NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ISOLATED WORD RECOGNITION FROM IN-EAR MICROPHONE DATA USING HIDDEN MARKOV MODELS (HMM)

by

Remzi Serdar Kurcan

March 2006

Thesis Advisor:	Monique P. Fargues
Co-Advisor:	David Jenn
Second Reader:	Ravi Vaidyanathan

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE *Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Isolated Word Recognition From In-Ear Microphone Data Using Hidden Markov Models (HMM)

5. FUNDING NUMBERS

6. AUTHOR(S) Remzi Serdar Kurcan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

This thesis is part of an ongoing larger scale research study started in 2004 at the Naval Postgraduate School (NPS) which aims to develop a speech-driven human-machine interface for the operation of semi-autonomous military robots in noisy operational environments. Earlier work included collecting a small database of isolated word utterances of seven words from 20 adult subjects using an in-ear microphone. The research conducted here develops a speaker-independent isolated word recognizer from these acoustic signals based on a discrete-observation Hidden Markov Model (HMM).

The study implements the HMM-based isolated word recognizer in three steps. The first step performs the endpoint detection and speech segmentation by using short-term temporal analysis. The second step includes speech feature extraction using static and dynamic MFCC parameters and vector quantization of continuous-valued speech features. Finally, the last step involves the discrete-observation HMM-based classifier for isolated word recognition. Experimental results show the average classification performance around 92.77%. The most significant result of this study is that the acoustic signals originating from speech organs and collected within the external ear canal via the in-ear microphone can be used for isolated word recognition.

The second dataset collected under low signal-to-noise ratio conditions with additive noise results in 79% recognition accuracy in the HMM-based classifier. We also compared the classification results of the data collected within the ear canal and outside the mouth via the same microphone. The second dataset collected under low signal-to-noise ratio conditions with additive noise results in 79% recognition accuracy in the HMM-based classifier. We also compared the classification results of the data collected within the ear canal and outside the mouth via the same microphone. Average classification rates obtained for the data collected outside the mouth shows significant performance degradation (down to 63%), over that observed with the data collected from within the ear canal (down to 86%). The ear canal dampens high frequencies. As a result, the HMM model derived for the data with dampened higher frequencies does not accurately fit the data collected outside the mouth, resulting in degraded recognition performances.

14. SUBJECT TERMS In-Ear Microphone, Isolated Word Recognition (IWR), Short-Term Energy (STE), Zero-Crossing Rate (ZCR), Mel-Frequency Cepstral Coefficients (MFCC), LPC-derived Cepstral Coefficients (LPC-CC), Vector Quantization (VQ), K-Means Clustering Algorithm, Hidden Markov Models (HMM), Forward-Backward Algorithm (Baum-Welch Re-estimation Algorithm), Viterbi Algorithm

15. NUMBER OF PAGES
178

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

ISOLATED WORD RECOGNITION FROM IN-EAR MICROPHONE DATA USING HIDDEN MARKOV MODELS (HMM)

Remzi Serdar Kurcan
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1999

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

and

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2006

Author: Remzi Serdar Kurcan

Approved by: Monique P. Fargues
Thesis Advisor

David Jenn
Co-Advisor

Ravi Vaidyanathan
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

Dan Boger
Chairman, Information Sciences Department

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis is part of an ongoing larger scale research study started in 2004 at the Naval Postgraduate School (NPS) which aims to develop a speech-driven human-machine interface for the operation of semi-autonomous military robots in noisy operational environments. Earlier work included collecting a small database of isolated word utterances of seven words from 20 adult subjects using an in-ear microphone. The research conducted here develops a speaker-independent isolated word recognizer from these acoustic signals based on a discrete-observation Hidden Markov Model (HMM).

The study implements the HMM-based isolated word recognizer in three steps. The first step performs the endpoint detection and speech segmentation by using short-term temporal analysis. The second step includes speech feature extraction using static and dynamic MFCC parameters and vector quantization of continuous-valued speech features. Finally, the last step involves the discrete-observation HMM-based classifier for isolated word recognition. Experimental results show the average classification performance around 92.77%. The most significant result of this study is that the acoustic signals originating from speech organs and collected within the external ear canal via the in-ear microphone can be used for isolated word recognition.

The second dataset collected under low signal-to-noise ratio conditions with additive noise results in 79% recognition accuracy in the HMM-based classifier. We also compared the classification results of the data collected within the ear canal and outside the mouth via the same microphone. Average classification rates obtained for the data collected outside the mouth shows significant performance degradation (down to 63%), over that observed with the data collected from within the ear canal (down to 86%). Recall that the ear canal dampens high frequencies. As a result, the HMM model derived for the data with dampened higher frequencies does not accurately fit the data collected outside the mouth, resulting in degraded recognition performances.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

A/D	Analog-to-Digital
ANC	Active Noise Control
ANN	Artificial Neural Network
ARPA	Advanced Research Projects Agency
ASR	Automatic Speech Recognition
B-W	Baum-Welch
CMC	Cepstral Mean Correction
CSR	Continuous Speech Recognition
DARPA	Defense Advanced Research Projects Agency
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DHMM	Discrete-symbol (or discrete-observation) Hidden Markov Model
DSP	Digital Signal Processing
DTW	Dynamic Time Warping
EM	Expectation-Maximization
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
FTE	Frame-Based Teager Energy
HMM	Hidden Markov Model
IID	Independent and Identically Distributed
IIR	Infinite Impulse Response
IWR	Isolated Word Recognition
LBG	Linde-Buzo-Gray
LFCC	Linear Frequency Cepstrum Coefficient
LPC	Linear Predictive Coding
LPC-CC	LPC-derived Cepstral Coefficient
MFCC	Mel-frequency Cepstral Coefficient
NIST	National Institute of Standards and Technology
PLP	Perceptual Linear Prediction

RC	Reflection Coefficient
RCC	Real Cepstral Coefficient
RCD	Remote Control Device
SNR	Signal-to-Noise Ratio
STE	Short-Term Energy
STZC	Short-Term Zero-Crossing
TIMIT	Texas Instruments and Massachusetts Institute of Technology
VQ	Vector Quantization
ZCR	Zero-Crossing Rate

ACKNOWLEDGMENTS

First, I would like to thank my thesis advisor Dr. Monique P. Fargues. Her precious guidance, continuous help, patience with my never-ending questions and inspiration kept me on track during the course of this thesis work. I would like to also thank my co-advisor Dr. David Jenn and my second reader Dr. Ravi Vaidyanathan for their valuable contributions.

Special thanks go to my best friends John Ortiz and his wife Danielle Ortiz. Their friendship and hospitality will never be forgotten.

Finally, I owe my parents Kemal Kurcan and Emel Kurcan and my sister Selda Kurcan for their endless support and love. Their encouragement during both bad times and good times always kept my spirits high when I was alone 10,000 miles away from home. I dedicate this thesis to them.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

The performance of modern speech recognizers may turn out to be poor under adverse conditions, especially when classifiers are trained under high signal-to-noise ratio (SNR) environments like noise-free chambers (typically where $\text{SNR} \geq 30\ \text{dB}$) and operated in real-world surroundings of relatively lower SNR. Consequently, many researchers have recently focused on the task of building more noise-robust recognizers that may be operated in noisy environments (within a car or a noisy flight cabin) with higher accuracy.

This thesis study is part of an ongoing larger scale research started in 2004 at the Naval Postgraduate School which aims to develop a speech-driven human-machine interface command and control package applicable for the operation of semi-autonomous military robots in noisy operational environments. Earlier related research by Vaidyanathan [Vaidyanathan, 2004b] showed that the in-ear microphone device selected was sensitive enough to make use of tongue movements for a human-machine interface, which led to considering its extension to speech signals for isolated word recognition.

This thesis study extended the earlier work by Newton [Newton, 2006] who collected a database of isolated word utterances of seven words from 20 adult subjects by using the in-ear microphone. The research conducted here develops a speaker-independent isolated word recognizer based on these acoustic signals and a discrete-observation Hidden Markov Model (HMM). Hence, the objectives of this research were to:

- Develop a MATLAB simulation for a simple yet complete isolated word Hidden Markov Model (HMM) recognizer,
- Investigate the feasibility of using the acoustic speech signals collected within the external ear canal via the in-ear microphone for isolated word recognition,
- Explore the performance and noise-robustness of the in-ear microphone data for isolated word recognition in noisy operational environments.

The speech database of this study included seven isolated words {down, up, right, left, pan, move, kill}, each repeated 50 times by 20 adult subjects and recorded with an 8 kHz sampling frequency.

The study began by investigating the characteristics of the in-ear microphone data by spectrogram analysis. The in-ear microphone data can be considered a low-pass filtered version of the speech signals emitted in front of the mouth. Spectrogram analysis showed that most of the signal energy was gathered below 2.5 kHz in the acoustic signals of the isolated word utterances. Mean correction was first applied to the in-ear microphone data to eliminate the DC bias of the microphone and then a high-pass filter was used to remove the noise distortion collected around 100 Hz based on the spectrogram analysis. After high-pass filtering, the acoustic signal is split into frames for short-term analysis. The most crucial part of pre-processing of the in-ear microphone data is the front-end detection to locate the speech boundaries and segment the acoustic signals of the isolated words from the non-speech or silence portion. This task becomes harder when bodily-created noises like gulps, coughs, tongue clicks or lip smacks occur during the in-ear microphone recordings. Non-speech events or noise may complicate the endpoint detection of speech waveform especially when dealing with words starting or ending with low energy phonemes (such as the weak fricative /f/, weak plosive burst /p/ or /t/) or nasals (such as /n/ at the end).

Two endpoint detection algorithms were implemented in MATLAB to determine the beginning and the termination of speech in a given isolated word utterance. The first segmentation algorithm combined two short-term speech signal analysis. The short-term energy (STE) measure was used to estimate initial speech boundaries while a second search of the zero-crossing rate (ZCR) was used to refine these boundaries. The second speech segmentation algorithm studied was based on the frame-based modified Teager energy. However, simulations showed this detection approach required longer processing time than the first algorithm due to its computationally-expensive search algorithm. In addition, the results showed the two algorithms provided comparable detection of endpoint locations. As a result, the focus rested on the first approach that uses STE and

ZCR to segment the in-ear microphone database due to its more computationally-efficient algorithm.

The purpose of speech feature extraction is to convert speech waveform to some type of parametric representation at a lower information rate for further analysis. Speech signals have non-stationary characteristics. As a result, the speech waveforms are commonly split into small frames (typically 5 ms to 40 ms) in which the signal characteristics are considered quasi-stationary to allow for short-term spectral analysis and feature extraction. A wide range of parametric speech representations may be used to generate input feature to speech recognizers. The most commonly used speech features include the Linear Predictive Coding (LPC) Coefficients, Real Cepstral Coefficients (RCC), LPC-derived Cepstral Coefficients (LPC-CC) and Mel-frequency Cepstral Coefficients (MFCC). LPC-CC and MFCC parameters were considered in this study to determine the best feature set for the in-ear microphone database. MFCC parameters are usually preferred over others because they are less susceptible to speaker-dependent variations likely to be present in speech signals. Classification results showed that MFCC parameters outperformed the LPC-CC parameters in recognition accuracy for the data under investigation. Dynamic delta-MFCC parameters were also derived to augment the spectral representation of the static MFCC parameters with some temporal information.

A discrete-symbol HMM (DHMM) with eight hidden states was selected as the classifier type because that type of classifier is simple to implement, has low computational load, and has been shown to perform well in isolated word recognition applications. We selected a codebook of length 128 which was generated by applying the K-means clustering algorithm to the training set of speech features. Next, vector quantization (VQ) was applied to map the continuous-valued speech features (12 MFCC and 12 delta-MFCC parameters) to a discrete set of codebook indices (or symbols), and quantized features used to estimate the optimum model parameters.

Two thirds of the data were used for generating the codebook and training the DHMM. The remaining one third of the data were used for testing the performance of the isolated word HMM recognizer. The average classification rate for this study’s multiple-speaker isolated word HMM recognition system was 92.77%. Results show that the acoustic signals originating from speech organs and collected within the external ear canal via an in-ear microphone could be effectively used for isolated word recognition.

The second dataset collected under low SNR conditions with additive noise resulted in 79.24% recognition accuracy in the HMM-based classifier. This result is consistent with the theoretical noise-shielding property of the human ear, thus justifying the use of in-ear microphone data for speech enhancement and improved recognition under low-SNR conditions.

Finally, we investigated using the in-ear microphone outside the mouth and comparing classification results obtained for the data collected within the ear canal and outside the mouth. Average classification rates obtained for the data collected outside the mouth shows significant performance degradation (down to 63%), over that observed with the data collected from within the ear canal (down to 86%). The ear canal dampens high frequencies. As a result, the HMM model derived for the data with dampened higher frequencies does not accurately fit the data collected outside the mouth, resulting in degraded recognition performances.

I. INTRODUCTION

This thesis study is a part of an ongoing larger scale research started in 2004 at NPS which aims to contribute to the development of a speech-driven human-machine interface command and control package applicable for the operation of semi-autonomous military robots in noisy environments. Towards this end, this research focused on developing an isolated word recognizer using Hidden Markov Models (HMM). The HMM classifier was trained on air pressure signals collected from within the ear-canal via an in-ear microphone because previous research results have shown these signals are less susceptible to environmental noise distortions than those collected in front of the mouth [Westerlund, 2003]. In a broad sense, the ultimate goal of Automatic Speech Recognition (ASR) is to build and train computers/machines or artificial intelligence that can receive and interpret spoken instructions and act upon them properly.

