[1]

Real-Time, Isolated Speech Recognition Using MFCC and DTW

Kelvin Cheng, *Undergrad, UCLA* and Bilkent Samsurya, *Undergrad, UCLA*

Final Design Report

EE 113D – Digital Signal Processing and Design

***Abstract*—For our senior design capstone project, we worked on a real-time speech recognition program that can identify all 26 letters of the English alphabet, given an environment with low noise and assuming a single speaker. We decided to work on this project because we were interested in seeing if it was possible to improve on known methods of comparison for Mel-frequency cepstral coefficient (MFCC) data, that would yield the most improvement in terms of speed and robustness. Our approach to implementing this program was to first get the MFCC data of the reference inputs and when a given input is recorded, we would perform the same procedure as before on the current input before utilizing dynamic time warping to obtain a "distance" between the input's and each of the reference's MFCC data. The lowest "distance" would thus give us the alphabet the program assumes to be the closest to what the speaker said. Our test results indicate that our letter detection program has an average accuracy of 65.7%. While it accurately recognizes certain letters, it is susceptible to getting to similar-sounding letters mixed up.**

***Index Terms*—Speech Recognition, Isolated Speech, Discrete Fourier Transform, Mel-frequency Cepstral Coefficients, Dynamic Time Warping, Speaker-dependence.**

I. Introduction

A. History

The ability of humans to so effortlessly understand speech has long fascinated scientists, and with the advent of computers that can store and process audio signals, the challenge emerged to enable computers to hear a person speaking and recognize what he/she is saying [4]. The task is challenging, but can roughly be broken down into three parts: 1) isolating speech segments using endpoint detection methods, 2) extracting the essential features of the speech, and 3) analyzing the features to predict what the speaker is saying [6]. In this paper, we describe our process of designing and implementing a speech recognizer using some well-known techniques, namely MFCC and DTW [5]. The goal is to have a program that recognizes isolated words from a built-in vocabulary. In the following pages, we describe our algorithm for detecting the endpoints of speech, the steps in extracting a speech segment's features using MFCC, and the dynamic time warping algorithm used to compare one speech sample to another. The results of our tests show signs of some effectiveness, but the accuracy will need to be improved before this program can be reliably used in a practical setting.

B. Global Constraints

1) *LCDK and CCS*: Our code is written and compiled in the Code Composer Studio (CCS) IDE, and run on the L138/C6748 Development Kit (LCDK). We make use of the built-in libraries CCS offers for efficient FFT computations, as well as the framework it provides to read digital audio samples from a microphone at a fixed sampling rate. Due to the limited amount of memory in the LCDK's embedded processor, we had to take care to use the available memory efficiently, for example, by limiting the size of our input array and using single-precision floating point instead of double-precision when possible.

2) *Time Constraints Due to Sampling Rate*: Our program takes in audio samples at 8 kHz. Every time a sample comes in, an interrupt handler is triggered. To make sure no samples are missed, we had to make sure the interrupt handler code finishes in less time than one over the sampling rate.

3) *Noise and Speaker Dependence*: Our current version of the speech recognizer assumes there is little to no background noise [2]. A noisy environment is likely to throw off our program's endpoint detection algorithm as well as its word-identification algorithm. Also, based a few brief tests and because the literature classifies DTW as a speaker-dependent method, we assume the test speaker is the same person who generated the reference words [5]. If the speaker turns out to be a different person, the behavior can shift drastically.

II. Motivation

In researching current and existing speech recognition tools, we noticed that they tend to:

1) Have trouble differentiating between similar sounding words/alphabets.

Some words/alphabets have very similar phonemes connecting them, and when they are spoken to a microphone/recording device, they tend to "confuse" the program, resulting in the program misidentifying the spoken word for another. This is largely due to the way existing programs compare the incoming input (the spoken word) with the reference data. Our design implementation was aim to

[1]

minimize the error rate, and maximize the accuracy in this domain.

2) Have low latency.

In order for many current speech recognition tools to accurately identify each word that is spoken, they have to be spoken and then have a wait period before the next word can be spoken. Depending on the type of comparison methods used, (hidden Markov model, neural network, dynamic time warping, etc.), in many speech recognition programs, the wait time can be rather long because the program has to stop recording so that the just spoken word can be processed identified. This results in a rather clumsy implementation that does not seem to serve many practical purposes. Of course, our implementation will not be able to completely eradicate this "wait-time", but our design will aim to decrease and minimize this time as much as possible, so as to give the speaker the illusion that he/she can have an actual conversation with the program..

