project and researching the online implementations of the DTW algorithm in Matlab, we discovered that the work by Dan Ellis [1] to be far superior and easily read and understood. One of the parts of the DTW algorithm was the phase vocoder step that readjusted the phase characteristics of a time warped signal in order to preserve the time domain qualities. This step allows the signal to be inverted back into the time domain after being warped and still maintain some intelligibility as the original signal just at a different speed. The algorithm changes only the phase characteristics and does not alter the magnitude of the STFT of the time aligned signal, which made it irrelevant for our project. However, we did not realize this until the algorithm had been completely implemented on the DSK and working in its entirety. So future groups that which to implement a dynamic time warping algorithm that involves re-synthesizing the speech back into the time domain to play it, we have developed the code necessary.

Voice Activity Detection (VAD)



The Voice Activity Detection (VAD) uses a standard deviation to get the energy level of the signal to determine the presence of speech. We calculate the standard deviation of one block and if the standard deviation is greater than the threshold, we set that as high. In our DSK implementation, we use the block size of 24ms. To calculate the threshold, we used Cumulative Distribution Function in terms of error function defined as:

$$\Phi_{\mu,\sigma^2}(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right], \quad x \in \mathbb{R}. \qquad [7]$$

Since we assume there will be white noise, we set mean to zero. We also set the standard deviation to one to make the calculation easier and thus, we obtained the Probit function:

$$\text{probit}(p) = \Phi^{-1}(p) = \sqrt{2}\,\text{erf}^{-1}(2p-1). \qquad [7]$$

By putting 99.99% into p in the Probit function, we are able to get the number of standard deviation steps from zero. The level we get by multiplying the number of steps to the average standard deviation of noise guarantees 0.01% chance of that level or above being noise. However, 0.01% means that it has that high of a chance of having false alarm whenever VAD gets run and that is not sufficient. Thus, we find the Time till a False Alarm (TFA) and set it make it to have only maximum one error in one hour. That is:

$$T_{Fa} = (3600\text{secs}/0.024\text{secs})(\text{samples}) \approx 180{,}000 \text{ samples}$$

From the TFA, we obtained the Probability False Alarm that is:

$$P_{Fa} = 1/T_{Fa} = 1/180{,}000$$

Therefore, we get a new probability:

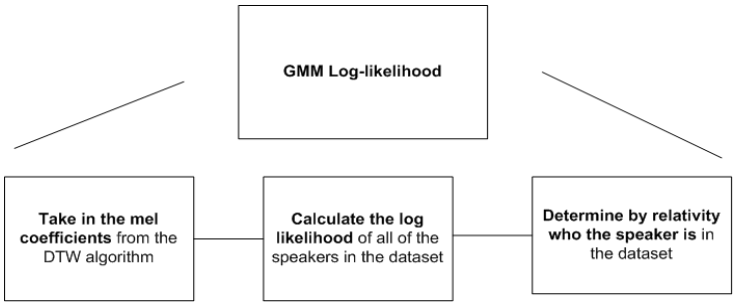
$$p = 1 - P_{Fa} = 99.9994\%$$

By putting that to the Probit function, we get 4.3376 steps from zero. Since the average standard deviation of noise we measured is 0.0008272, we get:

$$\text{Threshold} = 0.0008272 * 4.3376 = 0.003588$$

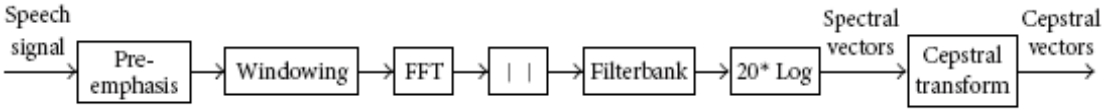
When the standard deviation of the signal goes higher than the threshold we set that as high. However, that is still not enough because it will ignore that silence in the middle of the speech and it will also pick up the clicking noise around. Thus, we use a buffer of 30 blocks. When there are more than ten blocks that are greater than the threshold, we say there is speech. This way, it is very simple to implement and works very well at the same time and that is why we use the method in our project.

Speaker Identification (SID)



Our system implements a text-independent closed-set automatic speaker identification system. This means that we will identify the speaker of unknown continuous speech from a set of trained speaker models. We use a combination of MFCC and GMM to train our models. We take the maximum log-likelihood as our identified speaker.

Speaker Identification consists of two phases: training and testing. In the training phase, we first take a length of input data and characterize the speech by some method. Currently MFCC is the most popular used parameter in Automatic Speaker Recognition systems since they are relatively easy to calculate and they are fairly good at representing the independent low level acoustics from speaker to speaker. Mel Frequency can be thought of a representation of human hearing. Just like our hearing capability, Mel Frequency is logarithmic and has more frequency resolution in the lower end of the spectrum and less resolution at higher frequencies[6].



Flow of MFCC Calculation [6]

To calculate the MFCCs, our system first takes the speech signal and applies a pre-emphasis filter in order to emphasize the higher frequencies [6]. This filter can be written as follows: x[t] = x[t] - .97x[t-1]. The signal in then windowed every 25ms with no overlap, by a hamming function. Traditionally, there is overlap but for the purpose of conserving cycles, we have opted for no overlap. A hamming function is used because there is less effect from the side lobes.

Next, we take a 512 point FFT and convert the log amplitudes of the result to the Mel scale. As stated before, the Mel scale is logarithmic such that there is greater granularity at lower frequencies, similar to human hearing. We then take the DCT of the result to get the cepstrum and we have our 13 MFCC. These coefficients is the used as input to train our Gaussian Mixture Model.