Great progress in ASR has yielded many practical applications in recent years, such as user-friendly speech interfaces in control consoles of cars, credit card number recognition and the verbal selection of menus over the telephone. However, after 50 year-long research efforts and considerable advances in ASR notwithstanding, robust speech recognition for human-machine interface still remains a challenging problem today. The performance of the modern speech recognizers may turn out to be poor under adverse conditions, especially when classifiers are trained under high signal-to-noise ratio (SNR) environments like noise-free chambers (typically where $\text{SNR} \geq 30\text{ dB}$) and operated in real-world surroundings of relatively lower SNR [Deller, 2000]. In contrast, a healthy human listener's performance is usually far more stable on average under similar training and operating conditions. Unfortunately many researchers agree that human-quality, adaptively-learning and noise-robust machines that recognize and interpret human speech will not be achieved in the near future [Deller, 2000; Gold, 2000; Owens, 1993]. However, even incremental improvements leading toward this ultimate goal in ASR are of great importance.

Next, a brief history of speech recognition technology is presented before addressing the difficulties in the ASR area.

A. INTRODUCTION TO AUTOMATIC SPEECH RECOGNITION (ASR)

Speech is the natural and the fundamental way of communication for most humans. Technically speaking, Automatic Speech Recognition (ASR) refers to a mechanism (hardware and software combined) that stores some representations of distinguishing characteristics of speech with a source of input equipment, such as a microphone and further processes these representations to match them to incoming speech in an effort to interact with machines, computers and/or human users

The first primitive recognizer was developed at Bell Labs during the early 1950s. However, it was the 1960s that brought many major breakthroughs to the field of ASR. Some of these achievements are noteworthy to mention herein because they did not only develop significant tools for speech recognition but also established the very basic concepts on which this thesis work is mainly based. Specifically, the development of the *Fast Fourier Transform (FFT)* by Cooley and Tukey in 1965 decreased the computational load of Discrete Fourier Transform (DFT) with a faster algorithm, thereby enabling the practical implementations of Digital Signal Processing (DSP) custom chips [Cooley, 1965; Gold, 2000]. Oppenheim, Schafer, and Stockham introduced *Cepstral Analysis* which performs deconvolution of the speech signal to separate an excitation sequence from an impulse response convolved with it [Oppenheim, 1968]. Cepstral coefficients and many derivatives have been widely used to represent the short-term spectral envelope of speech signals thus far.

It was also the late 1960s and early 1970s that saw another useful method for speech analysis, known as *Linear Predictive Coding* (LPC). One of the earliest and complete papers on the application of linear prediction to speech analysis was published by Atal and Schroeder [Atal, 1971; Deller, 2000]. Basically, LPC uses a pole-only (autoregressive) filter to model the speech signal. LPC coefficients and its derivatives are extensively used for transmitting speech spectral envelope information [Gold, 2000].

Most notably, the foundations for the statistical technique of *Hidden Markov Modeling*, which models an observed sequence as produced by a sequence of hidden states, dates back to the 1960s as well. However, the first successful applications of Hidden Markov Modeling to speech recognition were realized in the 1970s [Gold, 2000].

Baum and his colleagues developed a popular expectation-maximization (EM) algorithm, known as the *Baum-Welch Re-estimation Algorithm* (or Forward-Backward Algorithm), to estimate the parameters of a Hidden Markov Model (HMM) iteratively [Baum, 1966; Baum, 1970]. Hidden Markov Models (HMM) and the Baum-Welch Re-estimation Algorithm are widely used today in contemporary state-of-the-art speech recognition systems.

Dynamic Time Warping (DTW), a deterministic alternative approach to the statistical HMM was also introduced in the 1970s. DWT normalizes the different-length utterances of the same word and applies template-based classification to speech recognition.

Many different approaches incorporating DTW, HMM and Artificial Neural Networks (ANN) were developed for speech recognition in the 1970s. Among these studies, the project of the *Advanced Research Projects Agency* (ARPA), was a remarkable achievement in that it performed a 1000-word ASR system by using connected speech from a few speakers with a word error rate of less than 10% [Gold, 2000]. In the 1980s, the project of the Defense Advanced Research Projects Agency (DARPA Project) and the major other programs conducted by *Texas Instruments and the Massachusetts Institute of Technology* (TIMIT Project) and the *National Institute of Standards and Technology* (NIST) primarily concentrated on the collection of large corpora used for training and testing speech recognizers. These large corpora were subsequently used by the ASR research community at large for performance comparison of different approaches applied to speech recognition.

The ASR community witnessed some other important developments in the 1980s as well. Among those, the *Mel-Cepstrum Analysis* introduced by Davis [Davis, 1980], and the *Dynamic Cepstral Coefficients* proposed by Furui [Furui, 1986] can be considered remarkable techniques for speech feature-extraction due to significant improvement in recognition accuracy.

As for the speech recognizers of the 1980s, many researchers were experimenting with frame-based HMM recognizers, ANN recognizers or hybrid schemes combining HMM and ANN in isolated and/or continuous contexts of speech [Gold, 2000]. Most importantly, the contemporary speech recognition systems of today still use these methods predominantly.

In the following decades, advances in computationally-efficient digital computing power and abundant memory made the inexpensive implementation of DSP chips and Field-Programmable Gate Arrays (FPGA) possible. Needless to say, this has been a major boon to the performance of ASR technology. Commercial ASR applications such as speech-to-text dictation software have been available and used extensively since the late 1990s. However, despite considerable progress, many aspects of ASR are still mysterious and problematic even today because the engineering view of human speech production and human ear perception is not yet fully conclusive. Hence, the research community believes that the field of speech recognition is still in its early infancy and will remain to pose a challenging problem for the near future. [Deller, 2000; Gold, 2000].

Next, the dimensions of difficulty involved in ASR are discussed because understanding the complexity of the ASR problem is the first step to practical and achievable solutions.

B. DIMENSIONS OF DIFFICULTY IN ASR TECHNOLOGY FOR HUMAN – COMPUTER INTERACTIONS

Waibel and Lee addressed the question of complexity involved in ASR in [Waibel, 1990] as "*dimensions of difficulty.*" These are the factors that determine the complexity and the specifications of an ASR system. Deller et al. summarizes some of these factors that render speech recognition methods complicated [Deller, 2000].

First, speaker-dependency plays a defining role in speech recognition systems. A *speaker-dependent ASR system* is trained and tested on the utterances of a specific speaker to learn the representations characterizing speech. Thus, this type of recognizer is designed specifically to recognize the speech of its trainer. Most commercially available speech-to-text dictation software mainly uses this mode because it needs to be trained by each specific user before operation. On the contrary, a *speaker-independent system* is trained on many speakers in an effort to make the system capable of recognizing the speech of a generalized population who may be outside of the training population. The models used in this type of recognizers are derived from expected characteristics and are not fine-tunable to the end users. Therefore, there is no need for further training when a new speaker is introduced to a speaker-independent system. This operation mode is more convenient for a telephone application where users navigate through speech-activated menus. However, the convenience of speaker-independent systems may come with lower recognition accuracy than that obtained with speaker-dependent systems.

Second, *continuous speech recognition* (CSR) systems may seem natural and user-friendly at first glance, but they require more sophisticated recognizers to handle word boundaries issues. *Isolated word recognition* (IWR) systems minimize this problem as they require a deliberate pause between each utterance to simplify the end-point detection algorithm needed to locate the beginning and the end of speech. As a result, IWR is expected to have higher recognition accuracy. Generally speaking, CSR systems require more CPU power and memory than ISR systems. Further, inter-speaker and intra-speaker variations of the training population in articulation, pronunciation and intonation make it even harder for CSR systems to determine speech boundaries, thus yielding lower classification accuracy compared to that obtained with ISR systems.

Third, the *vocabulary size* is another key factor that affects the dimension of difficulties in recognition. Considering the number of ambiguous utterances (e.g., “knight” and “night”) and acoustic confusability (e.g., “beer” and “bear”) in the vocabulary of interest naturally, the performance of a particular recognizer is expected to degrade with the increasing vocabulary size [Deller, 2000]. Small vocabulary (less than 100 words) recognizers can perform relatively simpler tasks such as destination sorting systems for shipping tasks, credit card number or telephone number recognition. In these examples, specific models for each word in the vocabulary can be stored in the system and recognition is achieved by an exhaustive search through the whole vocabulary. As the vocabularies become larger, the recognition task creates increasing memory requirements. When training and modeling for each word in a larger vocabulary becomes impractical, models of subword units like syllables and phonemes are preferred over models of words.

Fourth, the waveform of a speech signal is very susceptible to the variations in channel, microphone characteristics, room reverberation or background noise. Nonlinear effects of noise and channel distortion can be very destructive for recognition tasks, especially when no a-priori knowledge is available about their characteristics.

Finally, another important parameter in ASR system performances is whether any linguistic information is built into the recognizer to fine-tune algorithms. Speech recognizers are trained on basic speech unit models such as phones, phonemes, syllables or words. Linguistic (or grammar) constraints deal with how these basic units should be concatenated to form a meaningful message in a particular language. Such linguistic information may be embedded into more sophisticated recognizers.

All the above factors have a major impact on the success of a particular ASR system and add up to determine the necessary level of system complexity in design phase. Among all of the above-mentioned factors, performance degradation in noisy real-world environments is probably the most significant factor limiting the progress of ASR technology today. Consequently, many researchers have recently focused on the task of building more noise-robust recognizers that may be operated in noisy environments (within a car or a noisy flight cabin) with higher accuracy [Moon, 1997; Shozakai, 1998; Graciarena, 2003]. Lippmann et al. reported a direct comparison of human listeners and machine recognizers on a range of tasks in his speech recognition research [Lippman, 1997; Gold, 2000]. His findings supported the fact that noise considerably degrades the performance of ASR systems even for simple tasks, whereas the superiority of human performance increases in noise and for more difficult speech contexts such as spontaneous speech unlike machine recognizers. His experimental results on human perception of distorted speech sets a good benchmark for the ultimate goal of more humanlike, noise-robust and adaptive ASR systems of the future [Gold, 2000].

Next, the main objectives are presented as well as the specifications of the experimental ASR system on which this thesis work is focused.

C. THESIS OBJECTIVE AND EQUIPMENT USED

The objective of this research was to design an isolated word HMM recognizer with a small vocabulary to evaluate the effectiveness of the acoustic speech signals collected within ear canal on speech recognition. This work uses a speech database collected earlier by Newton [Newton, 2006] which consists of isolated word utterances from 20 adult subjects. Each subject repeated a set of seven isolated words {down, up, right, left, pan, move, kill} 50 times, recorded with an 8 kHz sampling frequency.

The speech signals were collected via the in-ear microphone shown in Figure 1.1 in an indoor office environment with medium SNR levels to simulate real-world conditions and potential operational environments. A second set of data were collected under additive noise in the same office environment with the in-ear microphone located in the ear first and then in front of the mouth to evaluate the noise shielding properties of the in-ear microphone. Figure 1.2 shows the whole equipment set-up used for database collection including the in-ear microphone, analog-to-digital (A/D) converter, PCMCIA card and a general purpose notebook computer.

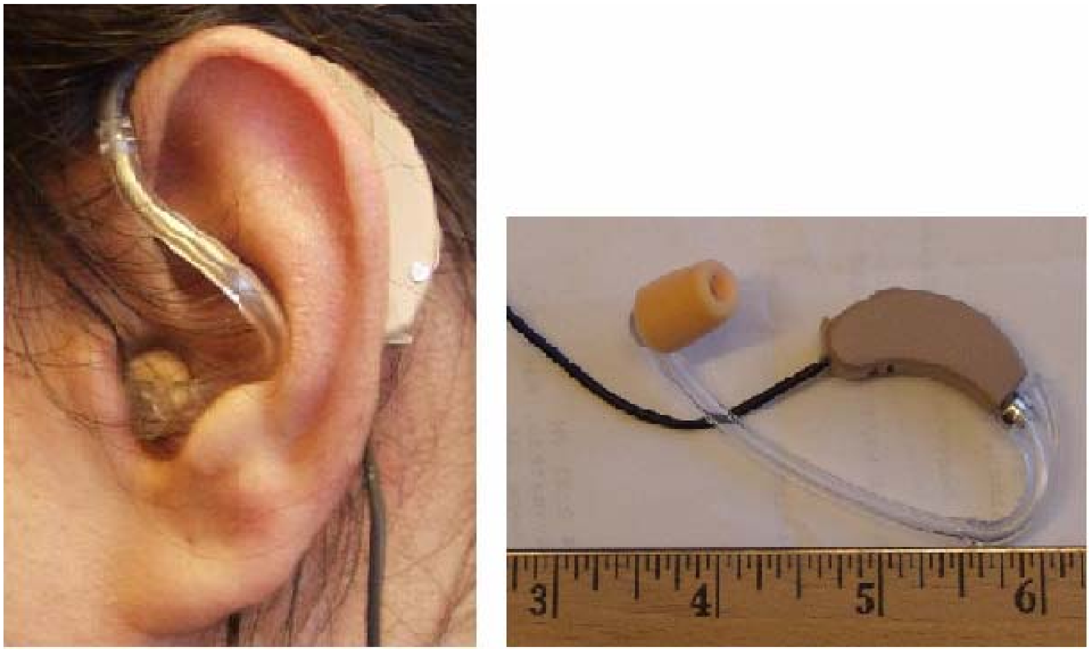


Figure 1.1. Foam-Encased in-Ear Microphone Device.

