I’m Not Drunk, I’m Just Exhausted From Drinking All Night

Group 4
Fall 2006

Rob Vargo (rsv@andrew)
Christina Hallock (challock@andrew)
Joe Laws (jlaws@andrew)
Chris Hoffman (cehoffman@gmail)

Table of Contents

Table of Figures

Appendices

1. Introduction

Driving Under the Influence (DUI) is a major problem in the United States. For those repeatedly convicted of this felony, the court requires that he or she must purchase a Breath Alcohol Ignition Interlock Device. This device requires you to pass a breathalyzer exam before ignition of your vehicle. If your Blood Alcohol Content (BAC) registers above the limit, the starter on your vehicle would be locked out, and you could not start your vehicle. However, a person can circumvent this situation by having a sober person take the breathalyzer test. A more intelligent system is needed to overcome this instability.

Our solution is to create a "drunken speech classifier". This classifier will not only have the ability to determine whether speech is drunk or sober, but it will also verify that the speaker is who he or she claims to be. It is important to note that our project will focus on the drunk/sober classification. Future work can be done to complete the voice verification and create the entire system.

Are people really interested in this product? Currently, there are attempted solutions to overcome the instability in the interlock system. The interlock system has the "rolling or running retest", in which the driver must perform arbitrary tasks (such as humming, or sucking in air) while the vehicle is in motion. According to the US National Highway Traffic Safety Administration (NHTSA) guidelines, there is only one subsequent test required for those with the interlock system (whereas the Canadian standard calls for multiple running retests)[1,2]. Not only is this dangerous to have people drive and take non-sense tests, but it clearly does not prevent the intoxicated from getting behind the wheel.

Our solution, the "Drunkalyzer", will give accurate, instantaneous results and ensure the safety of all drivers on the road. The system consists of a database of both drunk and sober speech. The person under test will have to read from a screen a random phrase or sentence. The person will speak into the device, and it will classify the person as either drunk or sober. The key here is that the phrases are generated at random. That way no one can pre-record phrases to deceive the system.

This project has never been attempted in 18551 Digital Communication and Signal Processing Systems Designs. For future work that can be done to improve this system, please see the Section 20.

2. Our Algorithm

The following is a brief description of our algorithm. More details will be included in later sections.

[1] http://www.totaldui.com/ignition_interlocks.htm
[2] http://www.1800duilaws.com/article/interlock.asp

We used Mel Frequency Cepstral Coefficients (MFCC's) to characterize speech. The MFCC's tell how much energy is in different frequency bands. MFCC's are used to tell what phoneme was said. For example, the MFCC's for the sound <a> will be different than the MFCC's for the sound <e>. It was our belief that the MFCC's would also be able to tell the difference between a sober person saying <a> and a drunk person saying <a>.

We created a large database of people saying the same phrases both drunk and sober. We separated the database into four sets: drunk males, sober males, drunk females, sober females. Next, we use the LBG Algorithm to break each of these sets into 100 codebooks. We store the location of the center of each codebook, known as a centroid.

Then, we have a test subject speak into a microphone. We break the person's speech into small segments, and find the MFCC's of that segment. If the person is male, we compare each segment to the 100 male drunk centroids and the 100 male sober centroids (if the person is female, we compare to the female sets). Next, we find what the closest centroid is. If the closest centroid is from the sober set, then we say the current segment is sober; if the closest centroid is from the drunk set, we say the current segment is drunk. Finally, we calculate the total number of drunk and sober segments from the entire speech sample. If more segments were drunk, then we say the person was drunk; and if more segments were sober, then we say the person was sober.

3. Database

A significant portion of the project was creating a database. The database consists of several native English speakers and their corresponding Blood Alcohol Content (BAC). We spent several nights collecting the data under careful supervision.

We had a list of several tongue-twisters for the speakers to say (see Appendix 1: List of Phrases). First we had everyone read the list in their sober state. Then, we had our subjects drink alcohol. As they were drinking, we had them say the same phrases, and we recorded their BAC. So, we had all of our subjects saying the same phrases over a wide range of BAC.

In the end we had a total of seven female speakers and 5 male speakers. Each person said the 12 phrases at several BAC levels. Each reading (meaning the person saying the 12 tongue twisters) consisted of about 1 minute of speech, sampled at 16 kHz, mono. There are approximately 10 minutes of recordings for each group (sober male, sober female, drunk male, drunk female).

Before calculating the MFCC's of the wave files, we made necessary changes to any portions of the waveforms. We cut out any laughter, and tried to eliminate silence by using the Audacity program on the lab PC's to edit our .wav files.

We then divided out database into a training set and a test set. We used about 90% of the database for our training set. We used the training set for the LBG Algorithm (we took

them MFCC's of all of the data in the training set, then separated those into codebooks; see Section 5). We could not bring in drunk people for our demo; therefore, we had to choose some of the phrases in our database to be in our test set. We randomly selected 10% of the database to use to test our system. Our test system consisted several different males and females saying the 12 tongue twisters. We then used these samples to test our system (see the results in Section 7).

4. MFCC's

4.1 Description

The problem of classifying if a person's speech sounds drunk or sober is very similar to a speaker identification problem or a language identification problem. We have to take a speech sample, extract information (features) from it, and find the closest match of those features. There are two dominant methods of extracting the features: the filter-bank spectrum analysis model, and the linear predictive coding (LPC) spectral analysis method [8]. Chapter 3 of the book Fundamentals of Speech Recognition by Lawerence Rabiner and Biing-Huang Juang describes both of these methods in detail.