The GMM models our parameters by using a mixture of 16 weighted Gaussian densities. The weights sum to one and the density used is the likelihood function which is defined as:

$$p(\vec{x}|\lambda) = \sum_{i=1}^{M} w_i p_i(\vec{x}).$$ [6]

Where the p_i is the following distribution function:

$$p_i(\vec{x}) = \frac{1}{(2\pi)^{D/2} \left|\Sigma_i\right|^{1/2}} e^{-(1/2)(\vec{x}-\vec{\mu}_i)'\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)}$$ [6]

Usually the log-likelihood is calculated instead such that the likelihood for the set of frames can be added together instead of multiplying as shown below.

$$\log p(X|\lambda) = \frac{1}{T} \sum_t \log p(\vec{x}_t|\lambda)$$ [6]

Commercial systems use anywhere from 64 to 256 mixtures but due to the computational time needed, we opted for 16 gaussians. The GMM is a mixture model that maximizes the likelihood of the data using a mixture of Gaussians using the EM algorithm. The EM is an iterative algorithm that increases the log-likelihood such that for a new model $\underline{\lambda}$, $p(X|\underline{\lambda}) > p(X|\lambda)$. The re-estimated weights, means, and variances are calculated as follows.

Mixture Weights:

$$\overline{p}_i = \frac{1}{T} \sum_{t=1}^{T} p(i \mid \vec{x}_t, \lambda)$$

Means:

$$\vec{\overline{\mu}}_i = \frac{\sum_{t=1}^{T} p(i \mid \vec{x}_t, \lambda)\vec{x}_t}{\sum_{t=1}^{T} p(i \mid \vec{x}_t, \lambda)}$$

Variances:

$$\overline{\sigma}_i^2 = \frac{\sum_{t=1}^{T} p(i \mid \vec{x}_t, \lambda)x_t^2}{\sum_{t=1}^{T} p(i \mid \vec{x}_t, \lambda)} - \overline{\mu}_i^2$$

Usually 5-10 iterations of the EM algorithm are sufficient to reach an approximate convergence, meaning that the change in likelihood is less than some threshold.

Testing of the SID system is simply taking the test input speech, calculating the MFCC, and calculating the likelihood of the test speech signal with each speaker model in the set. The speaker model that returns the maximum likelihood score is naively identified as the test speaker. This is the least computationally intensive decision algorithm, which is why it used in this project.

5. Training, Testing, and Results

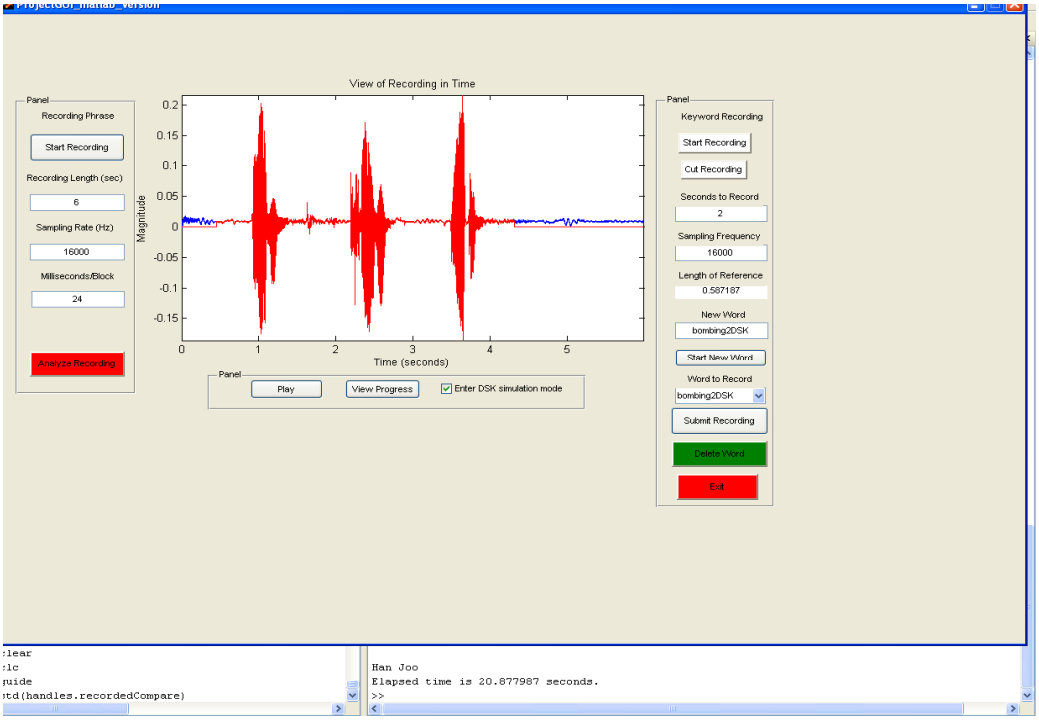


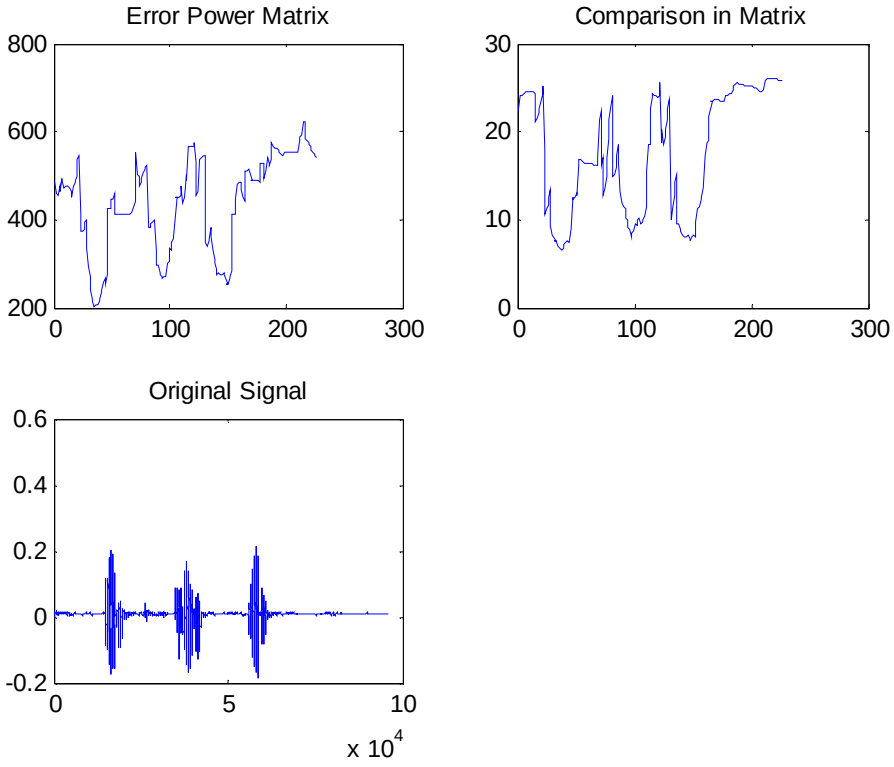
Figure. Our GUI for training and testing.