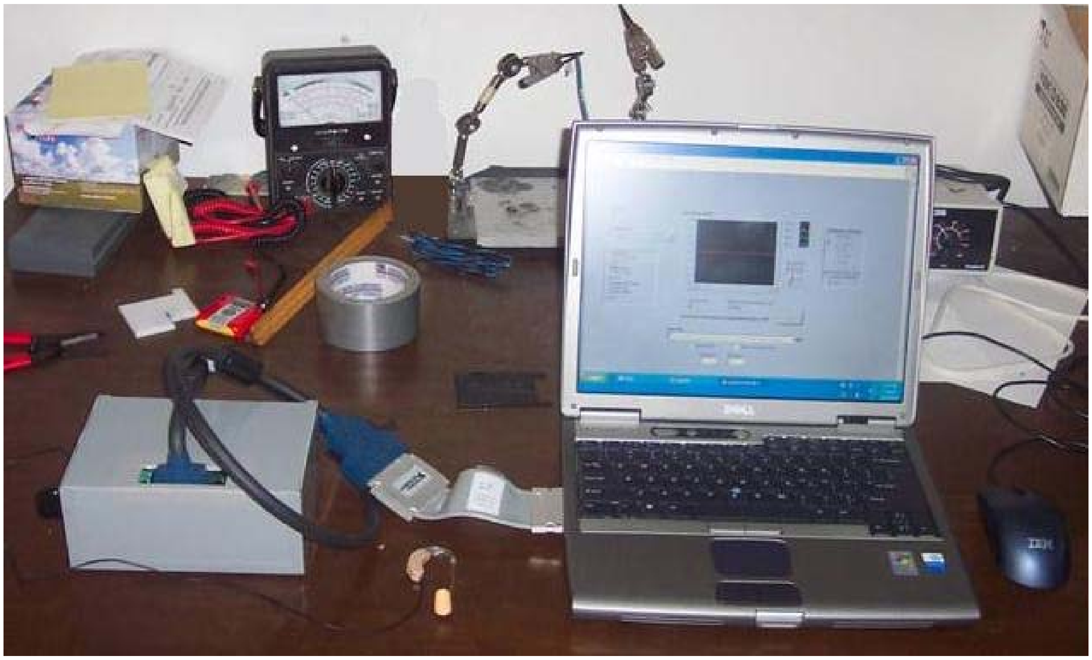


Figure 1.2. Equipment Components Used for Speech Data Collection.

Next, the motivation behind this study is discussed and previous related research summarized.

D. PREVIOUS RELATED RESEARCH

As ASR technologies are transferred more and more to daily-life applications, the need for greater robustness to noisy environments is growing. A great deal of recent work has highlighted the need for robust speech detection, enhancement and recognition via the use of multiple-microphone arrays [Zhang, 2004] or more sophisticated adaptive noise canceling devices [Westerlund, 2003].

Westerlund et al. proposed a sophisticated approach to improve speech recognition where a Hidden Markov Model (HMM) recognizer was used to evaluate the quality and intelligibility of speech signals recorded using an ear-microphone. One of the pieces of equipment used in his research was an Active Noise Control (ANC) headset equipped with an in-ear microphone and ear muffs. This work showed that the high-pass filtering property of the passive absorbers (ear-muffs) is convenient for noise reduction when combining an ANC headset with an in-ear microphone since the speech signal in the external auditory canal is a low-pass filtered version of the speech signal in front of the mouth. Consequently, Westerlund reported that this combination of passive and active noise reduction methods with the use of an ear-microphone had considerable impact on speech recognition performances, resulting in a 50% recognition rate increase in a severely noisy environment.

The idea that a microphone can be placed on other locations of the body rather than just in front of the mouth was also proposed by other researchers [Zhang, 2004; Graciarena, 2003]. Zhang et al. developed a headset prototype that integrates several heterogeneous sensors including a close-talk and a throat microphone shown in Figure 1.3 for robust speech detection and recognition, achieving promising results.

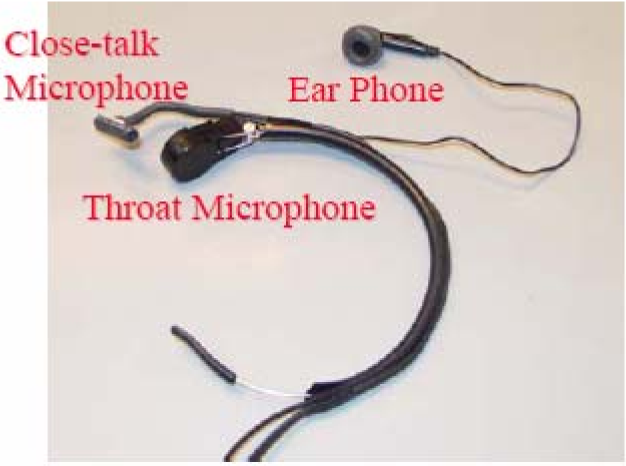


Figure 1.3. Throat and Air-Conductive Microphone (From [Zhang, 2004].).

Graciarena et al. also developed a method that combines standard microphone and throat microphone features in noisy continuous speech recognition experiments. This research provided significant word error rate reduction compared to that obtained with a single microphone [Graciarena, 2003]. Although all the above-mentioned research reported considerable robustness and improvement on speech recognition, the implementations used were either intrusive or involved rather complicated schemes in terms of computational load.

Vaidyanathan et al. recently introduced a unique signal processing strategy which maps the air flow signals generated from tongue movements collected within the ear canal to specific commands [Vaidyanathan, 2004a]. The in-ear microphone used in this work is simpler in design and more convenient to wear when compared to more complicated and intrusive alternatives. The microphone earpiece contains a small passive sensor, which is commercially available off-the-shelf and is similar to those commonly found in hearing aids. This work showed that various tongue movements within the oral cavity create unique and traceable pressure changes in the human ear, which can be captured via an ear-microphone and classified accurately. These results were applied to the hands-free tele-operation of a mobile robot named *“Whegs II hexapod robot.”* This research also reported that

> …Conventional signal processing techniques are generally inadequate to recognize the subtle pressure variations in the ear canal resulting from tongue movement. The ear canal itself is an interference-ridden, noise-amplified environment for acoustic recording. External noise (environmental sounds) can also easily obscure the slight pressure deviations accompanying tongue movement [Vaidyanathan, 2004b].

This earlier research study showed that the in-ear microphone device selected was sensitive enough to collect tongue movements, which led to considering its extension to speech signals. The next section presents the organization of this thesis work.

E. THESIS ORGANIZATION

This initial chapter of the thesis provided an introduction to ASR technology and focused on the main challenges in the ASR field so as to present the fundamentals of ASR. Then the chapter introduced the main objective of this thesis work, equipment used for speech data collection and explained the earlier related research.

Figure 1.4 represents the main framework of the Isolated Word HMM recognizer built to test the viability of using ear-microphone data for speech recognition in this research.

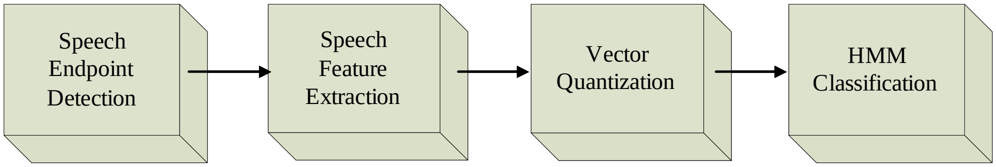


Figure 1.4 Main framework of the Isolated Word HMM recognizer

Chapter II first presents a discussion of the characteristics of in-ear microphone data. Next, the chapter covers the pre-processing of these acoustic signals. The pre-processing involves the short-term signal analysis, the endpoint detection algorithm to determine the onset and the termination of a spoken word in a given acoustic speech signal.

Chapter III introduces the popular speech feature coefficients used to extract spectral information from acoustic signals. The computation of Mel-frequency Cepstral Coefficients (MFCC) is explained using a step-by-step approach. Then, the dynamic delta parameters (delta-MFCC) are derived from the MFCC parameters to supplement information of MFCC parameters with information on its rate of change over time. The rest of the chapter deals with the vector quantization (VQ) since a discrete-observation HMM is to be used rather than a continuous-density HMM. The K-means clustering algorithm is described and used to generate a codebook which maps the continuous-valued speech features (MFCC and delta-MFCC parameters) to a discrete set of codebook indices.

Chapter IV discusses the fundamentals of HMM, the three basic HMM problems and their efficient solution algorithms. The chapter concludes with the discrete-symbol HMM implementation of the isolated word recognition (IWR) system to justify the effectiveness of the air flow signals collected within the ear canal for speech recognition.

Chapter V provides a system overview of the isolated word HMM recognizer and presents HMM classification results.

Chapter VI presents conclusions and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. IN-EAR MICROPHONE DATA DIGITAL PRE-PROCESSING

A. INTRODUCTION

Traditional speech recognition systems typically use a directional microphone to collect speech data. Most directional microphone systems are successful in low-noise environments. However, speech recognition performances may degrade significantly in high noise surroundings, and many different methods have been proposed to improve the robustness of speech recognition systems. Some of these include complex techniques using multiple microphone arrays (to work on multi-channel data), adaptive noise cancellation methods, combining different types of speech features or using sophisticated endpoint detection schemes. However, most of them are either intrusive or impractical to implement in real world applications.

In an effort to guard against noisy surroundings and allow for a relatively simple speech recognition approach, an unconventional recording device, an earpiece microphone placed inside the external ear canal was used for capturing the speech signal. Data were collected using an 8 kHz sampling rate to ensure at least telephone-quality-like spectral information would be available to the recognizer. A small vocabulary consisting of seven isolated words was recorded in an earlier study [Newton, 2006] and included the following words: *left, right, up, down, move, kill* and *pan*. Twenty experimental adult subjects uttered each word 50 times, which led to a database of $20 \times 50 \times 7 = 7000$ words.

First, we present a brief overview of human auditory and speech production systems is presented, and discuss the main differences observed between the speech data collected within the ear canal and the data collected in front of the mouth.

Fundamentally, the simplest form of communication occurs when a speaker releases an acoustic sound pressure wave in the air by using speech articulators such as vocal folds, palate, teeth, tongue, etc. For example, a single-frequency sound wave traveling through air from a speaker’s mouth to a listener’s ears excites a sinusoidal pressure variation in the direction of propagation.

Speech processing is a multi-disciplinary field as hearing is an integral part of the speech chain, and a broad review of sound-wave propagation and hearing mechanism can be found in the white paper, [Sibbald, 2001]. Due to the close connection between the unique method of speech-collection and the human hearing process, it is first necessary to highlight some basic related dynamics of the human auditory system before proceeding to an engineering view of speech production. The frequency response of a healthy human ear is between approximately 20 Hz to 20 kHz, with the high frequency capability diminishing with age. Consequently, the standard CD audio sampling frequency was chosen to be 44.1 kHz, which allows for frequencies up to the usual highest frequency perceivable by the human ear since the sampling frequency should be at least double the highest frequency component available. However, the human ear is most sensitive to a frequency spectrum ranging from 500 Hz to 4000 Hz, which roughly corresponds to the speech bandwidth carried along analog telephone lines [Marsh, 1999]. In addition to this inherent sensitivity, the human ear is capable of a wide dynamic range of hearing intensity, which is practically measured at approximately 130 dB from the threshold of hearing to the threshold of pain. It should be noted that for the purposes of this study, the structures of the outer and middle ear can be considered to be both a pre-amplifier and a limiter that contribute to this remarkable sensitivity and the dynamic range [Sibbald, 2001].

Although the details of the human hearing mechanism is beyond the scope of this thesis, it is important to note that the auditory canal can resonate and amplify sounds within a frequency range of approximately 2000 Hz to 5500 Hz by up to a factor of 10 [Marsh, 1999]. Besides, further amplification – up to 20 times at some resonance frequencies – takes place in the oval window of the cochlea. This knowledge provides a clue why minute pressure variations of external noise are amplified inside the ear. Hence, it was necessary to isolate the in-ear microphone from environmental noise sources by placing it inside a foam case during the data collection process.

The major components of the human speech production system are the lungs, trachea, larynx (organ of voice production), throat, oral cavity and nasal cavity; the last three together being referred to as the *vocal tract* in technical discussions. Finer acoustic organs critical to speech production include vocal cords (or vocal folds), tongue, lips, teeth, velum and jaw, which are called *articulators* by speech scientists. Speech is characterized as a discrete sequence of sound segments called *phones*, each corresponding to certain positions of articulators during their production despite the fact that the analog speech signal is naturally continuous and dynamic in time and magnitude. Phones are acoustic manifestations of *phonemes,* which are considered to be linguistic codes that comprise a language. An interested reader can refer to [Deller, 2000] for a list of approximately 42 phonemes that constitute the language of American English.