The idea of the LPC method is that the current speech sample can be approximated by a linear combination of past speech samples, such that

$$s[n] \approx a_1 s[n-1] + a_2 s[n-2] + \ldots + a_p s[n-p] \qquad [8]$$

One must also include an excitation source in the model, which we will call G*u[n]. The excitation source will be a pulse train for voiced sounds and white noise for unvoiced sounds. When including this term, the model becomes

$$s[n] = \sum (a_i s[n-i] + G*u[n]) \qquad [8]$$

By calculating the a_i terms, one can create an all-pole model of the current speech sample.

The LPC model has been shown to work well [8]. However, there are some disadvantages to the LPC model. First, LPC approximates speech linearly at all frequencies [11]. As stated in Section 4.2 MFCC Calculation, humans do not perceive speech linearly, so therefore the LPC will not accurately model human hearing. Second, the LPC model includes a lot more high frequency information than other models do [11]. High frequencies contain more noise, and this noise will negatively affect the performance of the system [11].

The second popular feature extraction method is the filter bank method. The idea of the filter bank method is to calculate the total energy in different frequency bands. A signal is passed through a series of bandpass filters. Then, the total energy from the output of each bandpass filter is calculated. The location of the bandpass filters is important. The filters can be spaced linearly, logarithmically, or non-uniformly [8]. The problem with

spacing the filters linearly or logarithmically is that human hearing is neither linear nor logarithmic. Therefore, a non-uniform filter bank will give the most accurate representation of human hearing [8].

One popular example of a non-uniform filter bank is the Mel-Frequency scale. Human hearing is approximately linear below 1 kHz and approximately logarithmic above 1 kHz [8]. The mel-scale tries to approximate human hearing. The mel-scale frequency (mel(x)) is given by

$$\text{Mel}(x) = 2595*\log_{10}(1 + x/700) \qquad [11]$$

Mel Frequency Cepstral Coefficients (MFCC's) can be calculated using a mel-scale based filter bank. The full calculation is described in Section 4.2. The MFCC's give the total energy in different frequency bands based on the mel-scale. This gives an accurate estimation of human hearing. MFCC's can capture phonetically important characteristics of speech [11].

MFCC's are very popular in many speech applications. MFCC's have been the popular choice in language identification problems [11][1]. They have also been used in accent detection [12]. We consulted with Professor Tanja Schultz from the Language Technology Institute when we were developing our algorithm, and she suggested using MFCC's for our application. We then consulted with Professor Richard Stern, and he agreed with Professor Schulz's recommendation. Because of the popularity of MFCC's, the fact that MFCC's model human hearing much better than the LPC's, and the recommendations of very experienced researchers, we decided to use MFCC's as our feature set.

In searching online, we found could not find much code available to calculate MFCC's. Eventually, we found MFCC code available from the European Telecommunications Standards Institute (ETSI). The code is available free for download from www.etsi.org , along with a very good reference document [3].

The code from ETSI needed to be changed in order to work on the DSK. It was originally designed to read from an input file, then write the MFCC's to an output file. We had to change this to read from either the codec or from a .wav file. The code originally stopped whenever it reached the end of its input file. We had to change this to fit our setup. We had a counter keep track of the current location in the input speech buffer we were operating on. Whenever this counter went beyond the length of the buffer, the code returned 0 and ended. Also, we had to change it so that the MFCC's would be stored in an array, and sent back to the PC.

Their code uses the following algorithm to calculate the MFCC's.

4.2 MFCC Calculation

The algorithm used by ETSI was standard across many sources we found [3][6] [9]. A block diagram is shown in Figure 1 [9]. The following section describes the algorithm (details taken from document that accompanies the code from ETSI [3]).

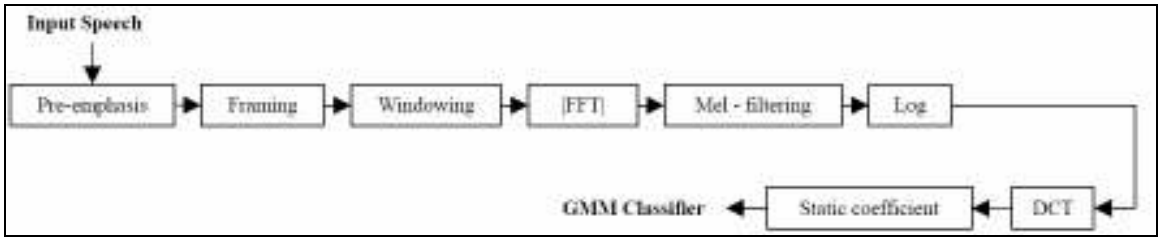

Figure 1: Block Diagram of MFCC Calculation

First, the signal needs to be sampled. This is done in one of two ways in our project: sampling by the codec or sending .wav file from the PC to DSK. Next, the signal is sent through a notch filter that removes the DC offset. In speech, the DC term is not important. But, if it were to be included, it could possibly skew our data. Therefore, it is removed by used the following notch filter:

$$s_{of}[n] = s_{in}[n] - s_{in}[n-1] + 0.999 * s_{of}[n-1]$$

where $s_{of}[n]$ *is the offset-free input signal, and* s_{in} is the original input speech.

The signal is then divided into smaller segments. We decided to use a 25 ms segment of speech. This is a standard segment size, because speech is approximately stable over a 25 ms period [3]. Each frame will overlap the next frame by 10 ms (see Figure 2). ETSI uses 10 ms overlap standard in their code, so we decided to keep it. Because we are using a 16 kHz sampling rate, each frame will be 400 samples, and will overlap 160 samples. We calculate the MFCC's on one frame, then move on and calculate the MFCC's on the next, and continue until all frames have been used.