3) Be Speaker Dependent

From all the sources that we have gathered thus far, the accuracy of the program pivots on the size of the data sets provided to the program, that is the larger the size of the data, the more references the program has access to. In this way, with more "material" to compare the incoming input with, the more accurate, the program will be. If the program was only given data sets provided by only 1 speaker then the program is only able to distinguish words if the input was coming from that same speaker. We wanted to improve upon this feature by attempting to make it speaker independent so that the program would be able to work as expected for a larger individual.

III. Approach

A. Team Organization

In working on this project, Kelvin mainly handled the endpoint detection and MFCC-computing code, and Bilkent was in charge of the dynamic time warping algorithm. When we got together in the lab, we would look at each other's code to point out bugs and areas of improvement, and also run tests on our code.

B. Plan and Implementation

A) Research Methods of Speech Recognition

The first step in our project was to research methods of speech recognition--specifically, feature extraction techniques and speech comparison techniques. From our readings [5][6] and our experience from a past mini-project, we decided MFCC was a good choice for finding the features of a speech signal relevant to recognition. Deciding on a comparison technique was slightly more complicated. We found that there are three common techniques used in recognition systems: dynamic time warping (DTW), hidden Markov models (HMM), and artificial neural networks (ANN). After some deliberation, we decided on DTW for its simple implementation and the way it processes a speech segment as a whole, allowing us to skip over the finer details of recognizing individual phonemes in a word.

B) Implement Program in CCS

Once we settled on our overall strategy to use MFCC and DTW [5], the next task was to implement them in CCS. At first, we set aside real-time processing and focused on processing a single-word speech signal. We wrote a program, *record_example.c*, that records 2 seconds of audio, splits the signal up into frames, performs endpoint detection, computes the MFCC of each speech frame, and stores the MFCC data into a file.

When we got this program working, we ran it on different words and stored each word's features into a separate file. Once we had the data for each word, we stored all of it into a 3D array in a header file, *alphabet_reference.h*; this array contains data for several words, each word is made up of several frames, and each frame has 13 MFCC coefficients.

Then we wrote a new program, *dtw.c*. Similarly to *record_example.c*, it gets the MFCC data from a word spoken in a 2-second interval. Then it runs our DTW algorithm to compute a "distance" between the spoken word and each reference word. It predicts what word was spoken based on the reference word with the shortest "distance" [5].

C) Test Program

Once we determined that our *dtw.c* program worked properly, the time came to test its accuracy. Our choice for the recognizer vocabulary was the 26 letters of the alphabet. In the beginning, we tried it with just the first 5 letters, and it seemed to do very well, as it made the correct prediction most of the time. Then we added each of the other letters, and the accuracy started to degrade; it did well for some letters, but quite poorly for others.

D) Enhance Program

After our first round of tests, we wanted to enhance our program so that it could (1) record samples continuously and do endpoint detection/recognition in real time, and (2) make more accurate predictions.

For the first goal, we modified our program to store samples in a circular buffer and do endpoint-detection in real-time. When a segment of speech is successfully isolated, the program switches from recording mode to recognition mode, in which the speech frames' MFCCs are computed and fed into the DTW algorithm. Once a prediction is made, the program returns to recording mode and waits for the next word to be spoken.

For the second goal, we remembered from our literature review that in addition to MFCC coefficients, one can also compute Delta and Double-Delta coefficients for each frame [5], which helps capture the dynamic character of the speech. Since these coefficients can be computed directly from the MFCC data, we did not have to re-record our reference examples. Instead, we added code to *dtw.c* that computes these

additional coefficients and includes them in the DTW function.

Once we did this, we re-tested our recognizer. This time, because we added real-time processing capabilities, we could test multiple words without having to restart the program.

C. Standard

Our project in part consisted of utilizing and implementing de facto standards, mainly the mel-frequency cepstral coefficients (MFCCs) and the dynamic time warping (DTW). Other related standards include Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), and short-time energy (STE).

D. Theory

1) Endpoint detection

When the program is running, input from the microphone is continuously being sampled at a frequency of 8000Hz before being stored into a circular buffer. If the number of samples we've acquired since the last complete frame is equal to a frame length, then we compute the short-time energy (STE) of that given frame. The reason for computing the STE is because we want to distinguish speech from random background noise. We would expect a speech signal frame to be of higher STE magnitude when compared to the STE of a noise frame. If the STE of the given frame exceeds the predetermined threshold, it means that the speech has begun.

Saving the start of the array index corresponding to the start of speech will allow the main function to be able to have the information to access the appropriate index of the circular buffer at which the speech signal begins. We continue computing the STE of each incoming frame and if we detect a frame having an STE below the threshold, we begin counting the number of frames preceding the first frame that falls below the threshold. If the number of these frames (quiet frames), exceeds a threshold, it would be confirmation that the speech has ended and we can proceed with post processing. We keep track of the number of quiet frames to ensure that the program does not "confuse" a brief pause within a speech signal as being the end of a speech signal. In doing so, we save the index of the buffer corresponding to the end of the speech (first instance when the given frame STE falls below the STE threshold), before stopping the interrupt function from getting more samples and transferring control to the post-processing code in the main function.