From a simple engineering point of view, speech production can be considered an acoustic filtering process in which a speech sound source excites the vocal tract filter [Deng, 2003]. More technically, the larynx provides a periodic vibration to the vocal cords, causing quasi-periodic *voiced speech*. Voiced sounds consist of a *fundamental frequency* and its harmonics produced by vocal cords. Likewise, the fundamental period, or *pitch period* (or only *pitch*), of voiced speech corresponds to the successive vocal cord closure rate [Owens, 1993]. According to Deller, an alternative definition of the term *pitch* refers to the perceived fundamental frequency of a sound, whether or not that sound is actually present in the waveform. Although every speaker has a habitual pitch level, pitch generally shifts up and down during speaking in response to various factors including stress, intonation or emotion. The vocal tract changes the periodic vibration causing *formant* (or resonances) and sometimes *anti-formant* (or anti-resonances) frequencies, owing to the poles and the zeros in the vocal tract transfer function, respectively [Witten, 1982]. Further, the vocal tract length significantly affects all vowel formants frequency locations. Formant locations and pitch period are commonly used characteristics of phonemes to extract information from the speech signal for further analysis.

Unvoiced sounds are generated by forcing air flow through a vocal tract constriction between the glottis and mouth. Unlike voiced speech, there is no fundamental frequency in an excitation signal with unvoiced sounds, resulting in a noise-like and aperiodic signal. Note that unvoiced consonants (such as the weak fricative /f/ and stop sound /t/ as in the word “Left” or the weak plosive burst /p/ as in the word “Pan”) have considerably less energy and are usually focused on higher frequencies, whereas the voiced vowels (such as /u/ as in the word “Up” or /o/ as in the word “Move”) have most of the energy located in the lower portion of the spectrum as a main difference

[Fargues, 2005]. Table 2.1 summarizes the phonemes and the corresponding characteristics of the words used in this study.

Word	**Phoneme**	**Manner of articulation**	**Voiced?**	**Example word**
Left	l	liquid	yes	**l**ove
	є (eh)	vowel	yes	b**e**t
	f	fricative	no	**f**lu**ff**
	t	stop	no	**t**o**t**
Right	r	liquid	yes	**r**oa**r**
	αι (ay)	diphthong	yes	b**i**te
	t	stop	no	**t**o**t**
Up	Λ (ah)	vowel	yes	m**u**d
	p	stop	no	**p**o**p**
Down	d	stop	yes	**d**i**d**
	αu (aw)	diphthong	yes	b**ou**t
	w	glide	yes	**w**o**w**
	n	nasal	yes	**n**o**n**e
Move	m	nasal	yes	**m**ai**m**
	u (uw)	vowel	yes	b**oo**t
	v	fricative	yes	**v**al**v**e
Pan	p	stop	no	**p**o**p**
	æ (ae)	vowel	yes	b**a**t
	n	nasal	yes	**n**o**n**e
Kill	k	stop	no	**k**ic**k**
	i (iy)	vowel	yes	b**ea**t
	l	liquid	yes	**l**ove

Table 2.1. English Phonemes and Characteristics for Vocabulary Words of Interest.

Having given a brief discussion of human auditory system, speech production and the general characteristics of speech signal, it is now possible to focus on the unique properties of the speech data collected via the in-ear microphone. Previous discussions will demonstrate the differences between typical speech signals recorded by conventional directional microphones and those obtained from the in-ear microphone.

Although most of the acoustic sound energy originates from the mouth, some sound waves are emitted from the nostrils, throat and cheeks as well [Deller, 2000]. All these pressure variations in the air constitute the analog speech waveform. This speech waveform is first recorded by the in-ear microphone placed inside the external ear canal and then converted to a digital format by an analog-to-digital converter for further processing. According to Deng, typical human communication spans from below 100 Hz up to 7 kHz due to the limitations of speech production organs [Deng, 2003].

In an effort to compare the two different data-collection methods, Figure 2.1 shows time plots and spectrograms obtained for the word “right” collected with the in-ear microphone placed within the ear canal (top plots) and in front of the mouth (bottom plots). Both the time plot and spectrogram illustrate a portion of 2,000 samples in the signals. Note that both signals sounded very similar except for a small difference in the signal loudness (or amplitude). Figure 2.1 shows that signal collected from the mouth has higher frequency components than that collected within the ear canal. The signal collected from the mouth (the lower waveform) has a comparatively higher frequency content up to 3000 Hz, while the signal collected from the ear has most of its energy located in the spectrum up to 1000 Hz. Similar behavior regarding the frequency ranges was also observed for the other words of interest in the vocabulary. Figure 2.1 also shows the presence of a low frequency distortion resulting from the collection equipment.

It was observed that signals obtained from within the ear canal had low amplitudes and the A/D converter gain control was adjusted accordingly before recording to ensure that speech signals levels were high enough for recording purposes.

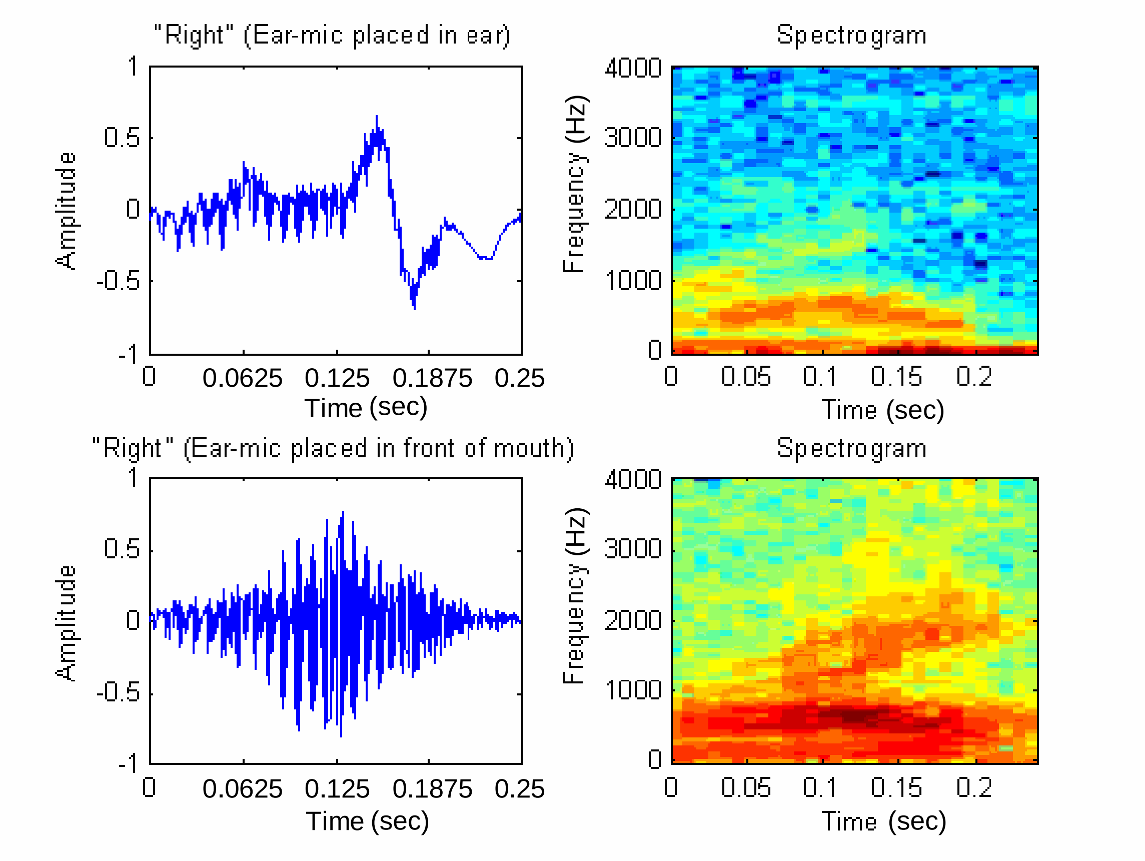


Figure 2.1. The Word "Right" Recorded via Ear-Microphone.

Next, speech pre-processing issues are discussed. Spectral analysis, filtering, framing and windowing for short-term analysis are covered in detail.

B. SPEECH PRE-PROCESSING

1. Spectral Analysis

Spectral analysis basically involves digital filtering techniques to remove the noise and emphasize important frequency components of interest. Considering that the foam-encase of the ear-microphone decreases the amount of external noise in the ear canal during recording, the need for filtering mostly stems from the existence of bodily-created noises such as gulps, tongue clicks, lip smacks or coughs in the signal or due to the collection equipment. In addition, the microphone and the A/D converter used also introduce undesirable side effects, such as a typically 50-60 Hz humming noise, a fluctuating dc bias in the microphone and some other non-linear distortions due to the converter [Picone, 1993].

Before extracting speech feature information from the in-ear microphone data, it is first necessary to deal with the distortions in the signal to eliminate any subsequent problems with the parametric spectral analysis.

a. Spectrogram Analysis

First, the speech signal spectrograms were analyzed to identify the frequency bands most affected by unwanted distortions. Figure 2.3 illustrates the spectrograms obtained for one trial of each word contained in the vocabulary spoken by the same adult male subject.

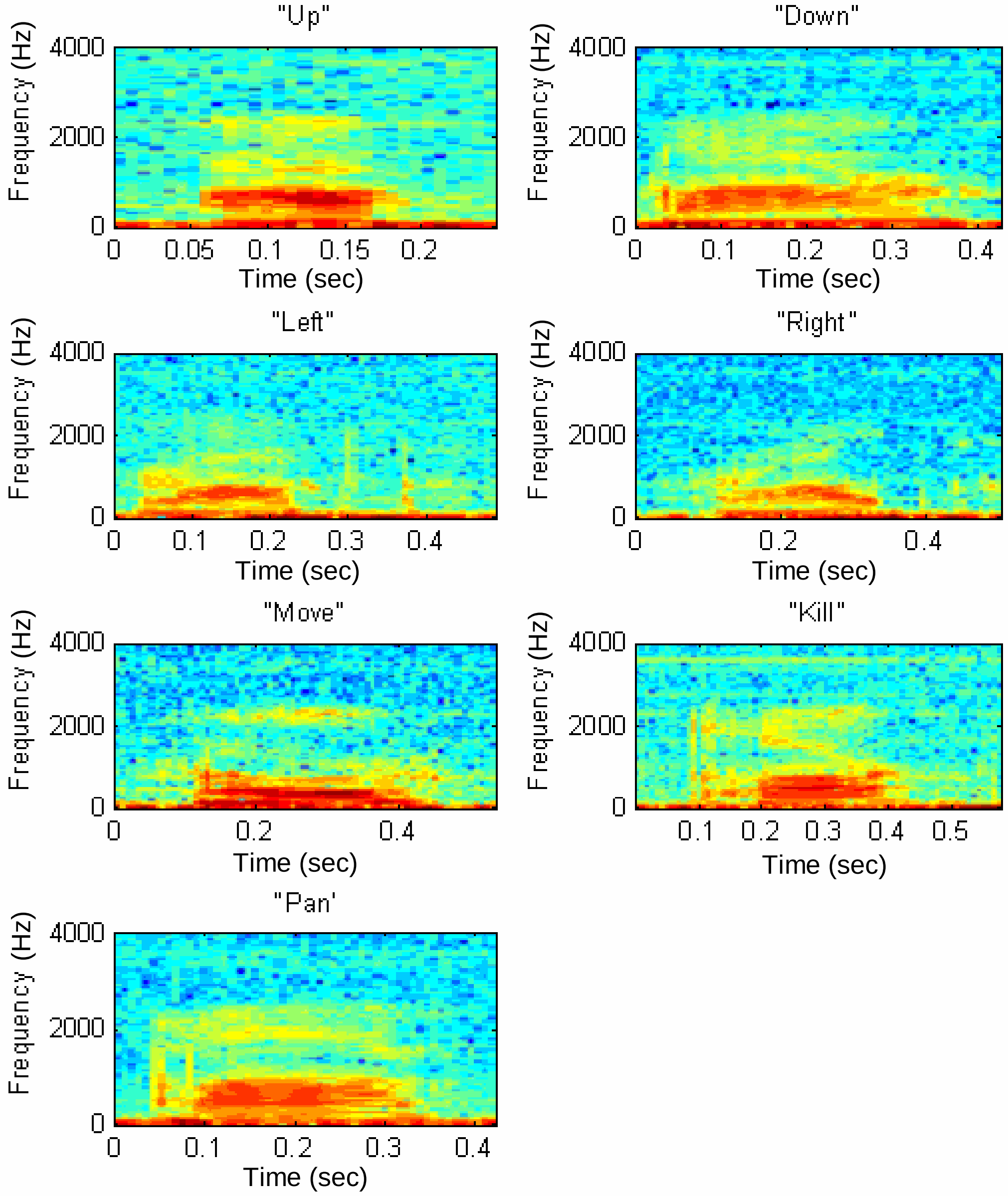
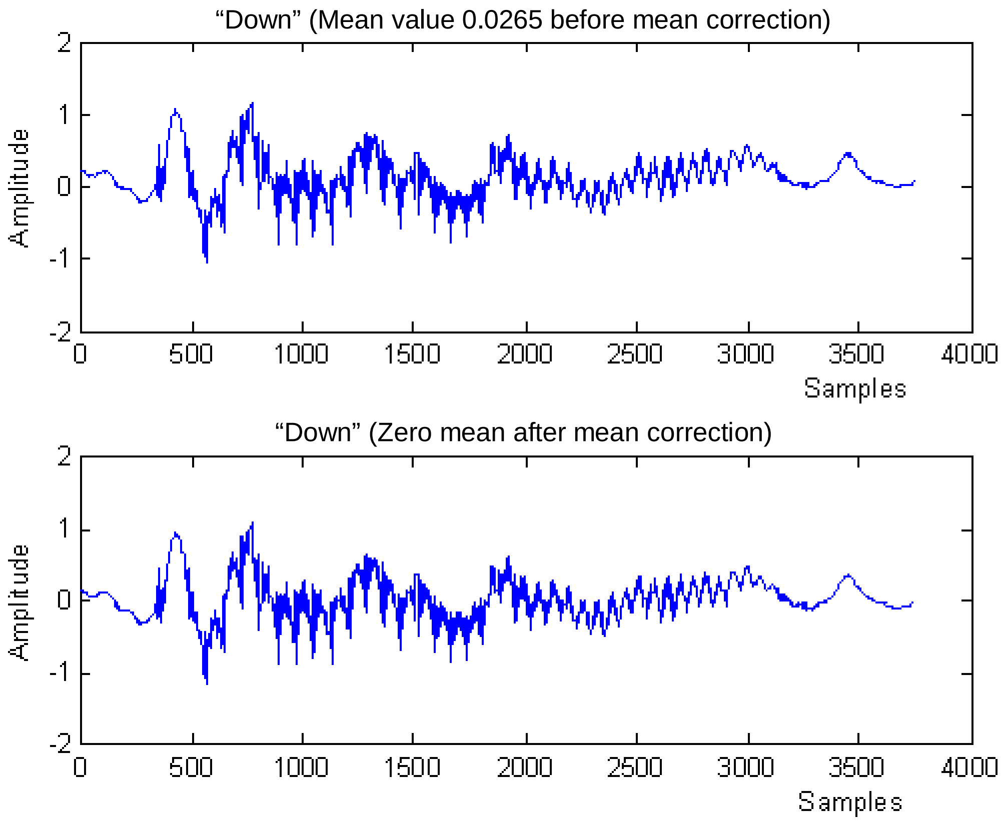


Figure 2.3. Spectrograms for the Utterances of Each Word in the Vocabulary.