The training and testing for the algorithms that we developed are both performed through the GUI that is pictured above. The broad range of functions in our GUI allowed us to simply and efficiently add and train new words into the data set. Also, the GUI was the window for recording the variable length segment of speech to be analyzed with the three algorithms. In order to determine the potential capabilities of the DSK in determining the presence of a keyword and identify a speaker, we implemented a DSK simulation mode in Matlab so that the results could be quickly analyzed and interpreted with different sampling rates and block lengths. For the dynamic time warping application it was crucial that the user was able to by hand choose the length of the word that was recorded and averaged because the algorithm had a limit on the amount that it could warp the input signal. By

keeping the incoming keyword recordings to the same length we were able to maintain an accurate representation of the keyword in Mel coefficients. When testing the validity of the keyword we used the DSK simulation mode in order to show what the results of our C code on the DSK would produce.

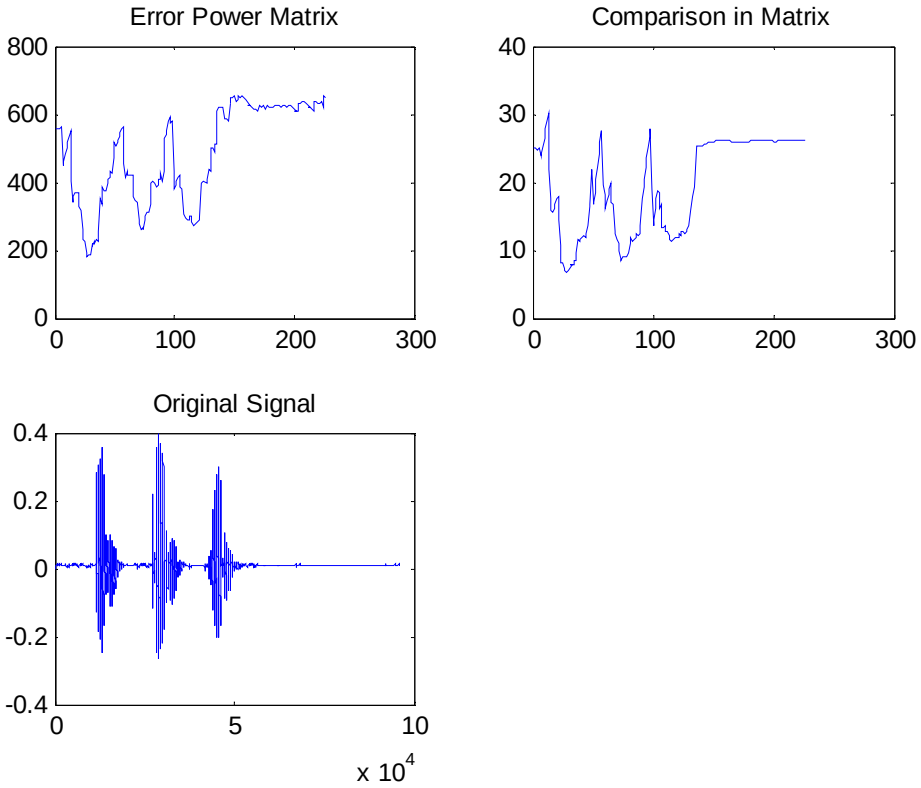
Result for Keyword Spotting

Below are some figures of the transformation and success of the recognition of the keyword in speech as the keyword is adapted with multiple trained users and an untrained speaker. As you can see in the diagram of figure one, the system will clearly identify the specific keyword when the keyword has been trained only with the person who is currently speaking.



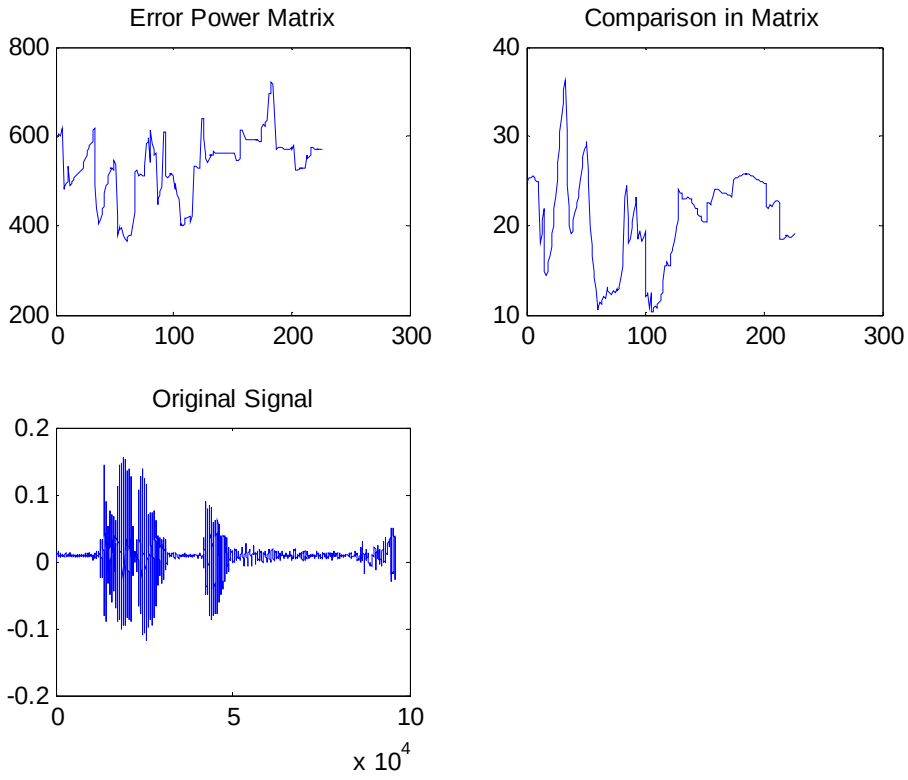
The graph to highlight is the graph in the upper left which is the results of the error power over time with the recorded speech below. The speaker was the only person trained to the word and they spoke the words "bombing, climbing, and robbing" with pauses in between the words. Bombing was the word that was trained for the specific user and as you can see it has the lowest peak of the three words which means that the Mel coefficients generated for that word match closest to the stored coefficients. The difference between the peaks is 13 percent of the entire peak of the matched word. Even though the other words which sound similar to the trained word are peaks in the data, they can be clearly identified as lying below the matched keyword.