2) MFCC

The mel-frequency cepstral coefficients (MFCCs) collectively represent the short-term power spectrum of a particular sound [5]. Moreover, the frequency bands are equally spaced on the mel band which resembles and approximates the human auditory system. As such, these MFCC features allow us to store a speech signal's relevant features so that they can be fed into a recognition algorithm. To obtain the MFCC features of a given speech signal, we have to perform the following steps and transformations: Given a frame's power spectral density (PSD), we:

1. Apply a series of triangular filters to the PSD of the given frame and store it in an array, Y.
2. Take the log base 10 of each element within Y and store it into array X .
3. Take the Discrete Cosine Transform of X, and keep the first 13 elements of the result.

3) Delta and double-delta coefficients

When running tests with our program, we saw that the rate at which the program correctly identified each spoken letter was below 50 %. We had to do further research on ways to help improve upon our initial implementation of MFCC features. We decided on utilizing delta and double-delta coefficients because they allow us to add information to existing speech references and inputs without requiring us to change our implementation. In doing so, the delta and double-delta features give a given reference or input signal more characteristics for us to use as comparison, allowing for greater accuracy. The implementation of the delta and double delta coefficients are as follows:

1. We store these coefficients in a 2D array of the same size as the MFCC features for a given speech signal
2. For each delta[k][i], k representing the number of frames in speech, and i representing the current coefficient element, we compute the, delta[k][i] = input_mfcc_features[k+1][i] -input_mfcc_features[k-1][i] + input_mfcc_features[k+2][i] -input_mfcc_features[k-2][i])/10; In other words, we compute the set of delta coefficients k, by taking the sum of the corresponding mfcc's k+1 frame , k-1 frame , k+2 frame , k-2 frame before normalizing the sum by 10.
3. If k is 0, we took delta[k][i] = input_mfcc_features[k+1][i] -input_mfcc_features[k][i] +

input_mfcc_features[k][i]-input_mfcc_features[k][i])/10

4. If k is 1, we computed delta [k][i] = input_mfcc_features[k+1][i] - input_mfcc_features[k-1][i] + input_mfcc_features[k+2][i] -input_mfcc_features[k-1][i])/10
5. The same principle applies when k = 11 and 12.

4) Dynamic time warping

To compare the MFCC features of two speech signals after we extract them, we use dynamic time warping (DTW) algorithm. This approach is useful because for a letter or word spoken at different speeds when compared to the one stored at the reference, we still want the program to correctly identify it [5].

Further, it is a relatively easy concept to understand and implement. In our implementation, we designate the y-axis for the reference and x-axis for the input, and each cell will be the sum of the difference in magnitude between the input and reference's MFCC features, delta and double-delta coefficients. When the table is completed, we look at the value stored at the upper-right most cell in the table. This value is the "distance" between the two speech segments. We repeat this procedure for the same input with all other references, and the reference corresponding to the lowest magnitude, would be the identified word/alphabet.

The computation of the DTW table is as follows [10]:

1. Start from bottom left most cell and incrementally work to the upper right most cell
2. For the first cell, we compute the magnitude difference between the input and reference signal by calculating the magnitude from the mfcc features, delta and double delta coefficients, before storing the value in a cell.
3. Following the first cell, the next cell is calculated by repeating step 2 and incrementing the result with the smallest value among the cell left from the current cell, bottom from the current cell and bottom left of the current cell.
4. If the cell we are calculating is at the edges, bottom or top of the table, we would simply use the cells available for getting the minimum value. For example if the table is 4 by 4 and we are currently at cell, table[1][0] (upper-left most), we would perform step 2 and increment the result with only the cell below it [0][0] because there is no cell left or bottom left of the current one.

E. Operation – Software/Hardware

1. Hardware:

1. LCDK
2. Computer
3. Microphone

2. Software:

1. Code Composer Studio (by Texas Instruments)
2. DSPLIB

3. To run our program,

1. Load CCS
2. When completed, load the GUI html program
3. Build the program by pressing Ctr-b
4. Refresh the LCDK and press F11
5. Run the program and speak letters into the mic

IV. Results

Once we completed our program, we tested its accuracy with test runs. For each word in our vocabulary, Kelvin spoke it into the microphone, and then we checked the recognizer's prediction in the debug window. The following table shows the accuracy for each word (out of 10 trials), along with incorrect words that the recognizer predicted.