Figure 2.3 shows that recordings contain non-speech components in the frequency band between 50 to 100 Hz. It also shows that most of the speech signal energy for all words considered is roughly contained in the frequency band ranging from 100 Hz up to 2000-2500 Hz.

b. Mean Correction

The purpose of mean correction is to remove any dc off-set that may be introduced by the microphone or the converter if any. Figure 2.4 shows the time-domain plot of the word, “down” with a mean value of 0.0265. Although the offset in the signal before mean correction is small, the effect of mean-subtraction can be seen as a shift in abscissa (amplitude) by close investigation.


“Down” (Mean value 0.0265 before mean correction)

“Down” (Zero mean after mean correction)

Figure 2.4. Mean Correction on One Trial of the Word “Down;” Top Plot: before Correction, Bottom Plot: after Correction.

c. High-Pass Filtering vs. Pre-Emphasis Filtering and Band-Pass Filtering

Today’s speech recognition applications require relatively high signal-to-noise ratio (SNR) levels (usually above 30 dB) to insure high recognition performances [Picone, 1993]. Non-linear distortions due to recording devices, A/D conversion and background noise degrade recognition performances. Such adverse effects can be partly removed or decreased by applying a finite impulse response (FIR) high-pass filter.

Usually, a one-coefficient simple digital filter, known as a pre-emphasis filter, is used. There are two common reasons for using this filter. First, voiced portions have a typical attenuation of about -12 dB per octave[1] with increasing frequency. Thus, the pre-emphasis filter serves to compensate such high frequency attenuation and improves the spectral feature extraction [Deng, 2003]. Second, human hearing is more sensitive in the frequency band above 1 kHz. The pre-emphasis filter amplifies this region of the spectrum and contributes in modeling the most perceptually significant portion of the spectrum [Picone, 1993]. A common form of the pre-emphasis filter is given in [Deng, 2003] as follows:

$$y(n) = s(n) - As(n-1), \tag{2.1}$$

where *s(n)* is the speech signal and *A* is typically chosen between 0.9 and 1.0, reflecting the degree of pre-emphasis.

However, recall that Figure 2.3 showed that the ear canal dampened high frequency components as most of the words energy content was contained in the frequency range between 100 Hz to 2500 Hz, which is lower than that observed with the speech collected outside the mouth with the same microphone. Consequently, the pre-emphasis filtering did not contribute to the efficiency of the spectral feature analysis in the experimental results with the MATLAB simulations. Thus, an alternative filtering technique, high-pass filtering, was preferred to mask the low frequency ambient noise. A 6th order infinite impulse response (IIR) elliptic high-pass filter was applied. The filter specifications were as follows:

- The stopband: 0 - 60 Hz.
- The passband: 100 – 4000 Hz (half of the sampling frequency).
- The passband ripple: 0.5 dB.

Figure 2.5 represents the frequency response of the high-pass filter used.

[1] Two frequencies are an octave apart if the ratio of the higher frequency to the lower frequency is two.

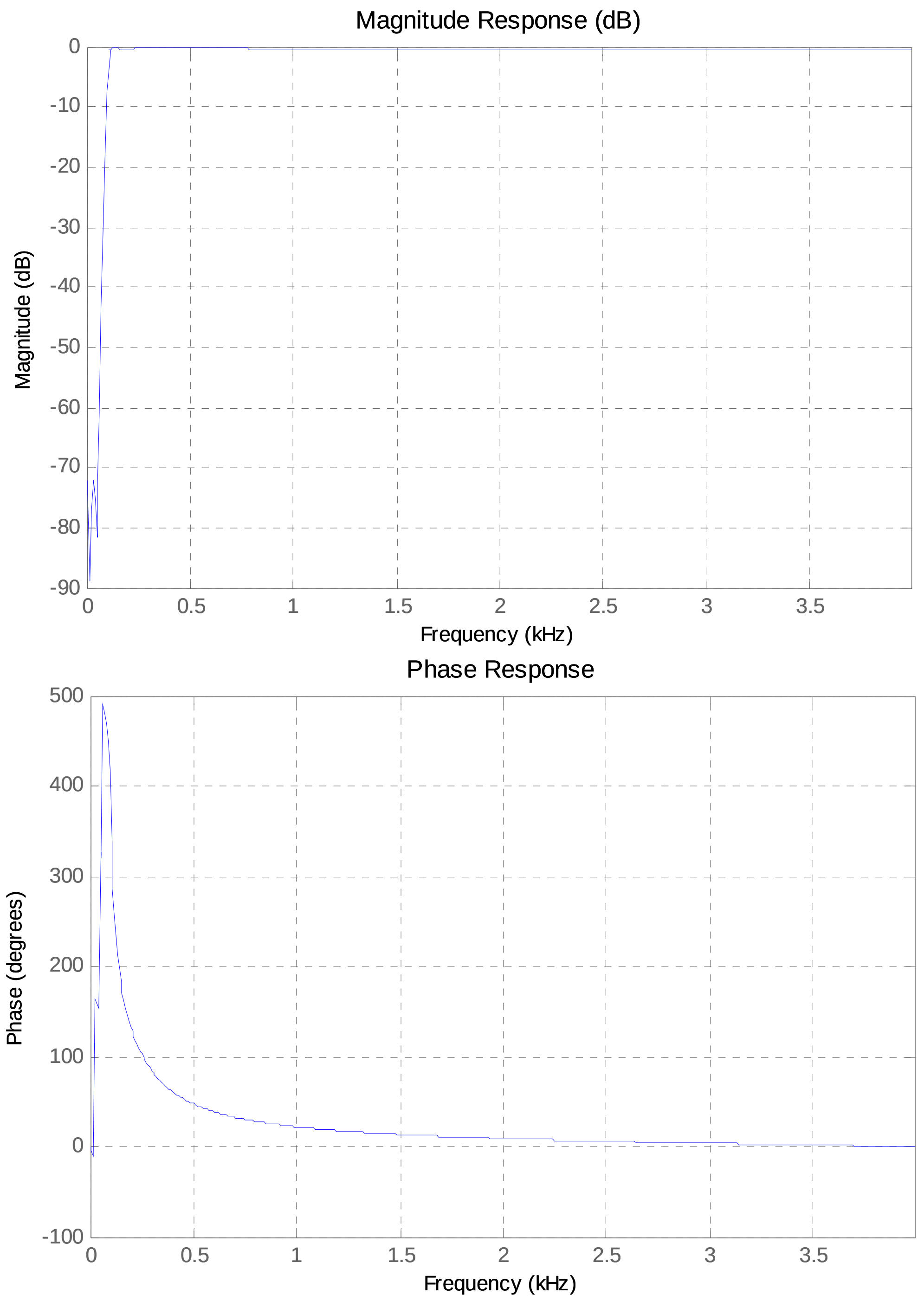


Figure 2.5. Frequency Response of the IIR Elliptic HPF.

Note that the nonlinear phase response may introduce some phase distortion in the filtered speech signal. However, verification was completed by listening to ensure that the filtered signal was still perfectly understandable to the human ear.

Figure 2.5 illustrates the high-pass filter effects on one trial of the word "Down." Upper and lower plots present time and spectrogram traces before and after filtering, respectively. Results show that the filter has removed the low frequency trends observed in the original which elicit the higher frequency components more clearly in the spectrogram plot of the filtered signal.

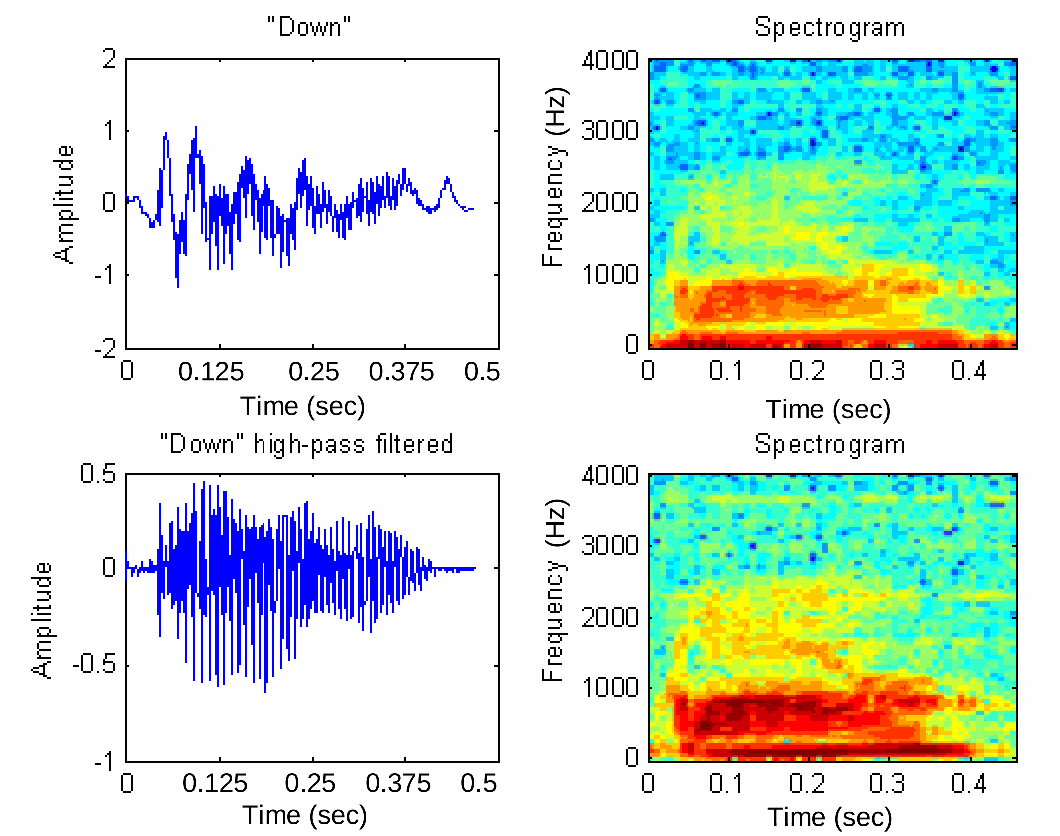


Figure 2.5. High-Pass Filtering Effect on One Trial of the Word "Down."

Experiments were also conducted with a 6th order IIR elliptic band-pass filter for performance comparison with the high-pass filter. This band-pass filter had the following specifications:

• The first stopband:	0 – 60 Hz.
• The passband:	100 – 2000 Hz.
• The second stopband:	2400 – 4000 Hz.
• The stopband attenuation:	60 dB.
• The passband ripple:	0.5 dB.

The high-pass filtered data resulted in, on average, 4% better recognition rates than the recognition rates obtained from the band-pass filtered data.

d. Framing and Windowing for Short-term Analysis

Speech is a dynamic and non-stationary process, as the amplitude of the speech waveform varies with time due to variations in the vocal tract and articulators [Deng, 2003]. However, speech analysis usually presumes that the statistical properties of the non-stationary speech process change relatively slowly over time. Although this assumption is not strictly valid, it makes it possible to process short-time speech frames, ranging typically from 10 ms to 40 ms, as a stationary process. Generally speaking, the use of short frame duration and overlapping frames is chosen to capture the rapid dynamics of the spectrum. Speech parameters are extracted on a frame-by-frame basis and the amount of overlap determines how quickly parameters can change from frame to frame [Picone, 1993].

Windowing means multiplication of a speech signal $s(n)$ by a window $w(n)$ to weight or favor samples by the shape and duration of the window. Coupled with overlapping short-term frames, successive windowing is equal to applying a sliding window to the long-term speech signal. The simplest window has a rectangular shape, weighting all samples of speech signal equally. In fact, not windowing segmented short-duration frames at all is equivalent to applying a rectangular window.