This time we trained the data with two more speakers to attempt to morph the keyword to be speaker independent and reliable at detecting the speaking of bombing for any user. After training three speakers, each 5 times, the third speaker repeated the same three words as the first speaker and the results are shown below.



As you can see in this figure, the mean of the data is much higher at 575 because the Mel coefficients have been warped for multiple users so they are not fundamentally a match for each of the speakers. However, the peaks themselves match almost exactly to the peaks of the first trained data set with just one person so the speaker independent capability of our keyword spotting algorithm seemed to be working at this time.

Next we tasked a random individual with speaking the word bombing in a sentence to determine if our method was truly speaker independently recognizing the keywords. The results are shown below of one of the members of another 18-551 classmate speaking bombing in a sentence. The sentence was not necessarily clearly spoken nor was the speech separated by word so the results would test the robustness of our method in the worst case scenario.



The results shown above were promising although not as profound as when the speaker was in the trained data set. The lowest peak in the upper left graph was in fact the word bombing in the sentence. However, the peak was not as low as the other results and there were some other low peaks in the data that were not the word bombing. This showed that the speech was harder to recognize when the words were more blurred together and the speech was more rapid. The results did show though that the keyword bombing was the lowest peak of the entire sentence, however the difference between the keyword and the next highest peak was only 4 percent of the mean.

We believe that these results were promising in showing that an unknown speaker could speak into the bug and have the specific trained keyword be recognized by our method. The margin of error would be high in the fact that the peaks of the recognized keyword would not differ much from the noise level, however with more trained users and with a higher sampling rate, we believe that the results would be more robust and accurate. After performing the same calculations on the DSK, the results showed to be almost exactly the same as the results in the Matlab simulation with some rounding errors that only affected numbers up to the second decimal place. We also noticed that the C code implementation that we developed in the Unix environment was different than the Matlab implantation and DSK implementation even though the code in the Unix environment and DSK environment were exactly the same. The error was very negligible and we

believe that it can be attributed to the difference in the <math.h> libraries of each of the respective coding environments.

Results for SID

Speaker Models for the Speaker Identification algorithm are trained using existing MATLAB code for GMMs and MFCC. This is done by recording long inputs of speech data from each model. The speech is then windowed into 24ms windows and the MFCC's are then calculated for each of the windows. The multi-gaussian parameters that maximize the log-likelihood for the entire set of MFCC are then calculated. Stored for each model are the means, variances, and the weights of each of the Gaussians in this mixture. As stated previously, we use 16 Gaussians with 13 MFCC to define each model. In the testing phase, the model that maximizes the likelihood of the given test speech input is selected as the identified speaker. Below are samples from one such test.

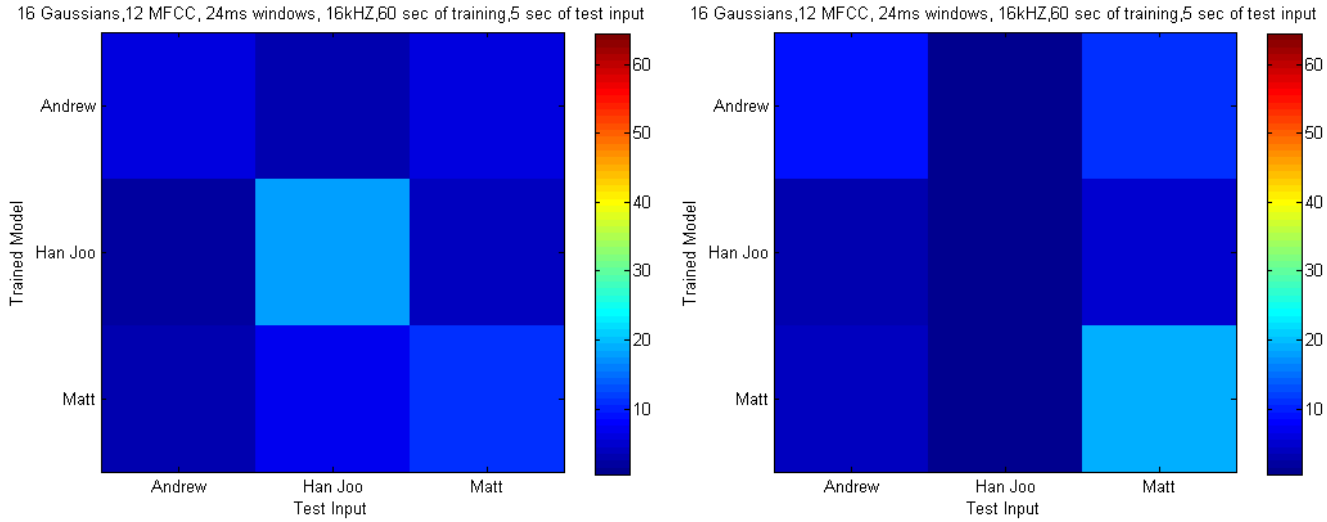
Test / Model	Test Andrew	Test HanJoo	Test Matt
Model Andrew	**-9.2926**	-11.6856	-9.0887
Model HanJoo	-10.3661	**-11.1870**	-9.7759
Model Matt	-9.9598	-11.7052	**-8.5416**

