Voice Recognition and Identification System
(Text Dependent Speaker Identification)

Final Report

18-551 Digital Communications and Signal Processing Systems Design

Spring 2000

Group 11
Tejaswini Hebalkar (tejaswini@cmu.edu)
Lee Hotraphinyo (leeh@andrew.cmu.edu)
Richard Tseng (srt@andrew.cmu.edu)

Problem

With the increasing use of personal electronics systems that store private information, and electronic commerce that is responsible for personal financial transactions, there is a potential danger of malicious use of systems by unwanted people. It is therefore important to develop fraud-proof identification and recognition systems to increase security in these highly susceptible areas.

Solution

In order to solve the problem of unwanted access to systems, we proposed a Text Dependent Speaker Identification System that uses two features to keep out intruders. If a user wanted to gain access, both the user's voice and his/her personal password will need to be recognized and authenticated by the security system. Even if an unauthorized user could obtain a sample of an authorized user's voice, access to the system will not be granted without the proper password. These two features build redundancy into the security system, making it difficult for intruders to get it.

Our voice based security system utilizes speech recognition algorithms such as Feature Extraction, Vector Quantization (VQ), and Dynamic Time Warping (DTW) to recognize the speaker and password. The voice signal is sampled at 11.025 kHz with 16-bit precision.

Prior Work

The only 18-551 project that has dealt with voice-based security was "*Text-Independent Speaker Identification"* from Spring 1998. Their project dealt with one aspect of our system i.e. "speaker identification", irrespective of the test spoken. If an unauthorized person was able to obtain a copy of the user's voice, that system would be defeated and access would be granted to the unauthorized user. An example of how this system could be defeated can be found in the movie *Sneakers.* In the movie a person was able to record the voice of the user and play it back to the security system to gain access to a private area of a building.

Other 18-551 projects that dealt with voice recognition were related to identifying certain words to controls household appliances or to command movements of a robot. Thus these projects dealt with the second part of our project i.e. "text identification", irrespective of the speaker. Any person could say these command words to control the system. If unauthorized people knew the commands, they would have access and be able to control anything in the system.

Feature Extraction

Before identifying any voices or training a person to be identified by the system, the voice signal must be processed to extract important characteristics of speech. By using only the important speech characteristics, the amount of data used for comparisons is greatly reduced and thus, less computation and less time is needed for comparisons. The steps used in feature extraction are: pre-emphasis, frame blocking, windowing, autocorrelation analysis, LPC analysis, Cepstral analysis, and parameter weighting. [1]

Pre-Emphasis

In this step, the signal is passed through a low-order FIR filter to spectrally flatten the signal and to make it less susceptible to finite precision effects. The transfer function of this filter is:

$$H(z) = 1 - az^{-1}$$ ……………………………(Eq. 1)

The value for $\boldsymbol{a}$ usually ranges from 0.9 to 1.0. For our system, we chose $\boldsymbol{a}$ = 0.9375.

Frame Blocking and Windowing

The signal is then put into frames (each 256 samples long). This corresponds to about 23 ms of sound per frame. Each frame is then put through a Hamming window. Windowing is used to minimize the discontinuities at the beginning and end of each frame. The Hamming window has the form:

$$W(n) = 0.54 - 0.46 * COS\left(\frac{2 * \pi * n}{N - 1}\right); \; 0 \le n \le N-1$$ …………………………(Eq. 2)

where N is the number of samples per frame. In our system N = 256.

Autocorrelation Analysis

In the third step, each windowed frame is autocorrelated. This is done to minimize the mean square estimation error done in the LPC step. The equation for autocorrelation is:

$$r_l(m) = \sum_{n=0}^{N-1-m} x_l(n)\, x_l(m+m); \; m = 0, 1, \ldots, p$$ ……………………………(Eq. 3)

where **p** is the order of the LPC coefficients. **p** = 13 in our system.