Window duration determines the amount of averaging used in power or energy calculation. Window duration and frame duration can be adjusted as a pair. For instance, a frame duration of 20 ms can be coupled with a window duration of 30 ms [Picone, 1993]. An alternative is to choose the window duration equal to the frame duration for simplicity.

As a matter of fact, the selection of proper frame duration (and indirectly the window duration) is eventually dependent on the change of rate of the vocal tract shape. Frames of 10 ms with 50% overlap are chosen and a rectangular window applied to each frame before calculating short-term parameters such as zero-crossing rate and short-term energy. Note that selecting too small a frame duration or equivalently increasing the frame rate would result in increased complexity and memory requirements.

2. Speech Boundary Detection

Determining the beginning and the termination of speech in the presence of background noise is a complicated problem [Rabiner, 1975].

The problem of separating speech from background silence and noise does not constitute a minor task, except in cases of very high signal-to-noise ratios (on the order of 30 dB or better). Environments with high SNR are rarely seen in real-world applications of speech recognition systems [Rabiner, 1975; Deller, 2000]. Further, a noise-robust endpoint detection algorithm must also be capable of dealing with speaker-dependent disturbances like coughs, gulps, tongue clicks, lip-smacks, etc. [Srydal, 1995].

Also note that the efficiency of accurate endpoint detection has a significant and direct effect on the performance of the entire recognition system. In practice, the process of accurate endpoint detection is not stable and many recognition faults (or misclassifications) can be traced back to poor endpoint detection [Qiang, 1998].

Rabiner and Deller et al. reported the broad categories of low energy phonemes, which usually cause endpoint detection failures [Rabiner, 1975; Deller, 2000]:

- Weak fricatives ({f, th, h}, such as /f/ located in the word, "hal**f**").
- Weak plosive bursts ({p, t, k}, such as /p/, /t/ and /k/ present in the words, "**p**an", "lef**t**" and "**k**ill" respectively).
- Final nasals ({m, n, ng}, such as /n/ present in the words, "pa**n**" and "dow**n**"), and
- Trailing vowels at the end (such as /u/ in "z**oo**").

Ordinarily, zero-crossing rate and short-term energy can be combined to form appropriate spatial or temporal features for determining the onset and termination of speech boundaries [Qiang, 1998]. Note that these temporal features can be extracted simply from the sample values of speech signal without transforming the signal into the frequency domain.

Rabiner et al. proposed a fairly simple and reliable algorithm for detecting the beginning and the termination of speech utterances in low noise environments (i.e., in a SNR level of at least 30 dB), based on short-time energy and a zero-crossing rate of speech [Rabiner, 1975]. This work mainly focused on the implementation of said algorithm in MATLAB.

a. Short-term Energy Measure (STE Measure)

Short-term energy is a common parameter that has been used since the 1970s, especially to distinguish between voiced sounds and unvoiced sounds or silence [Qiang, 1998]. Qiang et al. compared the performances of the following three short-time energy measurements in endpoint detection and concluded that the absolute short-term energy is the most effective energy parameter for this task.

- Logarithmic short-term energy: $E_{log} = \sum_{n=1}^{N} \log\left[s(n)^2 \right],$ (2.2)
- Squared short-term energy: $E_{sqr} = \sum_{n=1}^{N} s(n)^2,$ (2.3)
- Absolute short-term energy: $E_{abs} = \sum_{n=1}^{N} \left| s(n) \right|.$ (2.4)

Consequently, the absolute short-term energy was selected as the short-time energy parameter in this endpoint detection algorithm due to its simple implementation and efficiency. The speech signal was first divided into 50% overlapping frames of 10 ms and then passed through a rectangular window. The short-term absolute energy was computed by summing the absolute magnitudes of speech samples in each frame as shown in Eq. (2.4) above. Therefore, the short-time absolute energy can be computed as follows:

$$E_s = \sum_{n=m-N+1}^{m} \left| s(n) w(m-n) \right|, \tag{2.5}$$

where *w(m)* refers to the rectangular window with the *N*-length frame duration ending at $n = m$. The use of an absolute magnitude function rather than square or logarithm functions simplifies the computations, thereby decreasing the computational load in implementation.

b. Short-term Zero-crossing Rate (STZC Rate)

The number of zero-crossings, which is also a useful temporal feature in speech analysis, refers to the number of times speech samples change sign in a given frame. The rate at which zero-crossings occur is a simple measure of the frequency content of a narrowband signal [Rabiner, 1978]. The short-term zero-crossing measure, as formally defined in [Deller, 2000] for the *N*-length frame interval ending at $n = m$, is

$$ZCR = \frac{1}{N}\sum_{n=m-N+1}^{m} \frac{\left|\operatorname{sgn}\left[s(n)\right] - \operatorname{sgn}\left[s(n-1)\right]\right|}{2} w(m-n). \tag{2.6}$$

The multiplication by a factor of $\frac{1}{N}$ is intended to take the average value of the zero-crossing measure. Therefore, the factor of $\frac{1}{N}$ can be dropped off for simplification in computation.

The zero-crossing rate is a useful parameter for estimating whether speech is voiced or unvoiced [Deng, 2003]. Voiced speech has most of its energy collected in the lower frequencies, whereas most energy of the unvoiced speech is found in the higher frequencies [Rabiner, 1978]. Since speech is wideband, the zero-crossing rate corresponds to the average frequency of the primary energy concentration in the signal. Since high frequencies imply high zero-crossing rates and low frequencies imply low zero-crossing rates, high and low zero-crossing rate correspond to unvoiced and voiced speech, respectively.

c. Endpoint Detection Algorithm from STE and STZC Measures

A popular method for endpoint detection of speech boundaries was first published by Rabiner and Sambur [Rabiner, 1975]. The short-term energy and zero-crossing rate of speech have been extensively used to detect the endpoints of an utterance since then.

This algorithm was adopted and modified to better locate the beginning and the termination of speech from the ear-microphone data. This is a two-step search algorithm where the short-time energy for a coarse search is first used. Then, the zero-crossing measure fine-tunes the coarse boundaries expanding forward and backward. The zero-crossing measure applied in the second search helps detect low-energy phonemes at the beginning or end of the word, especially when dealing with weak fricatives (/f/, /th/, /h/), plosive bursts (/p/, /t/, /k/) or final nasals (/m/, /n/, /ng/). Figure 2.6 shows short-term energy and zero-crossing measures for a typical utterance of the word “left” from ear-microphone data. The mean and the standard deviation of the short-time energy and zero-crossing measures are first computed during the first 50 ms of recording, assuming there is only background noise in that interval. An upper and lower threshold (shown as T_u and T_l on the upper plot of Figure 2.6.) for the short-term energy and another threshold (shown as T_{zc} on the lower plot of Figure 2.6) for the zero-crossing measure are set based on these statistics and experimental findings as follows:

$$T_l = 8 \times MINSTE, \tag{2.7}$$

$$T_u = 32 \times MINSTE, \tag{2.8}$$

$$MINSTE = \min(IE, \text{mean}(STE) + \text{std}(STE)) \tag{2.9}$$

$$IE = 0.25 \tag{2.10}$$

$$T_{zc} = \min(IF, \text{mean}(ZCR) + \text{std}(ZCR)), \tag{2.11}$$

$$IF = 0.25 \times N. \tag{2.12}$$

where N is the frame length; STE and ZCR stand for short-time energy and zero-crossing rate, respectively.

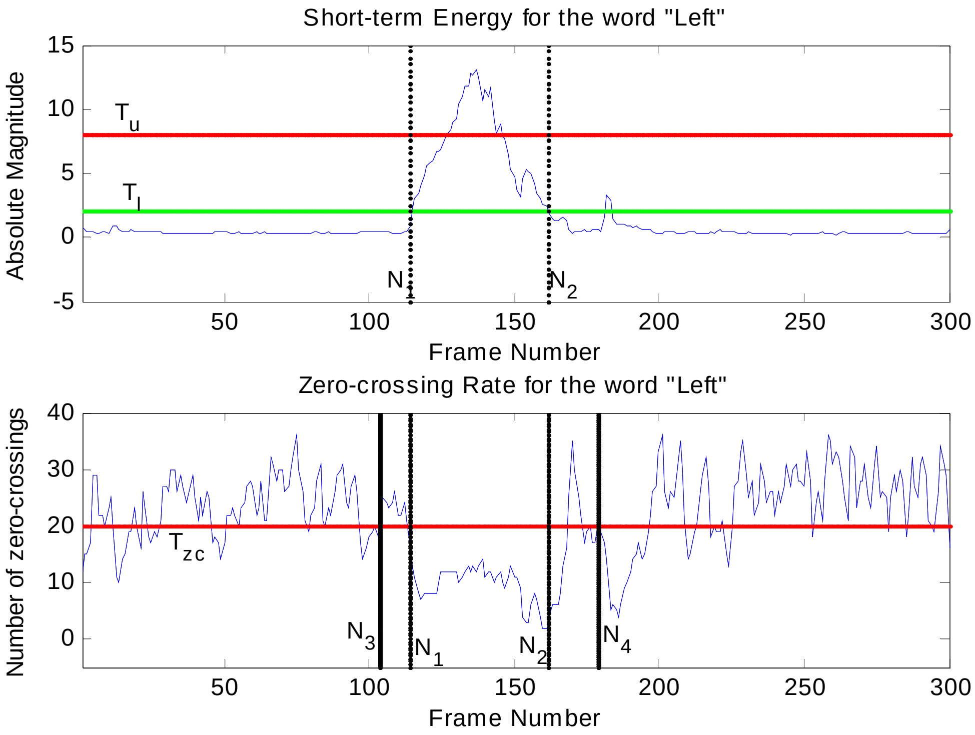


Figure 2.6. Typical Example of the Use of Short-Term Energy and Zero-Crossing Rate in Endpoint Detection.

The short-time energy curve is then searched to find the first and second successive crossings of the upper threshold T_u. Then, it is necessary to move backward and forward from these first and second crossings, respectively, to seek the coarse endpoints (shown as N_1 and N_2), where the lower threshold T_l is first exceeded. This initial search yields the tentative endpoints, N_1 and N_2 shown as dotted lines in Figure 2.6. Next, the fine search on the zero-crossing plot is performed, moving toward the ends from N_1 and N_2 for no more than 10 frames, examining the zero-crossing rate to find three occurrences of counts above the threshold T_{zc}. Last, the final endpoint estimate is moved backward and/or forward to the time of the first threshold crossing if three such occurrences are found. This is the case for N_3 and N_4 shown as solid lines in the lower portion of Figure 2.6. The endpoints remain at the initial estimate N_1 and N_2 when three such occurrences over the zero-crossing threshold are not found. The MATLAB code for this detection algorithm is given in Appendix A.1.

Note that incorrect detection of speech boundaries in isolated word utterances results in potentially two negative effects: increased computations and misclassified recognitions [Ying, 1993]. In addition to the method of short-term energy and zero-crossing rate, experiments were conducted with the modified Teager's energy for comparison.

d. Endpoint Detection from Modified Teager's Energy

Assume a signal sample is shown as $x_i = A\cos\left(\Omega_i + \phi\right)$, where A is the amplitude, Ω is the digital frequency and ϕ is the initial phase of the signal. In Teager's energy algorithm, the instantaneous energy E_i of the sample x_i is computed by [Ying, 1993]:

$$\begin{aligned} E_i &= x_i^2 - x_{i+1}x_{i-1} \\ &= A^2 \sin^2\left(\Omega\right) \\ &\approx A^2\Omega^2. \end{aligned} \tag{2.13}$$

Equation (2.13) shows that both the square of signal amplitude and the square of corresponding frequency component are taken into account in Teager's energy computation. Ying et al. applied the frame-based Teager's energy measure to the endpoint detection problem and developed the following algorithm [Ying, 1993]:

1. Compute the power spectrum for each frame of speech data,
2. Weight each sample in the power spectrum with the square of the frequency,
3. Take the square root of the sum of the weighted power spectrum.

The resulting summation in the last step above produces the energy of one speech frame and this measure is used to determine the endpoints of an utterance.

In this study's experiments, the speech data is split into frames of 20 ms with 50% overlap. The Frame-based Teager Energy (FTE) was computed as stated above. Silence energy is determined from the first 10 frames of speech. The same energy thresholds T_l and T_u from the previous detection algorithm (Rabiner and Sambur algorithm) are calculated using Eqs. (2.7) through (2.10) and a search scheme applied to determine where the FTE measure first exceeds the upper threshold. Finally, endpoint locations are refined by searching along the FTE curve for the location where the lower threshold is first passed down. This method does not include the short-term zero-crossing measure in the search scheme. The final algorithm implemented in MATLAB is included in Appendix A.2.

In both detection algorithms, the assumption was that only noise is detected when the segment identified by this search scheme is less than 100 ms, as the words considered in this study all have a longer duration. Thus, such detections are assumed to be false alarms and dismissed during the search.