Test result from 5 seconds of test input

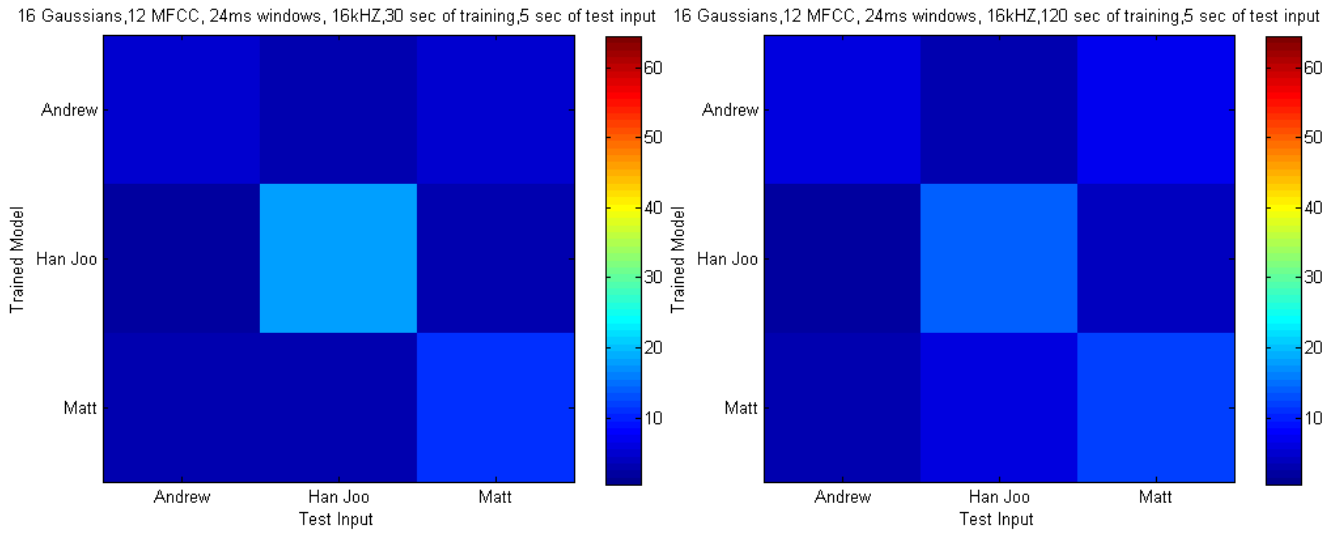
Ideally for each test input speech, the model that maximizes the likelihood is the same person as the test input speech (i.e. the diagonals are the maximum value in each column as shown in the sample results above). Training a balanced model, that has data from the entire range of someone's voice, poses a huge challenge in any SID system. In commercial speech verification systems, users are usually told to read several pages of text as naturally as possible in order to capture phonemically balanced data. Our models were recorded in just 60 seconds and the speakers were told to say anything they wanted in a natural everyday voice. This method worked amazingly well under certain controlled conditions, otherwise worked somewhat sporadically. The accuracy of the SID system is highly dependent on how close the input speech is like the model, and the correct ID percent rate was usually near perfect when a model is recorded at around the same time as the sample test

input. However, slight variations in voice, such as mood or drowsiness from working too much, can throw off the SID system as shown in the next figure.

The layouts of the results are in the same format as before, except normalized and utilizing the MATLAB image function for ease of view. Notice the middle column in each of the results. The left result is one such case where each person was talking fairly close to the model data such that the diagonal is the highest value (or lightest blue) for each test input. The right result is where Han Joo spoke with a little more excitement in his voice than usual, where as in the training model, he sounded exhausted and tired. As seen in the results, with the 'excited Han Joo' data, the identified speaker is not very clear.



More training data or more test data is not necessarily going to produce more accurate results, however generally more data is preferred to more accurately represent the speaker. If the test data was spoken in a fashion not covered by the training data, then there is a lower chance of identifying the correct speaker. The figure below shows one such example where 30 seconds of model data produced higher likelihood values than 120 seconds of model data.

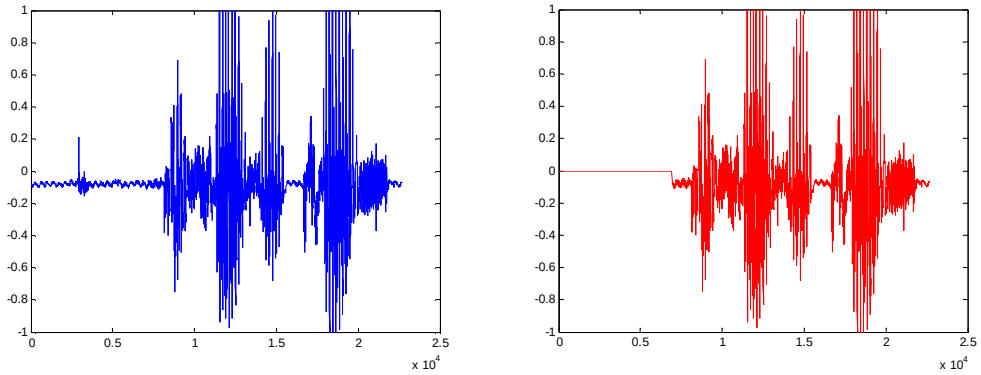


Note for Future Projects

The majority of current systems now employ the use of a universal background model or UBM, which is a single model trained by a handful of people representative of the population, in hopes of isolating speaker independent characteristics of speech [6]. Results have proved that UBM provide a more accurate identification at the cost of greater computation during the decision phase (one more model to calculate likelihood for and a more involved way of determining the winner). Future projects should consider the use of a UBM when creating a text-independent speaker identification system. It is also possible to construct a naive open-set ID system by using setting a likelihood ratio threshold between the likelihood of a speaker model, and the likelihood of a background model, for speaker verification purposes.

Results for VAD

Voice Activity Detection is implemented on DSK and MATLAB is used to visually graph the result. Below is one of the results that are successfully detecting the speech.