Linear Predictive Coding (LPC)

The features for each frame are extracted by linear predictive coding (LPC) which is represented by the following equation:

$$\hat{s}_n = \sum_{i=1}^{p} s_{n-1} \bullet a_i$$ ……………………………(Eq. 4)

where s_n is the nth speech sample, and the a_i are the predictor coefficients, and $\hat{s}_n$ is the prediction of the nth value of the speech signal. [2]

This is a finite-order all-pole transfer function whose coefficients accurately indicate the instantaneous configuration of the vocal tract.

Cepstral Analysis

Cepstral coefficients are obtained from the LPC coefficients. Cepstral coefficients are used since they are known to be more robust and reliable than LPC coefficients, PARCOR coefficients, or the log area ratio coefficients.[2] Recursion is used on this equation to attain the Cepstral coefficients

$$c(i) = a_i + \sum_{k=1}^{i-1} \left(\left(1 - \frac{k}{i}\right) \bullet a_k c_{i-k}\right); 1 < i \le p$$ …………………………(Eq. 5)

Here **p** = 13, $\boldsymbol{a_k}$ are the LPC coefficients, and ***c(i)*** are the Cepstral Coefficients.

Parameter Weighting

The Cepstral coefficients are then passed through a parameter-weighting step to minimize their sensitivities. Low-order Cepstral coefficients are sensitive to the overall spectral slope and high-order Cepstral coefficients are sensitive to noise and other forms of noise-like variability. The weighted Cepstral coefficients are in the form:

$$\hat{c}_m = w_m c_m; \; 1 \le m \le p$$ …………………………(Eq. 6)

where $\boldsymbol{w_m}$ is given by the following equation:

$$w_m = 1 + \frac{p}{2}\sin\left(\frac{\pi m}{p}\right);\ 1 \le m \le p \text{ ....(Eq. 7)}$$

Again **p** = 13 for our system.

The result is an ***i x j*** matrix, where ***i*** is the order of the LPC and ***j*** is the number of frames. Now that the important characteristics are extracted, the results can be used by VQ and DTW to compare the speaker and word respectively.

Vector Quantization

Vector quantization (VQ) is used for speaker identification in our system. VQ is a process of mapping vectors of a large vector space to a finite number of regions in that space. Each region is called a *cluster* and is represented by its center (called a *centroid*). A collection of all the *centroids* make up a *codebook* . The amount of data is significantly less, since the number of *centroids* is at least ten times smaller than the number of vectors in the original sample. This will reduce the amount of computations needed when comparing in later stages. Even though the *codebook* is smaller than the original sample, it still accurately represents a person's voice characteristics. The only difference is that there will be some spectral distortion. [1]

Codebook Generation

There are many different algorithms to create a *codebook* . Since speaker identification depends on the *codebooks* we generate, it is important to select an algorithm that will best represent the original sample. For our system, we will be using the LBG algorithm [4], also known as the binary split algorithm. This algorithm is well known and C code that implements the algorithm is also freely available. The algorithm is implemented by the following recursive procedure:

1. Design a 1-vector *codebook* ; this is the *centroid* of the entire set of training vectors (hence, no iteration is required here).

2. Double the size of the *codebook* by splitting each current *codebook* y_n according to the rule:

$$y_n^+ = y_n(1+\varepsilon)$$
$$y_n^- = y_n(1-\varepsilon)$$

   where ***n*** varies from 1 to the current size of the *codebook* , and **e** is the splitting parameter. For our system, **e** = 0.001.

3. Nearest-Neighbor Search: for each training vector, find the *centroid* in the current *codebook* that is closest (in terms of similarity measurement), and assign that vector to the corresponding cell (associated with the closest *centroid*). This is done using the K-means iterative algorithm

4. Centroid Update: update the *centroid* in each cell using the *centroid* of the training vectors assigned to that cell.

5. Iteration 1: repeat steps 3 and 4 until the average falls below a preset threshold. In our system it is when 1.001 times the current distance is less than the previous distance.