Experiments conducted on the same trials with these two detection algorithms produced comparable results for locating the speech boundaries in the given utterances. As a result, *the STE and ZCR detection algorithm* first described in Section (II.B.2.c) was used for the cropping task as it was simpler. Once the words are cropped the next step is to extract the speech features relevant to the data spectral and temporal characteristics. Speech feature extraction and vector quantization steps are discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SPEECH FEATURE EXTRACTION AND VECTOR QUANTIZATION

A. INTRODUCTION

Linear Predictive Coding (LPC) coefficients, Reflection coefficients (RC), Cepstral coefficients, LPC-derived Cepstral coefficients (LPC-CC), Mel-frequency Cepstral coefficients (MFCC) and their variations are commonly used as speech feature parameters.

Linear Predictive Coding (LPC) has been popular for speech compression, synthesis and as well as recognition since its introduction in the 1960s because it offers a reasonable engineering approach for speech signal analysis. Linear prediction models the human vocal tract as an infinite impulse response (IIR) system that produces the speech signal. In other words, LPC makes use of an autoregressive (pole only) model to estimate a short-term spectral envelope for the speech spectrum with an emphasis on the peak spectral values that characterize voiced sounds. The corresponding pole-only transfer function of the spectral model has the following form [Gold, 2000]:

$$H(z) = \frac{1}{1 - \sum_{k=1}^{P} a_k z^{-k}}. \tag{3.1}$$

The linear prediction problem can be stated as finding the coefficients a_k, which minimizes the mean-squared prediction error of the speech signal $s(n)$ in terms of the weighted sum of its past samples [Rabiner, 1993].

For vowel sounds and other voiced portions of speech, which have resonant frequencies and high degree of similarity over short time intervals at their pitch period, linear predictive modeling produces an efficient representation of speech. However, resonant frequencies vary among people. Hence, LPC-based speech features tend to include speaker-dependency into modeling, which degrades the performance particularly for a speaker-independent recognition system. Furthermore, the all-pole constraint of the linear prediction model can cause poor modeling of spectral nulls in noisy environments

[Deller, 2000]. Hence, LPC parameters are not as popular today in speech recognition although they are still widely used for compression [Picone, 1993].

Unlike LPC analysis, *Cepstral Analysis* provides more robust speech features in that they are less dependent on speaker-dependent characteristics and improves recognition in noisy environments. In addition, some weaknesses of LPC parameters can be smoothed by using Cepstral parameters derived from LPC analysis [Davis, 1980; Picone, 1993]. These two reasons mainly explain why cepstrum coefficients started to replace the direct use of LPC coefficients as the premium speech features in Hidden Markov Model (HMM) speech recognizers in the 1980s [Deller, 2000]. The cepstral analysis approach is discussed next.

B. CEPSTRAL ANALYSIS

Among all popular speech parameters, the most functional and efficient ones extract spectral information (in the frequency domain) from speech, because a more concise and easier analysis of speech can be performed spectrally rather than temporarily (in the time domain) [Vergin, 1999]. Although speech signals demonstrate a range of inter-speaker variations for the same utterance in the time domain, this utterance still exhibits consistency in the frequency domain, to some extent. For this reason, spectral analysis is preferred over temporal analysis to discriminate between phonemes and extract speaker-independent features from speech signal.

Cepstral analysis is a special case of homomorphic signal processing. A homomorphic system is defined as a nonlinear system whose output is a linear superposition of the input signals under a nonlinear transformation. Cepstral analysis has become popular in speech recognition since its discovery in the late 1960s, due to the powerful yet simple engineering model of human speech-production behind it [Rabiner, 1978; Picone, 1993]. According to this linear acoustic model, a speech signal is produced by filtering an excitation waveform through the vocal tract filter as depicted in Figure 3.1.

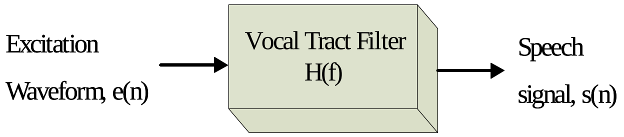


Figure 3.1. Linear Acoustic Model of Human Speech-Production (After [Picone, 1993]).

In this model, the speech signal is expressed as the convolution of an excitation signal $e(n)$ with the vocal tract response $h(n)$. The excitation sequence is either a quasi-periodic vocal cord pulse in the case of producing voiced speech or just random noise at the vocal tract constriction, which generates unvoiced speech [Deng, 2003]. Homomorphic signal processing offers a fairly simple method, known as *cepstral deconvolution,* to decouple the vocal tract response from the excitation response, thereby enabling it to model the vocal tract characteristics better. The decomposition of a speech signal $s(n)$ into the excitation sequence $e(n)$ and the vocal tract function $h(n)$ can be described as follows [Picone, 1993]:

$$s(n) = e(n) \otimes h(n), \tag{3.2}$$

where the operator, “$\otimes$“ represents the convolution operation. Recall that the convolution operation in time corresponds to a multiplication in the frequency domain. Thus, Eq. (3.2) becomes

$$S(f) = E(f) \cdot H(f). \tag{3.3}$$

Note that the complex speech spectrum $S(f)$ is composed of a quickly varying part, excitation spectrum $E(f)$ (which corresponds to high frequency components) and a slowly-varying part, vocal tract response $H(f)$ (which corresponds to low frequency components). Considering that the speech signal is real-valued, the logarithm of Eq. (3.3) on both sides leads to

$$\log(|S(f)|) = \log(|E(f) \cdot H(f)|) = \log(|E(f)|) + \log(|H(f)|). \tag{3.4}$$

Now that the signal components in Eq. (3.4) are linearly combined, a linear filter (also known as *liftering operation* in speech engineering terminology) can be applied to remove the noise-like, quickly-varying excitation part from the speech spectrum. Then, the inverse Fourier transform is applied to the remaining component to compute the *real cepstrum*. In short, under a cepstral transformation, the non-linear convolution of two signals $e(n) \otimes h(n)$ becomes equivalent to the linear sum of the cepstral representations of the signals, $C_e(s) + C_h(s)$. As a result, the real cepstrum is the inverse Fourier transform of the logarithm of the power spectrum of a speech signal [Deng, 2003]:

$$c_s(n) = \frac{1}{N}\sum_{k=0}^{N-1} \log\left|S(k)\right| e^{j2\pi kn/N}, \qquad \text{for } n = 0,1,\ldots,N-1. \tag{3.5}$$

Figure 3.2 shows a block diagram representation of the short-term real cepstrum computation.

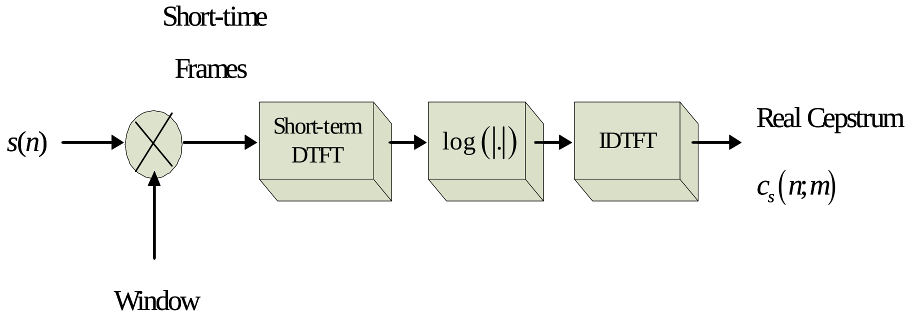


Figure 3.2. Real Cepstrum Computation (After [Deller, 2000].).

Figure 3.3 plots the real cepstrum computed for the voiced phoneme, /ae/ in the word “pan.”

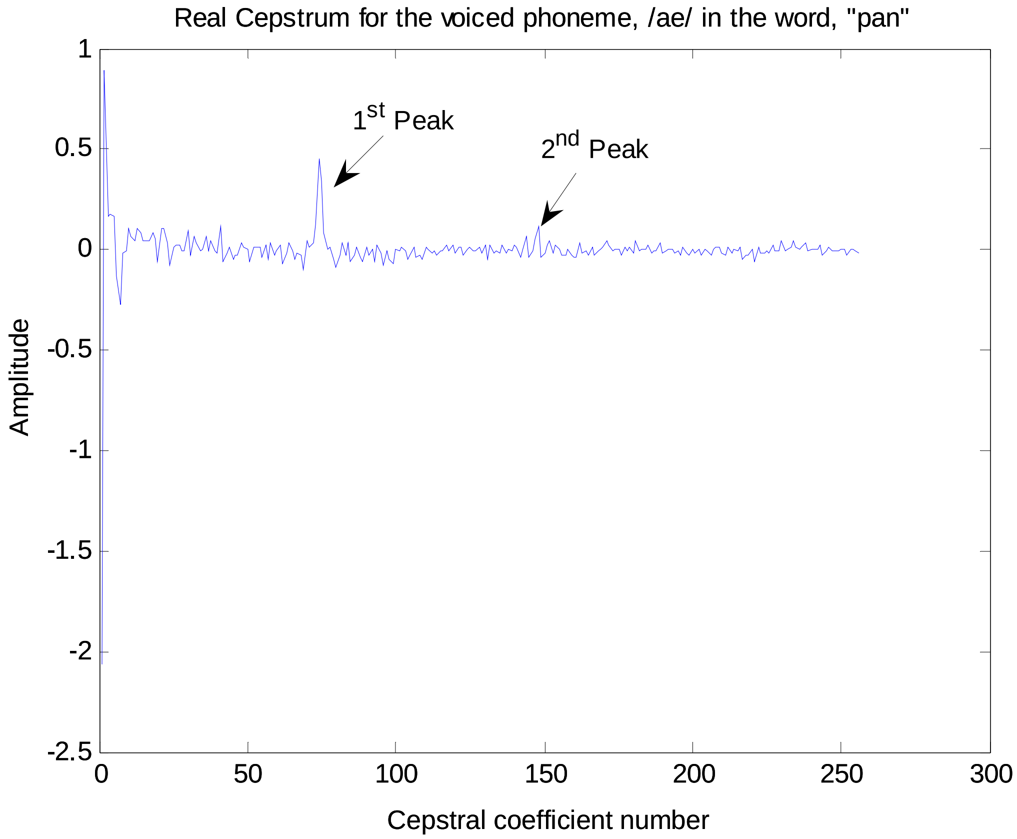


Figure 3.3. Real Cepstrum Computed for the Voiced Phoneme /ae/ in the Word “pan.”

Figure 3.3 illustrates the fact that the dominant contents of the cepstra are located near the origin, and that a small number of cepstrum coefficients can be used to provide enough spectral information about the phoneme /ae/. In addition, pitch period information may be extracted by the spacing between successive cepstral peak locations (in this case, the pitch period is about 75 samples or equivalently 9.375 ms).

Figure 3.4 plots the first 20 coefficients of the real cepstrum computed above, which shows that the first 8-10 coefficients carry most of the spectral information about the voiced phoneme /ae/.

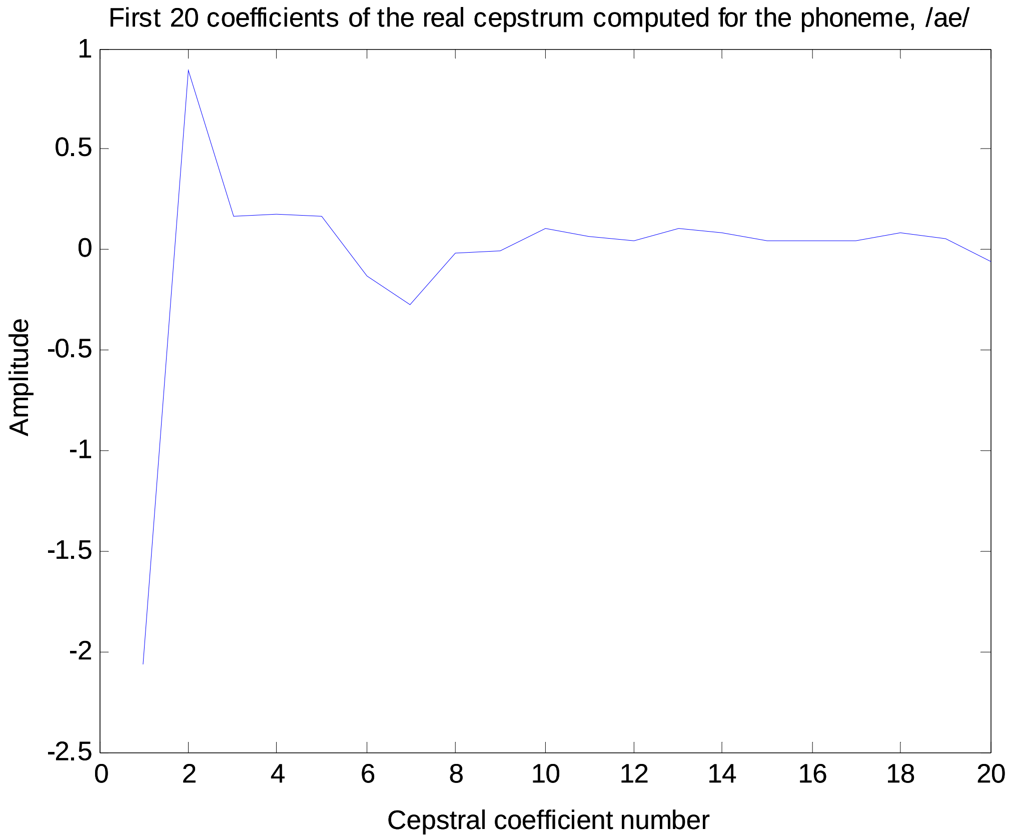


Figure 3.4. The First 20 Coefficients of the Real Cepstrum for the Phoneme /ae/.