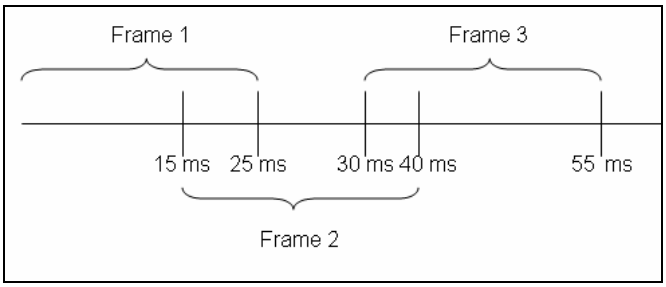

Figure 2: Framing of input data

Next, pre-emphasis is done on the frame of speech. This is done in order to make the signal less affected by finite precision effects [8]. In order to do this, we use a simple low-pass filter:

$$s_{pe}[n] = s_{of}[n] - 0.97 * s_{of}[n-1]$$

where $s_{pe}[n]$ is the output of the pre-emphasis filter.

The 25 ms frame is then passed through a Hamming Window. This is done to reduce discontinuities and the beginning and end of a segment [3]. The Hamming Window is defined by the function:

$$s_w[n] = \{0.54 - 0.46*\cos[2\pi(n-1)/(N-1)]\}* s_{pe}[n],\ 1 \le n \le N$$

where $s_w[n]$ is the output of the window, and N is the frame length (400).

This signal is now FFT'd. The current 400 sample segment is zero padded to a length of 512. A 512 point FFT is then taken, using an FFT algorithm that was included with the ETSI software.

The result of the FFT is then transformed from a regular frequency scale to the Mel Frequency Scale. The Mel Frequency Scale was developed empirically to attempt to model human hearing. It was found that human hearing is approximately linear below 1 kHz, and approximately logarithmic above 1 kHz [9].

To do this, the signal is multiplied by a series of triangular weighting functions, as shown in Figure 3 [8]. The ETSI software creates these filters. The filters are equally spaced on the mel-scale, using the formula $\mathbf{Mel(x) = 2595*\log_{10}(1 + x/700)}$ [11], where Mel(x) is the Mel Frequency and x is the regular frequency in Hz. 23 filters are used.

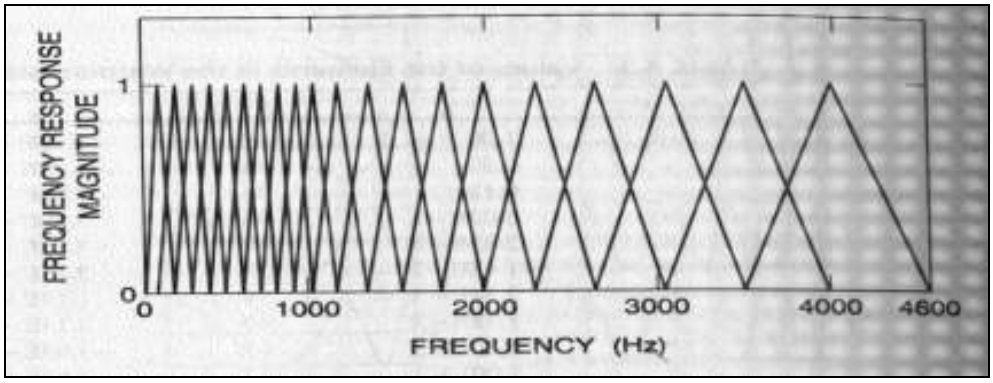

Figure 3: Triangular Weighting Filters used in MFCC Calculation
(NOTE: Our filters extend to 8 kHz, which is not shown in this picture)

The FFT is multiplied by each of the 23 triangular filters individually. This gives the energy for each frequency in that band. Then, the energy for each frequency in a band is added together. This gives the total energy in each band. The result is 23 numbers, each representing the total energy for one of the triangular filters. Let $fbank_i$ represent the total energy in the i^{th} frequency band, where $1 \le i \le 23$.

Next, the natural log is taken of the 23 sums, so that $\mathbf{f_i = ln(fbank_i), i = 1…23}$. Finally, a Non-linear transformation is performed on $\mathbf{f_i}$. In this case, a Discrete Cosine Transform is used, using the equation

$$C_i = \sum_{i=0}^{23} \mathbf{f_i} * \cos\left[\frac{\pi}{23}\left(n - \frac{1}{2}\right) i\right] \quad i = 0\ldots12 \qquad [3]$$

C_i is the Cepstral coefficient. There are 13 total Cepstral coefficients. The ETSI code does all of this. We provide it with the input buffer, and it performs all of the above calculations to get the MFCC's.

This algorithm is standard across several sources [3][6][9]. We are taking the the FFT of some signal, then taking the DCT of that result later. By taking the DCT of and FFT output, we are converting the signal to the "Cepstral domain." [2]. This gives information about the rate of change of the different frequency bands. This process is standard in all calculations of MFCC's [3][6][9].

5. LBG Algorithm

The Linde-Buzo-Gray (LBG) algorithm is a multidimensional clustering algorithm similar to the common K-means algorithm. Figure 4 [10] below shows an example of a two dimensional clustering partition generated by the LBG algorithm. The red stars are codevectors that represent the center of the blue encoding region. All input vectors that lie within a codevectors encoding region are assigned to the respective codevector. This partitioning of codevectors is referred to as a codebook.