6. Iteration 2: repeat steps 2, 3, and 4 until a *codebook* of size M is reached.

Figure 1 shows, in a flow diagram, the steps of the LBG VQ codebook generation technique.

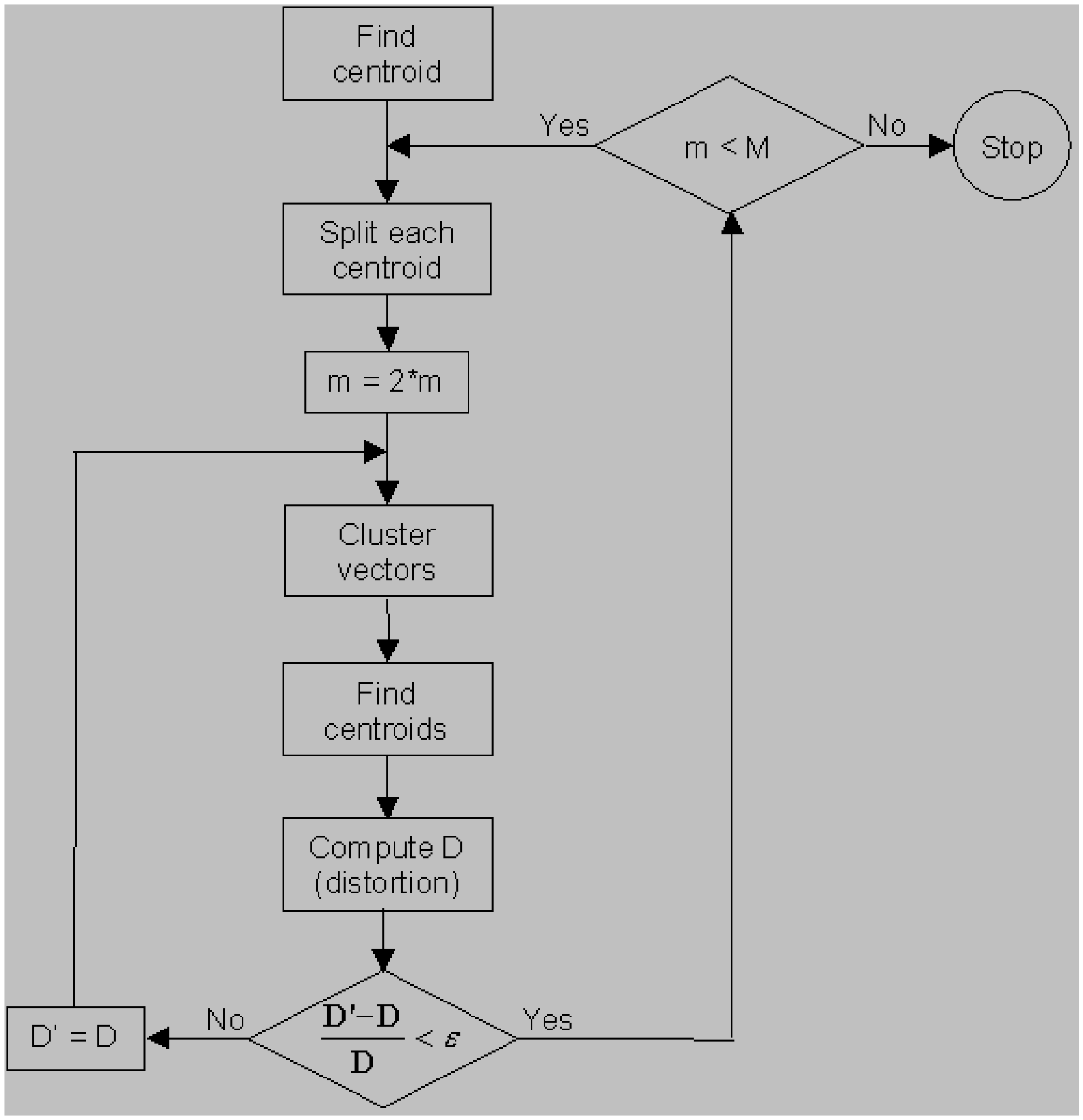

Figure 1 - Flow diagram of LBG algorithm

"Cluster vecto rs" is the nearest-neighbor search procedure, which assigns each training vector to a cluster associated with the closest centroid. "*Find centroids* " is the centroid update procedure. "*Compute D (distortion)* " sums the distances of all training vectors in the nearest-neighbor search so as to determine whether the procedure has converged.