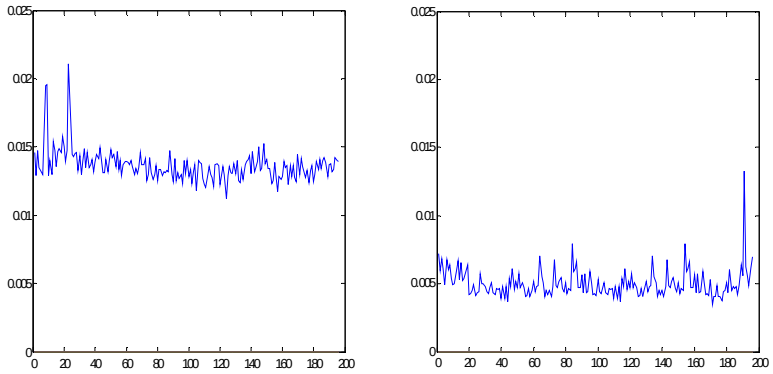


As can be seen above, it successfully removes the clicking noise and the non-speech region. Since we use average standard deviation of noise in the place we are interested in and determine the threshold based on that data, our VAD algorithm is able to erase the noise that will always present in that place, such as fanning noise. In addition, by using a buffer of 30 blocks after the standard deviation threshold, the algorithm can get rid of silence by differentiating speech and silence and, at the same time, can keep the short silence in the middle of the speech. Moreover, it successfully erases the clicking noise that can happen during the recording. We set it to decide the presence of speech when there are more than ten blocks that are higher than the standard deviation threshold in the buffer. Throughout the testing, the number ten turned out to be the best number that guarantees VAD not to miss any speech and minimizes the noise it may contain. Since we are more interested in speech and do not want to loose any of the speech data, our VAD more focuses on having all the speech even though it may contain very small

noise. However, it still very well erases silence and never had any clicking noise throughout the test. Although it may seem very simple, our algorithm is very powerful, and that is the main reason that we implemented this algorithm.

6. What worked and what did not

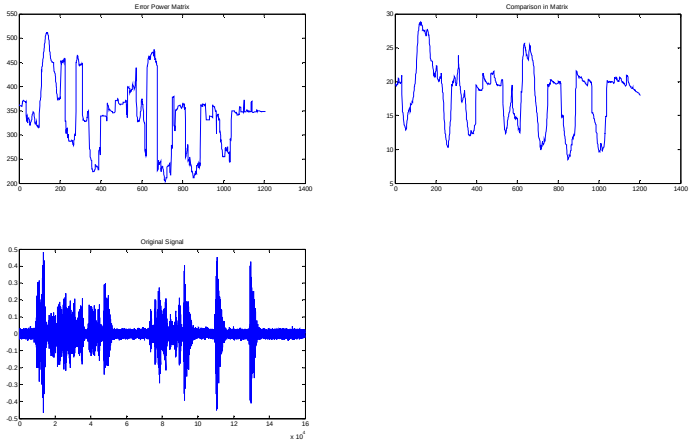
The system that we developed proved to be relatively robust for speakers that were trained in our set of data and for keywords that were spoken clearly in sentences. However, our system did face some errors and problems, mostly when situations were not ideal. Originally when we were testing our data, we used an omni-directional webcam in order to record audio signals, which was not ideal. When we finally received our headset that was suggested by the staff of 18-551, we tested our headset against the webcam and discovered that the headset had a much lower standard deviation for noise than the webcam. Below is a picture of the two standard deviations over time for a "speechless" environment.



The headset on the right has a standard deviation three times lower than the webcam. Also, the headset is a directional microphone pointed directly at the mouth and proves to show much better results in quality of recording the input speech. Another issue that we ran into during our testing was the variability of the noise in the PC sound cards. Different computers had different ambient noise levels and this changed the standard deviation threshold that was used by our code for voice activity detection. Once the standard deviation was recalculated, the VAD algorithm would work again immediately.

One of the errors with the keyword spotting algorithm was because the recorded audio was continuous speech, the words of the speaker naturally ran together and blended, not allowing for the algorithm to easily discern the pronounced beginning and ending of words. This led to errors in our results and peaks in places that would not normally have had detections had the individual pronounced each word clearly. This is an inevitable error that

could only be resolved with a variation on phoneme detection and word separation algorithms that could reconstruct words. Another obvious downfall of our approach is the use of error power as our detection method because that leads to many different words that sound like the specified keyword approaching the same levels of detection. There is no weighting on the different sections of the word so words that have the same endings as the keywords and have the same length would match the word in the parts that they have the same ending. In our current system we do not counteract these weak spots of the algorithm, but are still able to produce good results for unknown speakers. An avoidable error that we originally encountered was the attempt at recognizing unknown speakers while only having the keyword trained for one user.



In the upper left figure, the three lowest peaks on the right of the data were all occurrences of the word bombing and the peaks to the left were not any of the keyword. This sentence was spoken very quickly by an unknown speaker and there were errors involved. However this was expected and remedied by training multiple users per keyword.

Another problem we encountered was in the manner of speaking for the SID algorithm. If an individual was trained speaking with a specific demeanor and changed this manner of speaking in the analysis phase than it is unlikely that this person would be able to be recognized. This is the limitation of the GMM algorithm as it approximates a model for the given training data and does not allow for adaptations in the speech patterns of the individual. Limitations such as this exist for all SID algorithms and are difficult to be remedied in a non-text based speaker independent system; however, since we are assuming that the individual will originally speak normally, we will eventually be able to correctly identify the speaker.

The current SID system is a closed-set system which means that the algorithm will always identify one of the targets in the database. This is a known limitation and creating an open-set system requires further analysis in speaker verification which demands more resources than we currently have. Also the current system only detects one speaker at a time.

Conversation between two people will not work for the current system. A multiple-speaker detection system would require further processing to determine when the current speaker stops talking and another speaker has begun. Another variable that would severely decrease the probability of a true identification would be increasing the number of models in our set. Not only will this require more processing power, but it would decrease the chance of a correct match. For a small set of three models, the probability of identifying a correct speaker is much more likely than a set of ten speakers.

A note on the use of different FFT's is the choice of a specific radix to be used for the size of the FFT. We originally attempted to use the radix-4 FFT that was used in lab because of its superior speed to the radix-2 FFT; however we optimized all of our code to perform 512 length FFT's and for some reason the FFT did not return the correct results. This is because the radix-4 FFT's require a length that is a power of 4 which 512 is most certainly not. We had overlooked that fact and it should be noted by future groups in choosing the FFT that the length of the FFT is another determining factor of which radix FFT you ultimately decide to use. The cycle time for all of the FFT's we considered is included in our Appendix A.