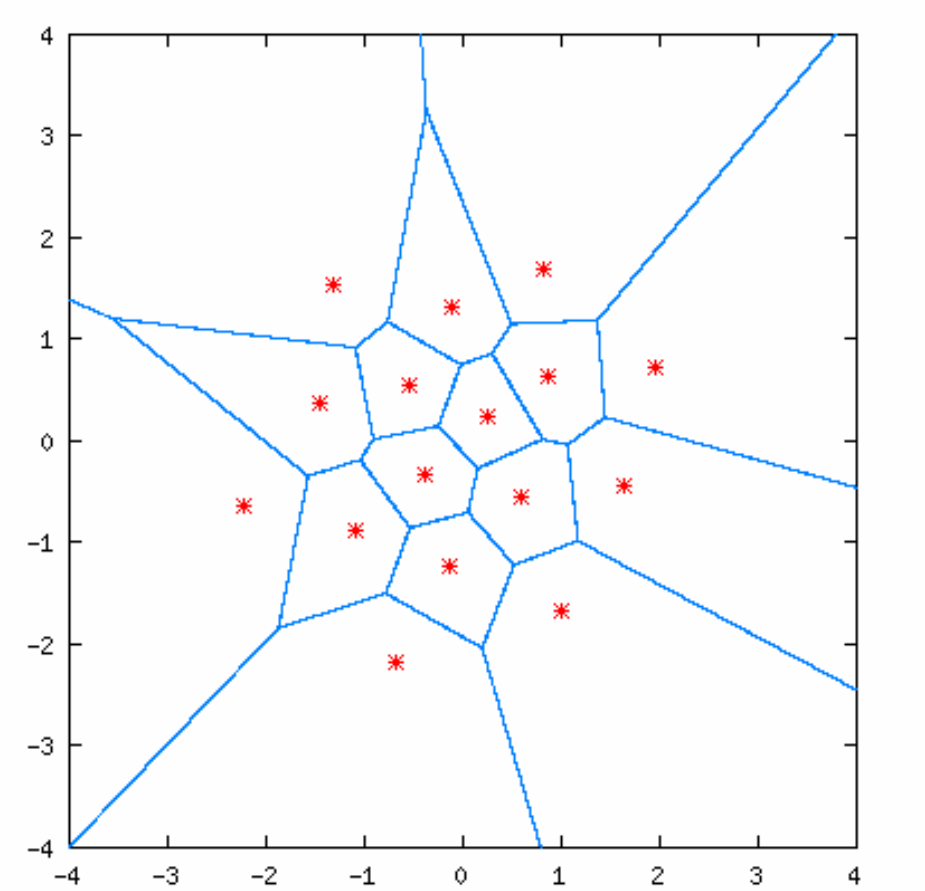

Figure 4: Two-Dimensional Codebook generated by the LBG algorithm

The algorithm starts by first generating an initial set of codevectors. Two common generation techniques are random assignment and splitting. Random assignment selects the specified number of codevectors from the input data set at random. Splitting starts with one codevector, then partitions it into two, dividing up the input data roughly equally between the two vectors. This division process is then repeated until the specified number of codevectors is reached.

We start with a training set **T** = {x1, x2,…,xM}, where M = 60,000 and **xm** is a 1-dimensional vector with thirteen MFCC values. The user specifies a parameter **N**, where N is the number of codevectors (red stars). Let **cn** denotes the nth codevector with its encoding region for denoted by **Sn**. The encoding region is determined by the Euclidian distance to the nearest codevector. Now given an input source vector **xi** we determine which region **Sn** it lies in and assign it to the codevector **cn** that was used to generate the region. In other words if the source vector **xi** is in the encoding region **Sn**, then its approximation (denoted by **Q(xm)**) is **cn**.

Once the entire assignment has been made the distortion of the entire system is calculated. The metric typically used is the mean squared-error distance measure, which is given by:

$$D_{ave} = \frac{1}{Mk} \sum_{m=1}^{M} \|\mathbf{x}_m - Q(\mathbf{x}_m)\|^2 \quad [7]$$

The next step is to generate a new set of codevectors that reduces the total distortion of the system. This is done by determining the center of all input vectors assigned to a specific codevector. The codevector is then assigned to this location, which minimizes the distortion for all the input vectors that lie within its region. After all of the codevectors are updated, we then go through the entire input data and reassign each vector to its new closest codevector. The system distortion is then recalculated. If the change is greater than the specified threshold represented by epsilon, then the codebook is accepted. Otherwise the process of updating the codevectors and reassigning the input vectors is repeated.

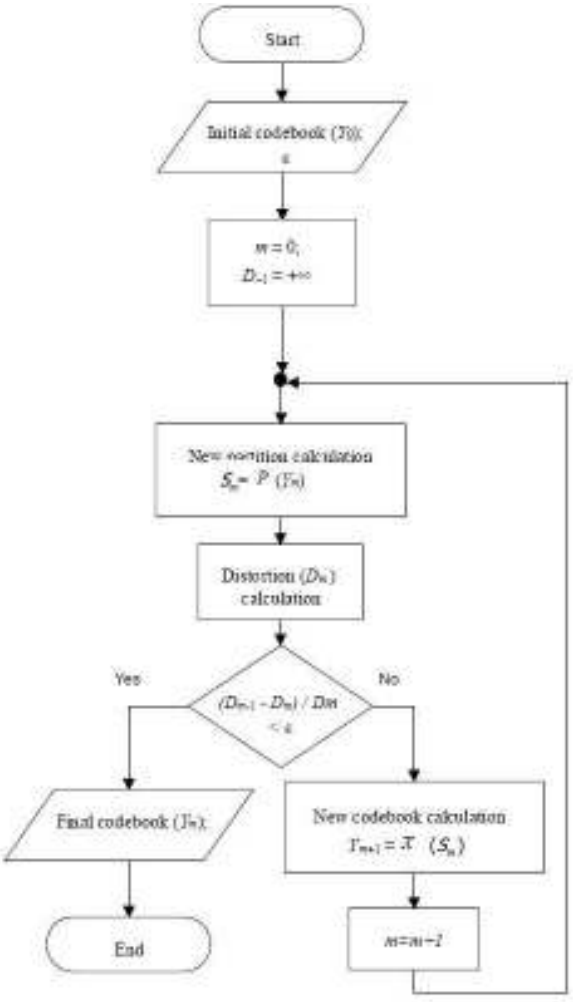

Figure 5: LBG codebook optimization [7]