Training

To train a user's voice, we recorded the user reading a couple of sentences from a book. Each person read the following excerpt:

> "In many applications, therefore, it is desirable to consider not only the best matching path but a multiplicity of reasonable candidate paths so that reliable decision processing can be performed. In the current context, the focus is on spectral matching or decoding; however, similar situations can arise in phoneme sequence matching or decoding, word as well as phrase sequence matching of decoding, and so forth. Such cases will be discussed in later chapters when speech-recognition algorithms and systems are presented." [1]

These sentences were chosen since a wide variety of the speaker's pitch and voice levels could be represented. About 40 seconds of a user's voice being recorded, which produced around 2,350 vectors for that sample. A 160-vector codebook was created from this sample and stored in the system's database.

Matching

For the identification process, an unknown person's voice is recorded for 1.5 seconds. The person speaks a single word (password), which will also be used to authenticate the user. Feature extraction is done on the small speech sample, which produces around 60-80 vectors. The vectors are compared to each codebook and the Euclidean distance is taken between the vectors and the centroids of each codebook. The speaker is matched to the codebook with the lowest distance. Figure 2 shows a conceptual diagram to illustrate the matching process. [3], [5]

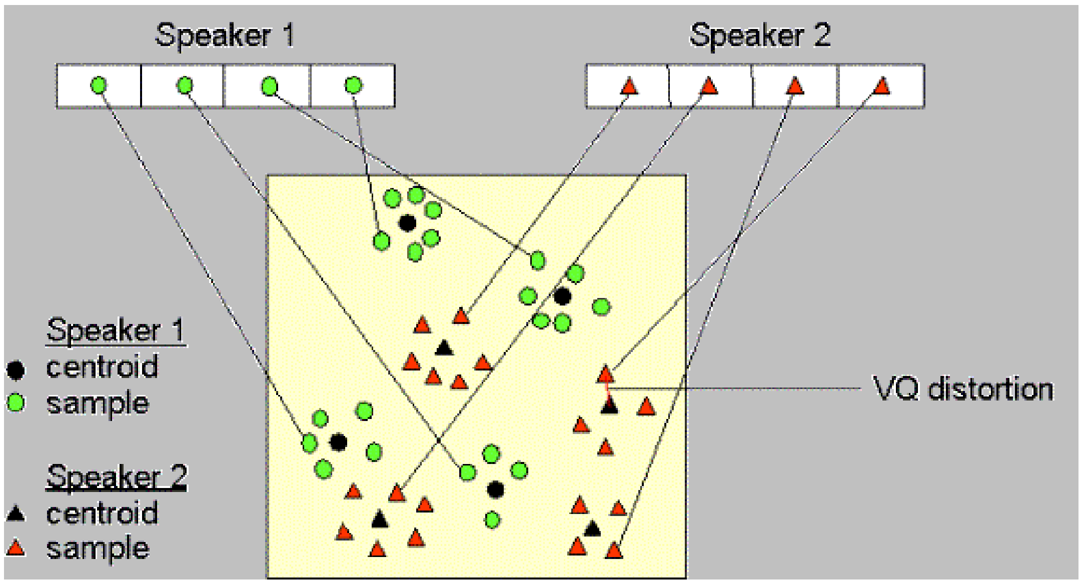

Figure 2 – Conceptual Diagram of Matching

Only two speakers and two dimensions of the features are shown. The circles represent the features of speaker 1 and the triangles represent the features of speaker 2. The solid black circles and triangles represent the centroids of speaker 1's codebook and speaker 2's codebook. VQ distortion is the distance between an unknown sample and the centroid of a codebook. We take the Euclidean distance as the distortion in our system. If the unknown voice was represented by the shaded triangles, as the figure shows, the distance between the shaded triangles and the solid black triangle is less that the distance between the shaded triangles and the solid black circles. Thus the unknown voice will be identified as speaker 2.