7. Processing Speed and Data Rates

Our original intent was to have the entire analysis data flow exist on the DSK and run in real time. However, getting into the project we realized that the difficulty of integrating the different parts together and maintaining correct results in addition to the limited amount of processing power on the DSK would not allow a real time simulation of our code. Also, we were having troubles with our microphone and the MIC IN line of the DSK getting the power boost to work and amplify the signal enough to be analyzed. Even with the changes in goals from the outset of the project, given some sacrifices we were able to get the code working on the DSK and matching the results of the PC-side C code and Matlab results. Another original goal was the have multiple keywords analyzed at the same time, however with processor limitations it was not possible to analyze this many in real time and only one keyword at a time was implemented.

We had started at the update with a sampling rate of 32 kHz but were reminded that this was an unnecessarily high rate because speech characteristics do not exist at the Nyquist frequency of 16 kHz. We lowered our sampling rate to 16 kHz which was an optimization in it reduced the amount of data we had to analyze for each block of data. Another plan we had implemented in the outset was recalculating the entire STFT each time a new block was received. The re-calculation was unnecessary because data farther than the number of FFT samples taken behind the current block cannot affect the STFT of the current block and therefore do not need to be

recalculated. Therefore, we optimized our original code by implementing the circular buffer for the STFT described in the DTW description section. The block length was set to 384 samples in order to make it a multiple of three times the FFT step size of 128 so that the 512 length FFT would fit perfectly three times in two blocks. This allowed no data to be missed when calculating the FFT and reduced the total number of FFT's to be taken while remaining a longer length, 512, FFT to keep the frequency characteristics of the data intact.

The block length of 384 at a sampling rate of 16 kHz also produced a window size of 24ms which was the recommended window size that was reported in many of the readings and Professor Sullivan in class as an accepted standard. An overlap of each window is also considered standard; however we accommodate the overlap in our FFT instead of in the raw data which allows us more time to process the current block of raw data. The number of Mel coefficients that we chose was 13 because it was the default value used in the Matlab scripts and it provided accurate results that are shown in the results section.

The voice activity detection algorithm that we selected proved to be extremely robust while relatively simple to implement. The total time that the code takes to execute, ~1 million cycles, is minimal compared to the processing of the other algorithms. The number of blocks out of 30 that were decided upon to determine speech was decided on a trial and error basis and the final decisions that we made proved to be both accurate and robust. A buffer of data of ones and zeros of length thirty also proved to be extremely small compared to the size of the data stored by the other functions. Overall the VAD algorithm was not processor or memory intensive.

It takes ~9,398,016 cycles just to calculate the log-likelihood for one 24ms window's worth of coefficients for just ONE speaker model. We have three models to test against and just testing one model, brings us above the threshold for running a real time system. Loop unrolling does not seem to affect efficiency. As a side note, double precision was used in order to get the same accuracy as MATLAB on the DSK. Using single precision would yield zero in some cases and completely screw up results, however this is only the case on the DSK. Using GCC on the PC, single precision was perfectly fine in representing small enough numbers.

Optimizations in calculations have been accomplished by pre-computing values not dependent on the input data x, (i.e. the coefficients in front of the exponential in each likelihood calculation) and storing them as updated weights. This reduces the number of calculations slightly by (Number of Coefficients) X (Number of Gaussians) + 2 = 210 additions, one subtraction, three multiplications, one division and 3 log operations per log-likelihood calculated. However this an insignificant improvement compared to the number of total cycles needed for one calculation of a log-likelihood.

$$p(\vec{x}|\lambda) = \sum_{i=1}^{M} w_i p_i(\vec{x}). \qquad p_i(\vec{x}) = \frac{1}{(2\pi)^{D/2} |\Sigma_i|^{1/2}} e^{-(1/2)(\vec{x}-\vec{\mu}_i)'\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)}$$

The dotted boxes show which part of the calculations can be pre-computed and combined independently from test input, x, before hand.

Because of the struggles to get all of the code working together we were unable to spend much time optimizing the code after we were able to get it working. In attempting to profile the non-optimized code we converted all of the code in the DTW algorithm into separate functions that could be separately profiled. The total number of cycles for each function of the Dynamic Time Warping algorithm is shown below in the figure. The total number of cycles to maintain real time was approximately 5,454,545 cycles which was obviously not met by the total number of cycles of the current program. The DTW took a total of 218,781,802 cycles and even with our suggested optimizations we do not believe that we could have met the real time goal. Surprisingly to us the actual dynamic programming calculations that were performed on the DSK took very few cycles of the total time for calculation. After analyzing the amount of processing taking place in the dynamic programming routine, there are not a lot of calculations and there is only one two dimensional matrix that is traversed. However, unlike the dynamic programming routine, the similarity matrix calculations take most of the time on the DSK. In retrospect this makes sense because of the large number of matrix manipulations and the matrix multiplications that need to be performed to set up the dynamic programming matrix. The dual access to external memory for each matrix multiplication that is performed multiple times accumulates to make the total number of cycles extremely large in comparison to the other procedures.

We initially anticipated the STFT of the entire word with each new block to take the most time in our cycles and the calculations still allowed us to perform the results in real time. However, we realized that only three FFTs needed to be performed each block and upon coding that the DTW algorithm was the heart of the processing time on the DSK. Also, this processing time did not allow us to continue with our real time approach. Some optimizations that could be made would be to reduce the two dimensional references to the external memory matrices to one dimensional accesses to avoid mallocing two dimensional arrays. We believe that this could reduce the amount of time spent in accessing external memory matrices by half of the current amount. Another optimization would be moving the frequency domain of the STFT to the rows of the matrices to increase the possibility of a cache hit due to the large amount of accesses along the row dimension. Obviously, as in lab 3, paging the data from external memory to internal memory to perform the calculations would decrease greatly the number of cycles to perform each calculation. As seen in lab 3 this can reduce

the amount of calculations by almost a factor of 10 assuming that the number of actual calculations is small inside the loops. This would be a task that could be completed by a future group or our group given more time to complete the project. Overall, external memory accesses and the large number of matrix calculations needed for the DTW algorithm were what took the most amount of time and would need to be reduced in order to get the code working on the DSK in real time.