Our implementation of the algorithm, written entirely by Joe in Java, takes in thirteen dimensional MFCC vectors and generates codebooks containing a specified number of codevectors. This code is run separately from the code implemented on the DSK. In our case we had the algorithm generate codebooks containing 100 and 500 codevectors for male sober (BAC <0.08), male drunk (>0.08), female sober (<0.08), and female drunk (>0.08) MFCC's. More, alternate codebooks can easily be generated by supplying the code with only the MFCC's you would like to cluster. After the codebooks are generated they are then saved to disk to be sent to the DSK to perform the classification.

The LBG algorithm typically calls for a training set from a large database. Each of our four training sets consists of about ten minutes of speech segments. Utilizing the DSK we take the ten minutes of speech and generate the sets of MFCC vectors. Since each MFCC is evaluated at 25ms segments overlapped by 10ms, the total number of MFCC vectors in the training set is: 10*60 seconds / 25 ms ≈ 30,000 vectors (each with 13 Mel Coefficients).

6. Executing the Algorithm

First, we had to create the database. Once we had the database, we separated it into four groups: male drunk, male sober, female drunk, female sober. We took each sample (sample here meaning one person saying one phrase) in the database, and calculated the MFCC's for it. We got one set of 13 values for each 25 ms segment. So, for a 10 second speech sample, there were approximately 500 sets of MFCC's (10 sec/25ms, but it will be slightly higher because of the overlap). We had about 10 minutes of speech for each group (male drunk, female drunk, etc.). Therefore, we have about 30,000 sets of MFCC's for each group.

We then used the LBG Algorithm to organize that data. As stated in Section 5, the LBG Algorithm will group the 30,000 sets of MFCC's into 100 different codebooks for each group. The center of each codebook is called the centroid. Ideally, each centroid represents the average MFCC's for a particular phoneme. Our hypothesis is that the average MFCC's for one phoneme said drunk will be different from the average MFCC's of the same phoneme said sober.

We calculate the locations of all of the centroids on the PC. This only needs to be done once. We find the locations of the centroids, then save this information to a .txt file. When we first start our program, we send the data in the .txt file from the PC to the DSK. So, the DSK has the locations of all of the centroids stored in external memory (in variables such as 'male_drunk', 'female_sober', etc.).

The DSK is then ready to classify a user as drunk or sober. The user clicks the 'Start' button on a GUI on the PC. The GUI then displays a phrase for the user to say, and the user says it into the microphone. The codec samples the speech signal, and stores this data into a buffer (we actually used 2 buffers in order to get real-time calculations; see Section 8).

We then break the buffer into 25 ms segments, and calculate the MFCC's on each segment. We now know the MFCC's for the current speech segment, and also the average MFCC's for the different phonemes for both drunk and sober people. So, we compare the current MFCC's to all of the drunk centroids and to all of the sober centroids (for either a male or a female). We find the Euclidean distance between the current MFCC's and the closest centroid in both the drunk and sober sets. If the distance to the closest drunk centroid is less than the distance to the closest sober centroid, then we say that the current segment is drunk (and vice-versa).

We keep a running total of how many segments were drunk and how many segments were sober. When the user has finished saying the phrase, he/she hits the 'Stop' button on the GUI. We do the final classification based on the total number of drunk and sober segments. If the total number of drunk segments is greater than the total number of sober segments, then we say that the person is drunk; and if the total number of sober segments is greater than the total number of drunk segments, then we say that the person is sober.

We debated using an alternate way to make our final classification. Instead of counting the number of drunk segments vs. the number of sober segments, we could have used the total distance. For each segment, we calculate the distance to the closest centroid. We then could keep a running total of the distances to the closest centroids. At the end, we would see if the total distance to the closest centroid for each 25 ms segment was greater for drunk or sober. If the total distance was less to the sober centroids, then the person was sober, and vice-versa. However, we had good accuracy using the other method, and decided not to implement this method. It may or may not have improved our accuracy some, but probably not significantly.

7. Results

We said that if between 45% and 55% of the segments were drunk/sober, then the current sample was too close to call. If a sample was too close to call, we did not use that sample in our results.

We tested our system on the following groups:

- Males in database
    - Read phrases in database and phrases not in database
- Males not in database
    - Read phrases in database and phrases not in database
- Females in database
    - Read phrases in database and phrases not in database
- Females not in database
    - Read phrases in database and phrases not in database

We had good accuracy with all of these groups. This leads us to believe that our system will work for any user, even if they have not trained their voice for the database. Also, we believe that our system will work no matter what phrase the user says. These results are described below.

We did several different experiments to test our system. First, we had a person whose voice was used in the database (Rob) speak phrases that were used in the database. There were 12 phrases in the database, and Rob spoke each of them into the microphone while sober. He spoke 3 randomly selected phrases a second time, to get a total of 15 phrases. The system correctly identified Rob as being sober 12/15 times, or an accuracy of 80%.

Next, Rob spoke 5 phrases that were not in the database (See Appendix 1). He spoke each phrase twice. The system correctly identified Rob as being sober 10/10 times, for 100% accuracy.

Originally, we thought that the system would only be able to accurately characterize a person if they said phrases that we used in making the database. However, this assumption is not correct. When we break a sound into 25 ms segments, we are basically just looking at one phoneme. Therefore, it does not matter what word was spoken. We calculate the MFCC's for each phoneme, not for each word. So, as long as we have all phonemes in our database, it does not matter what word is said. The system will be able to characterize any phrase that a user says.