Dynamic Time Warping

Dynamic Time Warping (DTW) is used for word recognition in our system. This is found to be a fast and easy way to recognize a spoken word.

Algorithm
The first step is to put two feature sets into a frame. One feature set is the unknown word and the other is a known word that is stored in the database. Usually the known word is put on the y-axis and the unknown word is put on the x-axis of the frame. [1]

First the local distance is taken of the frame. This is the sum of Euclidean distances between both samples at certain points in time. Next the accumulated distance is calculated to determine if the word matches. The equation to find the accumulated distance is given as:

$$D_A(1,1) = d(1,1) \bullet m(1)$$

$$D_A(i_x, i_y) = \min_{(i'_x, i'_y)} [D_A(i'_x, i'_y) + \zeta((i'_x, i'_y), (i_x, i_y))$$

$$d(X, Y) = \frac{D_A(T_x, T_y)}{M_\phi}$$

where ζ is the weighted accumulated distortion (local distance) between point (i'_x, i'_y) and (i_x, i_y)

Three accumulated distances are calculated for the frame, the upper path, the middle path, and the lower path. Figure 3 shows what each path means.

The gray area is all of the space where a distance can be calculated. The upper path is the distance near the gray boundary above the dashed line and the lower path is the distance near the gray boundary below the dashed line. The middle path is the distance near the dashed line and in the figure is represented by the solid line. The minimum distance of these three paths is taken as the final distance between these two samples.

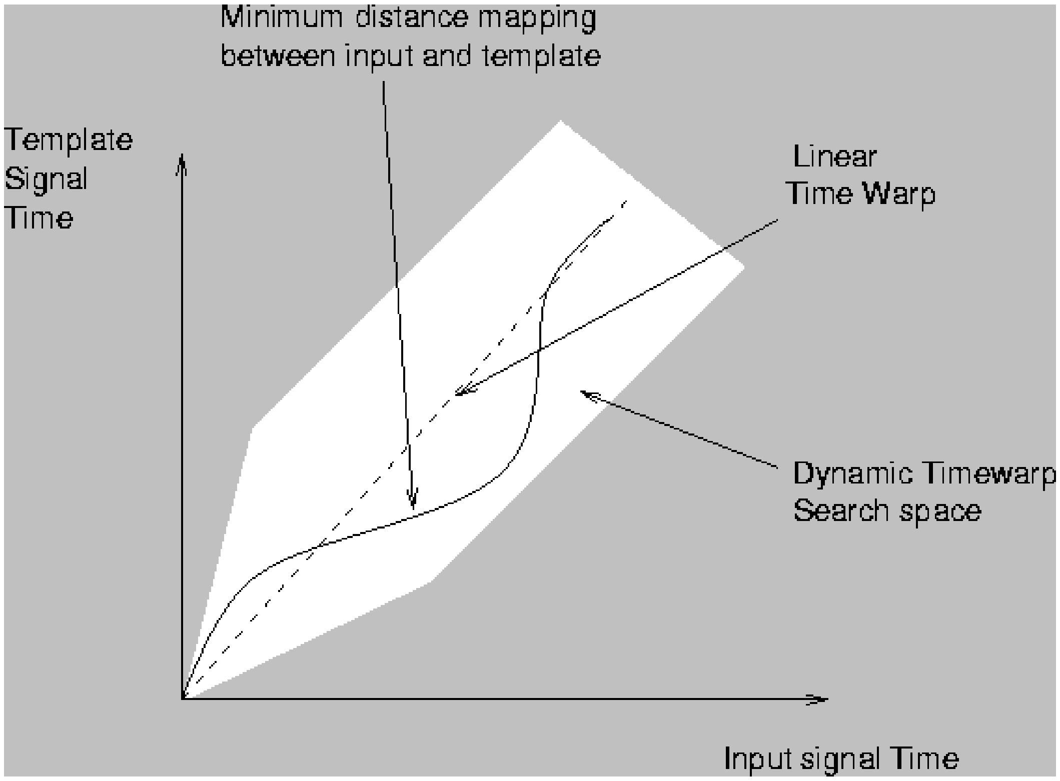
Minimum distance mapping between input and template
Template Signal Time
Linear Time Warp
Dynamic Timewarp Search space
Input signal Time