There is a wide range of accuracies, ranging from 0 correct predictions on some letters, to 10 out of 10 on others. Taken altogether, assuming each letter has equal weight, the average

Table 1
Speech Recognizer Accuracy Trial

Letter of the Alphabet (pronunciation)	Accuracy (%)	Mistaken letters
A (ey)	100	none
B (bee)	20	D, P, V
C (see)	30	B, D, P, T
D (dee)	60	E, P
E (ee)	90	D
F (ef)	90	S
G (jee)	20	D, E, T
H (eych)	100	none
I (ahy)	80	K
J (jey)	90	A
K (key)	40	A, J
L (el)	90	F
M (em)	100	none
N (en)	90	M
O (oh)	90	L
P (pee)	90	T
Q (kyoo)	40	B, S, P, T, U
R (ahr)	80	F
S (es)	70	F
T (tee)	60	P, V
U (yoo)	70	F, M
V (vee)	20	B, P
W (duhb-uhl-yoo)	100	none
X (eks)	0	F, S
Y (wahy)	90	F
Z (zee)	0	B, P, T, V

accuracy is about 65.7%. We make a few observations. First, the letters that contain the ee vowel sound are very often confused by the recognizer. This is not too surprising, given that there are 9 of them (B, C, D, E, G, P, T, V, Z), and virtually the only part that's different is the consonant part (or lack thereof) in the beginning. We have noticed that fine-tuning the STE threshold in our endpoint detection algorithm helps to ensure the consonant sounds are not excluded from the analysis.

Interestingly, the recognizer did very well in distinguishing the letter "M" from the letter "N", and the letter "I" from the letter "Y".

To conclude, our implementation does show some aptitude for speech recognition. Although its accuracy is suboptimal and its behavior is not very robust, it performs much better than random chance. Considering the complexity of human speech, we consider it no small feat that our program is able to get certain letters right most of the time, and for others, at least its mistakes are usually confined to similar-sounding letters.

V. Conclusion

We learned a lot from working on this speech recognition project, from the algorithms we employed to the intricacies of doing real-time DSP. Admittedly, our program is not the most robust, and there are definitely many ways to potentially improve it. In addition to increasing the accuracy, we can envision making the recognizer speaker-independent by storing more references for the same word from a variety of speakers. We can also experiment with different features besides MFCCs and explore different methods of comparison such as Hidden Markov Model and Artificial Neural Networks.

References

[1] "The Disadvantages of Voice Recognition Software." *Techwalla*, www.techwalla.com/articles/the-disadvantages-of-voice-recognition-software.

[2] Purdy, Kevin. "Four Relatively Quick Ways to Improve Your Phone's Voice Recognition." *ITworld*, ITworld, 17 June 2014, www.itworld.com/article/2832683/mobile/four-relatively-quick-ways-to-improve-your-phone-s-voice-recognition.html.

[3] "Training Deep Neural Networks for Reverberation Robust Speech Recognition - VDE Conference Publication." *Design and Implementation of Autonomous Vehicle Valet Parking System - IEEE Conference Publication*, Wiley-IEEE Press, ieeexplore.ieee.org/document/7776212/.

[4] Hannun, Awni. "Speech Recognition Is Not Solved." *Speech Recognition Is Not Solved – Awni Hannun – Writing About Machine Learning*, awni.github.io/speech-recognition/.

[5] B. Mohan and R. Babu N, "Speech Recognition using MFCC and DTW." In: *2014 International Conferance on Advances in Electrical Engineering (ICAEE)* (Jan. 2014), pp. 1-4. https://ieeexplore.ieee.org/document/6838564/metrics?reload=true

[6] J. Deller, J. Proakis, and J. Hansen, *Discrete-Time Processing of Speech Signals*. New York: Macmilian Publishing Company, 1993, p. 601-622.

[7] H. Qiang and Z. Youwei, "On Prefiltering and Endpoint Detection of Signal." In: *1998 Fourth International Conference on Signal Processing ICSP '98*. (Oct. 1998), pp. 749-752. https://ieeexplore.ieee.org/document/770320/

[8] "Crypto." *Practical Cryptography*, practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[9] "Mel-Frequency Cepstrum." *Wikipedia*, Wikimedia Foundation, 12 June 2018, en.wikipedia.org/wiki/Mel-frequency_cepstrum.

[10] Using Multi-Dimensional Dynamic Time Warping for TUG Test Instrumentation with Inertial Sensors." Semantic Scholar, www.semanticscholar.org/paper/Using-multi-dimensional-dynamic-time-warping-for-Al-Jawad-Adame/6866489f0db97439f6cdbde3c6a4cbf9dfc42467.