Next, we have a person whose voice was not used to make the database (Joe) try our system. Joe read the same phrases as Rob read. The system correctly characterized Joe as being sober 11/15 times, for an accuracy of 73%. This is very close to the accuracy that we got when Rob spoke (12/15). Therefore, our system can be used on a user who has not trained their voice.

This is because the centroids we use for comparison represent the average of all of the males. The MFCC's for each male in the database are combined together. The LBG Algorithm breaks the MFCC's into 100 groups (each representing about 1 phoneme). It then finds the center of that group. So, the LBG Algorithm is taking the average value for all of the males for each phoneme. Therefore, it does not matter if the person speaking trained their voice in the database. As long as the user's voice is close to the average, the system will work.

We then did the same thing for females. In our system, the GUI on the PC tells the DSK if the user is male or female. Therefore, a male user is only compared to male codebooks, and a female user is only compared to female codebooks.

We had females who were in the database read phrases in the database, and then read phrases not in the database. We then had a female who was not in the database read phrases in the database and not in the database. We found that the female in the database was correctly identified as being sober 19/24 (79%). For a female not in the database, the system correctly characterized her as sober 15/20 (75%). These numbers are very close to what we found for the males, which indicates that the system works equally well for both males and females.

We now knew that the system could correctly characterize a sober person. We had to now test if it could correctly characterize a drunk person. We tested this in two ways. First, we had saved .wav files of previously recorded drunk people saying phrases. These .wav files were not used in generating the codebooks. We parsed the .wav files, and sent the raw data from the PC to the DSK. Second, we had a drunk person come in and speak into the microphone. The person was under very close supervision of the other group members while doing this, so that all people/equipment would be safe and the results would be accurate.

We had recordings (.wav files) of a female in the database saying phrases in the database. We sent these 12 phrases from the PC to the DSK, and had the DSK characterize the person. The DSK correctly characterized the person as drunk 12/12 times (100%). Unfortunately, because of the lack of girls in ECE who were willing to get drunk for this, we were not able to have a drunk girl come in and test our system in person.

We were able to have a drunk male come in and test the system. The male was in the database, and said phrases in the database and phrases not in the database. The system correctly identified him as drunk 13/17 times (76%). We also had recordings of a male saying phrases in the database stored as a .wav file. The DSK correctly identified his as drunk 11/12 times (91%).

Figure 6 contains a table of all of our results.

	Male Drunk	Male Sober	Female Drunk	Female Sober
Correct	24	42	12	34
Incorrect	5	8	0	10
% Correct	83%	84%	100%	77%

Figure 6: Table of Results

Also, we wanted to see what would happen if we input ambient noise into the system. Ideally, the system would characterize ambient noise as being neither drunk nor sober. Ambient noise should show up as 'too close to call'. We did 21 samples of this. 5 were said to be sober, 6 were said to be drunk, and 10 were too close to call. The ones that were characterized as drunk or sober were very close to our threshold (around 60% of segments were drunk or sober). We think that we set our threshold too close. We think we should have made the threshold of something being too close to call at closer to 60%. If we were to increase the threshold to about 60%, we expect to see that ambient noise will always show up as neither drunk nor sober.

Some phrases were recognized incorrectly a lot. For example, the phrase "Sally sells sea shells by the sea shore" was usually recognized as being drunk, even when a sober person said it. This is probably because the <s> sound sounds like a person slurring their speech. Other phrases with high inaccuracy were "which wristwatches are swiss wristwatches" and "lesser leather never weathered wetter weather better". These phrases both have a lot of fricative sounds (<s>, <th>, <sh>, etc.). We believe that our classifier

has trouble with fricatives. If we would have omitted these three phrases, our accuracy would have been much better.

Also, noise was a big factor in our performance. The lab environment in which we were testing was very noisy. This caused problems in our testing, and also problems in our demo. When we were recording .wav files on the PC, we had a very quiet environment. So, the database is made of a lot of very clean .wav files. When we sent a .wav file from the PC and told the DSK to classify it, it was right almost 100% of the time. So, if we had a quieter environment in which to test, our results would probably be higher.

8. DSK

Most of the work is done on the DSK. The DSK collects the input data, either by receiving a .wav file from the PC or by reading directly from the codec. If reading from the codec, we set the sampling frequency to 16 kHz. We have 2 different buffers to store the data from the codec, each of length 4000 (will store ¼ second of input data). We fill up the first buffer. Once that is full, we send that to the function that will calculate the MFCC's for those 4000 points. While the DSK is calculating the first set of MFCC's, it is using interrupts to fill the second buffer with data from the codec. When the second buffer is full, the MFCC's are calculated on that buffer, and the new input data is stored in the first buffer again. The MFCC calculation takes much less than ¼ second, so we never have an issue.

The DSK can also read a .wav file from the PC. We saved all .wav files on the PC as mono, 16 bit, 16 kHz samples. There is code on the PC to parse the .wav file and get the raw data, then send that to the DSK. The DSK mallocs the appropriate space in external memory, then saves the data from the .wav file there. It then takes the whole .wav file and calculates the MFCC's on each 25 ms segment of it.

The DSK also does the comparison to the codebooks. Whenever the program is started, the PC sends the four groups of codebooks (male drunk, male sober, female drunk, female sober) to the DSK. These are stored in external memory. For every 25 ms segment, the DSK compares the current MFCC's to each of the codebooks. It then determines what the closest codebook is, and if it is drunk or sober. Finally, the DSK makes the final classification of whether there were more drunk or sober segments.