Figure 3 – DTW Diagram

Matching
For our system, DTW will only work if the size of the unknown sample is 0.5 to 1.5 times the size of the known sample. The distance produced by the DTW algorithm is then compared to a threshold value. If the distance is less than the threshold value, the unknown word corresponds with the known word. After testing different words spoken by different people, the threshold for the DTW was determined.

System Implementation

Unknown voices are first stored as AIFF files and then the files are read into the system. The C code that we found does most of the feature extraction as well as the DTW and codebook generation using AIFF files.

When the system is started, there are three main operations it can do: train a word to be stored in the database, train a person's voice to be stored in the database, and identify an unknown voice.

If the "train person's voice" option is selected, the user records the noted sentence. The system then does feature extraction and generates a codebook for that sample and stores it in the database of known voices.

If the "train word" option is selected, the user records their personalized password. The system does feature extraction and saves the features in the database of known words and corresponds that to a known codebook of the speaker's voice.

If the "identification" option is chosen, the system reads in the unknown word. Feature extraction is then done on the unknown sample and the Euclidean distance is found between that and all the codebooks in the database. The codebook with the smallest distance is selected as the identified speaker. The password for that person is then retrieved from the database and then DTW is done to find how similar they are. If the distance for DTW is below the threshold, the system will identify the person and print it on the screen. If not, the system will ask the user to try again. If identification fails three times in a row, the system will exit.

PC – EVM Interaction

Most of the calculations are done on the EVM. The PC loads a file and tells the EVM what option was chosen (and therefore, what task to perform). The EVM gets the sample by means of the host port interface (hpi), and does feature extraction on it and then one of the following will happen based on the option chosen:

1. The features are then transferred to the PC to be saved in the database
2. The features are used to calculate the codebook on the EVM, then transferred to the PC to be saved in the database
3. The features are used to calculate the Euclidean distance on the EVM with known codebooks that are transferred from the PC

The identified person's password is then transferred to the EVM and DTW is performed. The EVM will transmit a message to the PC stating if it is below the DTW threshold or not. The PC will display one of two messages on the screen, either identifying the person or not.

Optimization

All of the feature extraction was done in internal memory, which used up about 6kB of internal memory. There were also some redundant operations in the code we obtained, which we reduced to optimize the code. As an example, the code calculated the coefficients for the Hamming window for every frame instead of just calculating it once and storing it for future use.

Before optimization was done, feature extraction took ~2,518,573 cycles or ~100 ms per frame to calculate. After optimization, feature extraction took ~407,836 cycles or ~16ms per frame to calculate. However, the most significant cause for improvement in the speed of our system was changing the EVM target options to use the C67xx instead of the C62xx. This allowed the EVM to do floating points calculations a lot faster and since all of our calculations were in floating points, our calculations were done faster. For a voice sample that was 80 frames, or about 1.858 seconds long, it took about 1.3 seconds to do the feature extraction. This is even more important when training a person's voice for the system. If the sentence took 40 seconds to speak, there would be around 1,722 frames that needed to be used for calculations. Feature extraction for the optimized program took ~27.5 seconds while the unoptimized program took ~172 seconds. This saved about 144 seconds or 2.4 seconds.