For the implementation that we were able to get working on the DSK we transferred the entire STFT for the signal and the stored sample each block time. This transfer totaled to 86,352 bytes for the input signal and the same number of bytes for the reference signal. By timing on the PC the amount of time to transfer this amount of data we were able to measure the transfer speed of the small amount of data that we were sending. The average time to send the entire amount of data was 101.5 microseconds which transferred to 830.8 kilobytes/second transfer from the PC to the DSK. This data transfer rate was much smaller than that measure in the labs and we believe that this reason is similar to the reason described by Group 6 of this year in their presentation that the chunks of data are sent only once and are rather small. This means that they suffer from the overhead that must be established before they are able to transfer and are not able to reach the peak transfer rate before all of the data has been sent.

Amount of memory used per algorithm:

Algorithm	Memory Size (Calculation)	Memory Size (Overhead)
DTW	425,333	10,920
VAD	0	120
SID	3,888	4,672

Amount of cycles per DTW algorithm:

DTW Algorithm Section	Number of Cycles
Compute Magnitude	5,607,810
Compute Sim Matrix	156,272,570
Dynamic Programming Calc.	46,998
Phase Vocoder	26,235,003
Mel Coefficients	30,619,421
Total Number Cycles	218,781,802

Amount of cycles per VAD algorithm:

Total Number of Cycles: 1,568,908

Amount of cycles for the SID algorithm:

Total Number of Cycles for One Speaker Model (there are 3): 9,398,016

Appendix A. FFT cycle timings

FFT Name	Radix	Length of FFT	SP or DP	Average or Single Execution	Number of Executions	Incl. Total	Incl. Max	Incl. Min	Incl. Average	Theoretical
DSPF-sp-cfftr2_dit	2	512	SP	Average	50	538994	11117	10773	10779	9258
DSPF-sp-cfftr4_dif	2	512	SP	Average	50	623672	12838	12466	12473	8187.5
DSPF-sp-cfftr2_dit	4	256	SP	Average	50	209593	4626	4183	4191	4138
DSPF-sp-cfftr4_dif	4	256	SP	Average	50	188151	4254	3753	3763	3696

9. Final Work Schedule

Tasks completed by person:

- Matt Keagle
  - Researched and implemented the DTW algorithm in Matlab and C Code.
  - Implemented the melfcc code in Matlab and C to categorize the speakers.
  - Designed the GUI interface for training and analysis of keywords.
  - Attempted to combine in real time all of the algorithms using the MIC input and DSK -> PC output.
  - Developed the DTW DSK simulator in the GUI for the worst case scenario of not getting the code to work due to massive bugs.
- Andrew Lam
  - Performed the extensive research to decode the Matlab GMM data.
  - Implemented the log-likelihood analysis in C and worked together to combine the code on the DSK.
  - Integrated the log-likelihood analysis into the DSK simulator in Matlab.

    - Worked very hard on the final paper early thinking it was due today, making it easier for us later.
- Han Joo Chae
    - Performed the mathematical calculation and analysis of the standard deviation Voice Activity Detection algorithm
    - Implemented the Voice Activity Detection algorithm in Matlab and C and made the GUI output look much more sophisticated.
    - Worked to integrate all of the elements of the GUI together in a seamless manner to allow it to be visually appealing and readily understandable.
- All
    - Worked as hard as possible and put in the maximum amount of effort to get the project done

10. References

A note on what we did and what had been done:

The Matlab code for the DTW algorithm had already been implemented [1] and used in order to warp time samples. However this code was not expanded for use in comparison and all of the comparison was implemented by our group. Using the Mel coefficients instead of the lowest cost path resulted because the Mel coefficients are averaged over each new recording and the STFT is not. Using the guidance of [3] and the advice of Professor Stern we were able to implement the DTW sliding model method. GMM code was readily available on the web [5] and is considered the front-runner in text-independent speaker identification. This code along with [2] help in understanding the algorithm we were able to simply implement the C code for the log likelihood classifier and Matlab training data. Many groups have used standard deviation to get energy level of the signal in image processing; however our method was totally new in VAD, and the code and algorithm was implemented completely on our end without the knowledge from other sources.

[1] D. Ellis (2003). Dynamic Time Warp (DTW) in Matlab
    - Web resource, available: http://www.ee.columbia.edu/~dpwe/resources/matlab/dtw/.
    - Provided all of the Melfcc and DTW code in Matlab

[2] PEREZ-MEANA, HECTOR. Advances in Audio and Speech Signal Processing : Technologies and Applications. Hershey, PA: Idea Group Pub. 2007.
    - Overview speech process (373)
    - GMM (398)

[3] RAMACHANDRAN, RAVI. Modern Methods of Speech Processing. Boston: Kluwer Academic Publishers, 1995.
    - Word Spotting (123)
    - HMM Parameters (193)
    - Word Modeling (195)
    - Speaker Recognition (299)

[4] KLEVANS, RICHARD. Voice Recognition. Boston: Artech House, 1997.
    - Summary of techniques (15)

- Tested Results (107)

[5] Alexander, Anil. "Automatic Speaker Recognition: A Simple Demonstration using Matlab for the Biometrics course, Communication Systems, EPFL, 2004". Updated 17-November-2007.

- GMM MATLAB example code
- http://scgwww.epfl.ch/courses/Biometrics-Lectures-2005-2006-pdf/03-Biometrics-Exercise-3-2005/

[6] EURASIP Journal on Applied Signal Processing 2004:4, 430-451.
A Tutorial on Text-Independent Speaker Verification

- This link was especially helpful in explaining the basic overview of the speaker identification process. It provides a useful explanation of MFCC and GMM and how they are usually implemented in an Automatic Speaker Recognition system.

[7] Cumulative distribution function. *Normal distribution*. Wikipedia. November 3, 2007. November 13, 2007. <http://en.wikipedia.org/wiki/Normal_distribution/>

- This link was helpful to get a basic idea of how the Probit function works to calculate the standard deviation steps from zero in VAD.