9. PC

The PC has the GUI which allows the user to control the DSK. The GUI will be discussed more in Section 12.

Also, the PC was used to calculate the codebooks by using the LBG Algorithm. This algorithm only needed to be run once, so there was no need to put it onto the DSK.

10. Signal Flow

Figure 7 shows our signal flow diagram. Speech can be input in 2 ways: through the codec or from the PC. If it comes through the codec, the person speaks into a microphone. The microphone is connected to the 'Mic In' input. The codec then samples this at 16 kHz, and stores the result in internal memory.

If the speech is input from the PC, the sound must be saved on the PC as a .wav file. The .wav file is 16 kHz, 16 bits, single channel. The appropriate network connects are made, and the data is sent through the Ethernet cable to the DSK. The PC first sends the length of the file to the DSK. The DSK then mallocs the appropriate space into external memory. The PC then sends the .wav file over. The DSK reads the file using the rcvData() function given to us in Lab 3.

The codebooks are stored as .txt files on the PC. Those are also sent through the Ethernet cable to the DSK. The DSK stores those in external memory as well.

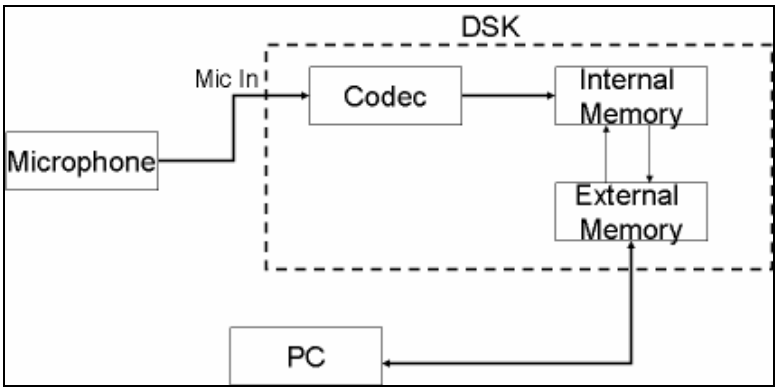

Figure 7: Signal Flow Diagram

11. Data Rates and Memory Management

We sampled all sounds at 16 kHz. Initially, we wanted to store the entire input sound wave on internal memory. Then, once the user was done speaking, we would calculate the MFCC's. However, we found that we could only store 1 second of input data in internal memory. Our code was 233 kB, all stored in internal memory. We also had several other variables that we were storing in internal memory, so we only had enough space to store 1-2 seconds of input data at one time.

To solve this, we decided to calculate MFCC's at the same time that we were reading in new data. We had 2 buffers, each of length 4000. As described in Section 8, we fill one buffer, then calculate the MFCC's on that buffer. As we're calculating those MFCC's, we're reading data from the codec into the second buffer. Once the second buffer is full, we calculate the MFCC's on that buffer, then start filling the first again. We calculate the MFCC's for each 25 ms segment. Once we calculate an MFCC for a 25 ms segment, we find if its closest codebook is drunk or sober. Then we increase the total number of drunk or sober segments by 1. This way, the user can speak for as long as necessary and we can

give them results essentially in real time. As soon as the user hits 'Stop', they have their result.

Our biggest speed issue was when we were comparing the current 25 ms segment's MFCC's to all of the codebooks. For each 25 ms segment, we have to compare 13 elements in a vector to 200 other 13 element vectors. This requires us to find the Euclidean distance, which means we have to take (x1+x2)^2 13 times. Originally, we were using the pow() function to do that operation.

When running our code, we noticed that it was slow and figured that the problem was in the comparison function. So, we did Profile Function on the comparison function of our code. The profile results were:

Include: 4,345,007 Exclude: 200,848

The only function that our comparison function called was pow(). Therefore, we figured the problem was with pow(). Instead of calling pow(), we decided to just do the actual multiplication ([x1*x2]*[x1*x2]). We profiled the code again, and the results were:

Include: 125,028 Exculde: 125,028

This was a massive improvement in speed. By not using pow(), the comparison function now takes 2% of the time that it used to. Before, it used to take 5-10 seconds to determine if a person was drunk or sober. Now, it is essentially instantaneous. As soon as the user hits the stop button, the result is displayed.

12. GUI

We wrote a GUI to allow the user to control the DSK board. Figure 8 shows a screenshot of our GUI.

Figure 8: Screenshot of GUI

The GUI will allow the user to choose how the sound will be input, either through the microphone or by sending a .wav file from the PC. If the person chooses to send a .wav file, the GUI will open a dialog box that allows the user to choose which .wav file to use. Then, the GUI will send the file to the DSK. All necessary networking requirements are done in the code for the GUI. The DSK then calculates MFCC's for the .wav file.

If GUI also enables the user to speak into a microphone. The user will hit the 'Start Recording' button. The GUI then sends a command to the DSK that tells the DSK to enable interrupts. The DSK then records from the codec, and calculates all MFCC's.

The DSK then classifies the speech as drunk, sober, or too close to call. The DSK sends this result back to the GUI, and the GUI will display the result for the user.

Also, the GUI conveniently provides new drink recipes for the user.

13. Contributions

Below is what each group member did on this project.

Rob: Worked on code for MFCC's, integrated MFCC code onto DSK, reading data from codec and storing into 2 buffers, integrating DSK with GUI

Christina: In charge of database, researched algorithm, researched MFCC's vs. LPC, data collection/algorithm verification

Chris: Wrote entire GUI, worked on integrating GUI with DSK, in charge of all networking (sending large .wav files from PC to DSK, etc.)