Testing

We created our own database to test our system. Due to time constraints, we were only able to add seven people to our database. We recorded each person saying their password at least twice and other words to attempt penetrating the system. We had a total of 30 words recorded to test the system. 22 of the words succeeded in its task, either acknowledging the correct person and password or rejecting the wrong person or password. This results in an accuracy of ~73%. The ones that failed identified the wrong person or password.

The main cause of failure in the system is our voice identification portion. We assume that any person who tries to gain access is in the system. In our system, we calculate the Euclidean distance between the unknown word and the codebooks; then the lowest value of the distances is identified as the correct person. Ideally we would have liked to set a threshold and accept the person whose distance falls below the threshold. After a series of tests we found that was not possible. Table 1 shows a part of our recorded VQ test values

Table 1 – VQ test values

	wini-ins	lee-pass	chris-anti	andrea-mu	cathy-au	lin3-eng	sarah-com
wini	2.20885	2.215629	2.259597	2.375617	2.588123	2.185962	2.187027
lee	2.234855	2.087408	2.244103	2.336507	2.566765	2.149922	2.266571
chris	2.32575	2.167784	2.112945	2.361468	2.670387	2.056206	2.145388
andrea	2.34338	2.12003	2.305699	2.250459	2.642527	2.289017	2.372835
cathy	2.260864	2.169406	2.183698	2.380831	2.517215	2.077011	2.099489
lin3	2.285111	2.10679	2.070511	2.309676	2.685701	1.982938	1.996174
sarah	2.377913	2.295451	2.108312	2.433657	2.651677	2.021101	1.980313

We compared a person's password against all of the codebooks and the Euclidean distances were noted. As can be seen in Table 1, we could not set a threshold since the distances varied for each word. If we did set a threshold, either most of the values would be too high and the system would not identify anybody or the values would be too low and almost everybody would be identified as the unknown person. However, if we accepted the person with the lowest distance, 6 out of 7 would be identified as the correct person. If we spent more time experimenting with other algorithms, we could probably use the threshold method.

For word recognition, we were able to set a threshold. After a series of tests we determined that the threshold would be 275. Table 2 shows a small sample of our test results[RST1].

Table 2 – DTW test results

	inspiration	communication
wini	538.030396	288.49795
	238.9096	n/a
	179.8916	n/a
richard	502.103607	197.846527
sarah	N/a	117.28109
	N/a	149.804916

In most of the cases, DTW identified the word correctly. It can also be seen that some of the distances are almost twice the threshold value when it should have been below the threshold. This is mostly likely because of differences in the recording conditions. There could have been sufficient amount of noise in the background to interfere with the spoken word. Also, there could have been some extra silence either at the beginning or end of the file and that could distort the DTW calculations.

Source Code

PC Side - tdsi.c

EVM Side - tdsievm.c

Special Thanks to Voice Trainers

Andrea M Okerholm, amo@andrew.cmu.edu
Tejaswini Hebalkar, hebalkar@cmu.edu
Lee Hotrephinyo, leeh@andrew.cmu.edu
Yue Chang, yuec@andrew.cmu.edu
Christopher Lin, lin3@andrew.cmu.edu
Christopher L Verburg, verburg@andrew.cmu.edu
Sarah E Jackett, jackett@andrew.cmu.edu
Richard S Tseng, srt@andrew.cmu.edu

References

1. L. Rabiner & B. H. Juang, “Fundamentals of Speech Recognition”, Prentice Hall, 1993

2. R. Klevans & R. Rodman, “Voice Recognition”, Artec House, 1997

3. F. Soong, E. Rosenberg, B. Juang, and L. Rabiner, "A Vector Quantization Approach to Speaker Recognition", AT&T Technical Journal, vol. 66, March/April 1987, pp. 14-26

4. Y. Linde, A. Buzo, & R.M. Gray, “An algorithm for Vector Quantization”, IEEE Transactions on Communications, vol. COM-28 No. 1, Jan 1980, pp. 84-95

5. Digital Signal Processing Mini-Project: *An Automatic Speaker Recognition System* http://lcavwww.epfl.ch/~minhdo/asr_project/

[RST1]