Joe: Wrote code to generate codebooks, researched different algorithms for codebooks (decided on LBG), wrote code to compare each MFCC vector to all codebooks, helped debug code

Our final schedule was as follows:

- October 31: Finalize algorithm details (decided to use MFCC's)
- November 10: Had code for MFCC's working on DSK
- November 21: Had all communication working between PC and DSK; able to send .wav files from PC to DSK, calculate MFCC's, and send MFCC's back to PC; PC stores MFCC's in a .txt file
- November 25: Codebooks generated on PC
- December 1: Able to read from codec
- December 2: Store data from codec into the dual buffer; able to calculate MFCC's in real time; testing of full system
- December 3: Finalize testing of system; compile all results listed in presentation/report

14. Improvements

This was the first time that a project like this was attempted, so there are some things that we realized we should have done differently. First, we thought that having the people say tongue twisters would give us the biggest difference between drunk and sober. So, when developing the database, we had people recite tongue twisters.

However, this was not the best choice. First, the sober people had difficulties saying the tongue twisters. Also, we realized that the database should include all phonemes. We should have had the people say "broadcast phrases" (phrases that include all phonemes). The tongue twisters did a bad job characterizing all phonemes.

Also, as stated earlier, some of the tongue twisters always came up being drunk. Phrases like "Sally sells sea shells by the sea shore" always are characterized as drunk. This is probably because the <s> sound is just a lot of high frequency noise. If a person is saying a lot of <s> sounds, then they sound like they are drunk. This is another reason that using tongue twisters is bad. We should have used just normal phrases.

Future work could also be done to create the entire system. Ideally, the system would have a speaker identification system to make sure that the person speaking was the correct person. It would then determine based on their speech if the person was drunk or sober. Future projects could concentrate on combining these two aspects into an entire system.

Also, we tried to design a user-independent system. We put all of the speech samples from all of the users into one large database (one for males, one for females). This means that we train on system on the average of many different people. We could have designed a user-dependent system. This would have improved our results. We could have had one person say the phrases many times. Then, we would have used the LBG Algorithm to develop 100 codebooks for that one person. By using only one person, it could have improved our numbers.

15. References

[1] Abel Vilcca Roque, Velik Bellimin, William Y. Liu. "Group #5: Language Identification" 18-551 Group #5 2002

[2] "Cepstrum." Wikipedia. < http://en.wikipedia.org/wiki/Cepstrum>.

[3] "ETSI ES 201 108." European Telecommunications Standards Institute. <http://webapp.etsi.org/exchangefolder/es_201108v010103p.pdf>. 2003.

[4] Guiwen Ou, Dengfeng Ke. "Text-Independent Speaker Verification Based on Relation of MFCC Componenets." IEEE, March 2006.

[5] Judy, Scott. "11-751 Project Final Report: Analysis of Drunken Speech".

[6] "Mel Frequency Cepstral Coefficient." Wikipedia. < http://en.wikipedia.org/wiki/Mel_frequency_cepstral_coefficient>.

[7] Patane, Giuseppe; Russo, Marco. "The Enhanced LBG Algorithm." *Institute of Computer Science and Telecommunications, University of Catania.*

[8] Rabiner, Lawerence; Juang, Biing-Hwang. *Fundamentals of Speech Recognition*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[9] Shah, Jashmin K. "Robust Voiced/Unvoiced Classification using…Gaussian Mixture Models". *IEEE International Conference on Acoustics, Speech, & Signal Processing (ICASSP).* IEEE, May 2004.

[10] "Vector Quantization." <http://www.data-compression.com/vq.html>.

[11] Wong, Eddie and Sridha Sridharan. "Comparison of Linear Prediction Cepstrum Coefficients and Mel-Frequency Cepstrum Coefficients for Language Identification." *Proceedings of 2001 International Symposium on Intelligent Multimedia, Video and Speech Processing*. 2001.

[12] Zheng, Yanli et. Al. "Accent Detection and Speech Recognition for Shanghai-Accented Mandarin." < http://www.stanford.edu/~jurafsky/p1304.pdf>. 2004.

Below is a description on the information we found from each source.

[1] Group 5 from 2002 did a language identification project. We read over their project to see how they did that, since it is similar to ours. They also used MFCC's, and we got the idea of vector quantization from them.

[2] Told why we need to take a DCT after we already take an FFT, which was a question asked of us at our demo

[3] ETSI provided code to calculate MFCC's. You can find this code by going to www.etsi.org, then searching for ES 201 108. It is a free download. Also, they provide a document detailing the calculation of the MFCC's.

[4] Talks about a use of MFCC's

[5] Scott Judy did a research project involving speech recognition of drunk speech. This was done in 11-751, and we read it to see what previous research has been done.

[6] Discusses the process of calculating MFCC's

[7] Details the LBG Algorithm. Used this as a reference in writing the LBG Algorithm in Java.

[8] Book details different methods of feature extraction, namely LPC and filter banks. We consulted this book in deciding which method to use.

[9] Discusses MFCC's, and process involved in calculating them. Also discusses Gaussian Mixture Models, which was a technique we were considering instead of LBG Algorithm

[10] Describes Vector Quantization and codebooks

[11] Discusses differences between LPC's and MFCC's, benefits of each

[12] Uses of MFCC's

Appendix 1: List of Phrases

We used the following phrases when testing the system. (*) denotes phrases not included in our database.

unique new york
sally sells sea shells by the seashore
three free throws
black bloody bat
shy shelly says she shall sew sheets.
so they flew through a flaw in the flue
lovely lemon liniment
which wristwatches are swiss wristwatches
lesser leather never weathered wetter weather better
moose noshing much mush
old oily Ollie oils old oily autos.
Quick kiss. Quicker kiss.

Initialize the board support library*
Insert code here*
TCP server socket*
call the function*
example of digit-reversal*