1. Mel-Frequency Cepstral Coefficient (MFCC) Computation

In a broad sense, feature extraction aims for data reduction by converting the input signal into a compact set of parameters while preserving spectral and/or temporal characteristics of the speech signal information. Cepstral analysis has been extensively used for feature extraction in speech recognition. Perhaps, the most popular derivation of cepstral analysis combines the cepstrum with a nonlinear frequency-warping, known as mel-scale conversion [Deller, 2000]. The resulting coefficients are called Mel-frequency Cepstral Coefficients (MFCC) and these coefficients were used in this study.

The motive behind the derivation of MFCC's lies in the efforts to mimic human ear operation, and more specifically, the critical bands [Davis, 1980]. First, the spectral resolution of the human auditory system is not linear along the frequency axis [Deng, 2003]. In other words, human sound perception is nonlinear. For example, humans can easily discriminate between closely spaced low frequency tones such as 300 and 350 Hz but not between closely spaced high frequency tones such as 3000 and 3050 Hz. Second, the human ear has some critical bands, meaning that only the stronger of two closely-spaced frequency tones falling into the same critical band will be perceived by the ear. The critical bands of the human ear were experimentally determined by observing the frequency band in which the perception of subjects change abruptly as the frequency of a sound is varied [Deller, 2000]. As a result, the mel-scale conversion is expressed as linear frequency spacing below 1 kHz and a logarithmic spacing above 1 kHz to mimic the spectral characteristics of the human ear. Simply, it maps an acoustic frequency to a perceptual frequency scale as follows [O'Shaughnessy, 1987; Picone, 1993]:

$$F_{Mel} = 2595 \cdot \log_{10}\left(1 + \frac{f(Hz)}{700}\right). \tag{3.6}$$

In analogy, the operation of the inner ear is often viewed as a bank of bandpass filters spaced linearly at low frequencies and logarithmically at high frequencies. Davis et al. introduced the mel-scale critical-band filtering with a simple set of 20 triangular bandpass filters [Davis, 1980]. Davis computed MFCC parameters for monosyllabic word recognition as follows:

$$MFCC_n = \sum_{k=1}^{20} X_k \cos\left[n\left(k - \frac{1}{2}\right)\frac{\pi}{20}\right], \text{ for } n = 0,1,2,\ldots,M, \tag{3.7}$$

where M is the number of MFCC coefficients and X_k, $k = 1,2,\ldots,20,$ represents the log-energy output of the k^{th} filter.

The choice of parametric representations may significantly affect recognition performances in isolated word recognition applications. Note that several feature representations exist for parametrically representing the speech signal in speech recognition. Davis et al. tested and compared the performances of mel-frequency cepstrum coefficients (MFCC), linear frequency cepstrum coefficients (LFCC), linear prediction coefficients (LPC), linear prediction cepstrum coefficients (LPC-CC), and reflection coefficients (RC) for monosyllabic word recognition with a template-based, dynamic time warping recognizer [Davis, 1980]. Their primary conclusion was that the MFCC parameters outperform other parameter types in speech recognition applications with a compact set of coefficients capturing the information relevant for recognition. The basic conclusion of this paper is still considered valid today although there have been many improvements in speech signal representations since the early 1980s [Karnjanadecha, 2001].

According to Davis, MFCCs allow for better suppression of insignificant spectral variation in the higher frequency bands and form a particularly compact representation. Davis also reported that the Euclidean distance metric defined on the cepstrum parameters apparently allows a better separation of phonetically distinct spectra.

Motivated by Davis et al. work, mel-frequency cepstral coefficients and their dynamic derivatives were used for feature extraction from the ear-microphone data during this study. Figure 3.5 shows the block diagram for the MFCC feature extraction step by step. The key difference between MFCC's and the cepstral coefficients lies in the use of a mel-scale filter bank. This study's MATLAB implementation for MFCC feature extraction is a modification of that available in the Voice Toolbox [Brookes, 1997] and is included in Appendix B.1.

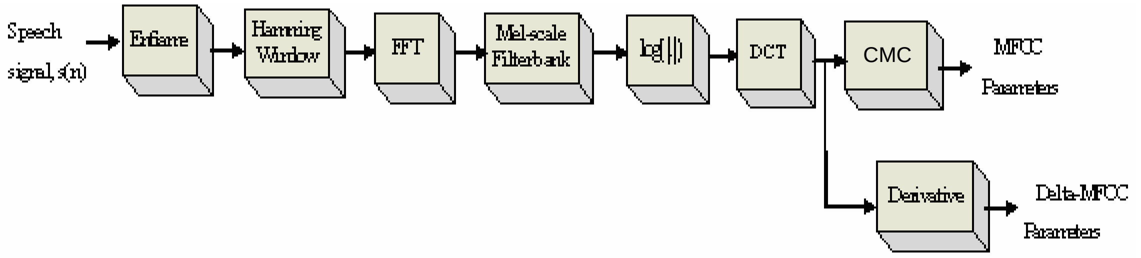


Figure 3.5. Block Diagram for the MFCC Feature Extraction.

The details of the block diagram are described below.

- *Framing*: First, the cropped speech data is blocked into frames of $N = 256$ samples (corresponding to 32 ms) with 40% overlapping to better capture temporal changes from frame to frame.
- *Windowing*: Next, each frame is multiplied by a window function for short-time analysis. Applying a window which tapers down at the ends minimizes the spectral distortion due to discontinuities or abrupt changes at the window endpoints. Typically, a Hamming window is a good choice in speech analysis. Thus, a Hamming window of the form was selected:

$$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\dfrac{2\pi n}{N-1}\right), & n = 0,1,\ldots,N-1. \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

The window length is chosen as a trade-off between frequency resolution and temporal resolution [Becchetti, 1999]. Recall that a short duration window allows for the detection of the amplitude decay of formants with better temporal resolution. However, using a longer window allows for better spectral resolution. Although a longer window is useful for capturing fine spectral variations, it is also in contrast to the requirement for the quasi-stationary analysis of signal segments. As a result, a window length equal to the frame length ($N = 256$) was selected.

- *Fast Fourier Transform (FFT)*: the Fast Fourier Transform is a fast implementation of the Discrete Fourier Transform (DFT) which converts *N*-samples of frames into frequency spectrum.
- *Mel-frequency Warping*: The mel-scale filter bank implementation used in this study includes 24 triangular filters non-uniformly spaced along the frequency axis, as shown in Figure 3.6.
- *Log Energy Computation*: This step applies a logarithm transformation to the absolute magnitude of the coefficients obtained after mel-scale conversion. The absolute magnitude operation discards the phase information while the logarithm operation performs a dynamic compression, making feature extraction less sensitive to speaker-dependent variations. [Becchetti, 1999].
- *Mel-frequency Cepstrum Computation*: Mel-frequency cepstral coefficients are computed by applying the inverse FFT to the logarithm of the magnitude of the filter bank outputs. Note that the inverse DFT reduces to a Discrete Cosine Transform (DCT) operation as the log magnitude spectral of the coefficients are real and symmetric [Becchetti, 1999]. Moreover, the DCT has the advantage of producing highly uncorrelated features [Jayant, 1984; Deng 2003]. The resulting mel-frequency cepstral coefficients are computed as follows:

$$c(k) = w(k)\sum_{n=1}^{N} \log\left(\left|x_k(n)\right|\right)\cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right), \quad k = 1,\ldots,L, \tag{3.9}$$

where

$$w(k) = \begin{cases} \dfrac{1}{\sqrt{N}}, & k = 1. \\ \sqrt{\dfrac{2}{N}}, & 2 \le k \le L. \end{cases} \tag{3.10}$$

$$N = 256 \text{ (window length).} \tag{3.11}$$

$$L = 24 \text{ (Number of filters in the mel-scale filter bank)}, \tag{3.12}$$

and $x_k(n)$ represents the output of the k^{th} filter in the filter band.

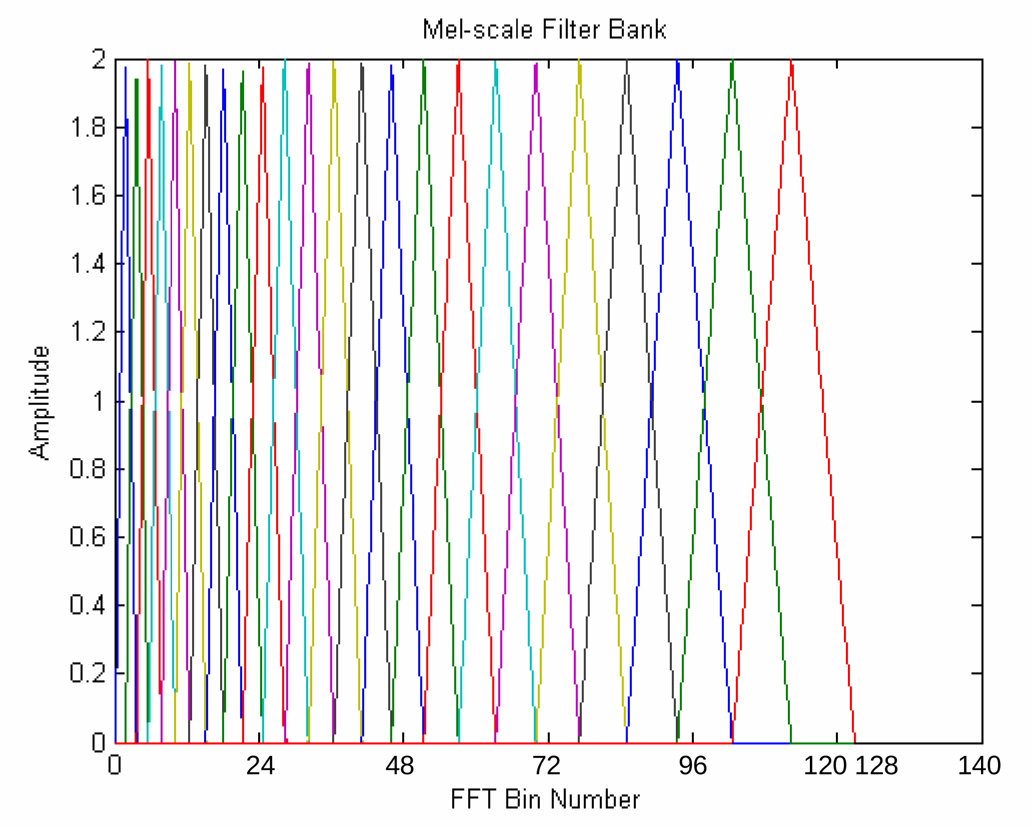


Figure 3.6. Mel-scale Filter Band

The zero-order MFCC coefficient c_0 represents the average log energy of the frame. This coefficient is usually discarded from the feature space because absolute power measures are not reliable in speech recognition [Picone, 1993; Becchetti, 1999]. Experimental results demonstrate that 12 coefficients were sufficient to represent the speech data under investigation. Therefore, the zero-order coefficient was excluded and the remaining first 12 MFCC coefficients kept for a compact representation.

- *Cepstral Mean Correction (CMC)*: It is inevitable that a speech signal is subject to some spectral distortions during recording and pre-processing prior to recognition due to microphone offsets, environmental effects (like reverberation) and/or transmission channel [Becchetti, 1999]. The cepstral mean correction was applied to compensate for such distortion by subtracting the cepstral mean of a frame from the cepstral coefficients for additional robustness in recognition.

- *Delta-MFCC Computation*: Feature vectors consisting of only MFCC parameters appear to provide smooth estimates of the local spectrum.

However, these features do not contain information regarding the speech signal dynamic evolution, which also carries relevant information in speech recognition [Becchetti, 1999]. Therefore, improvements in recognition performance can be obtained by taking into account the dynamic characteristics of the MFCC features [Deng, 2003; Furui, 1992]. The simplest approach to obtain these dynamic features takes the basic difference of coefficients between consecutive frames. The resulting coefficients, known as *dynamic* or *delta parameters*, reflect cepstral changes over time. Furui et al. reported significant improvements in recognition due to the addition of the differential features to the instantaneous coefficients [Furui, 1986]. However, researchers have also argued that the basic differencing operation is too sensitive to random inter-frame variations and should be replaced by a smoother estimate of the local time-derivative [Furui, 1986; Deller, 2000; Deng, 2003]. Consequently, most speech recognition systems today incorporate delta features as an add-on to the static parameters such as MFCCs [Gold, 2000]. For these reasons, 12 delta-MFCC parameters were included in this study's set of speech features in addition to the regular MFCC parameters. The delta-MFCC coefficients are computed using the following regression formula:

$$d_k = \frac{\sum_{\alpha=1}^{M} \alpha\left(c_{k+\alpha} - c_{k-\alpha}\right)}{2\sum_{\alpha=1}^{M} \alpha^2}, \tag{3.13}$$

where the configuration parameter for the difference length is $M = 4$ and d_k represents the delta coefficient computed in terms of the corresponding static coefficients $c_{k+\alpha}$ to $c_{k-\alpha}$.

The regression formula shown in Eq. (3.13) is equivalent to passing the static MFCC coefficients through a linear differential filter with impulse and magnitude response are shown in Figure 3.7. One potential problem with the use of delta MFCC parameters is that they are not in the same range of amplitude with the static MFCC parameters, which results in poor vector quantization. Through experiments, it was found that the delta parameters approach the range of static parameters when they are weighted by a factor of six.

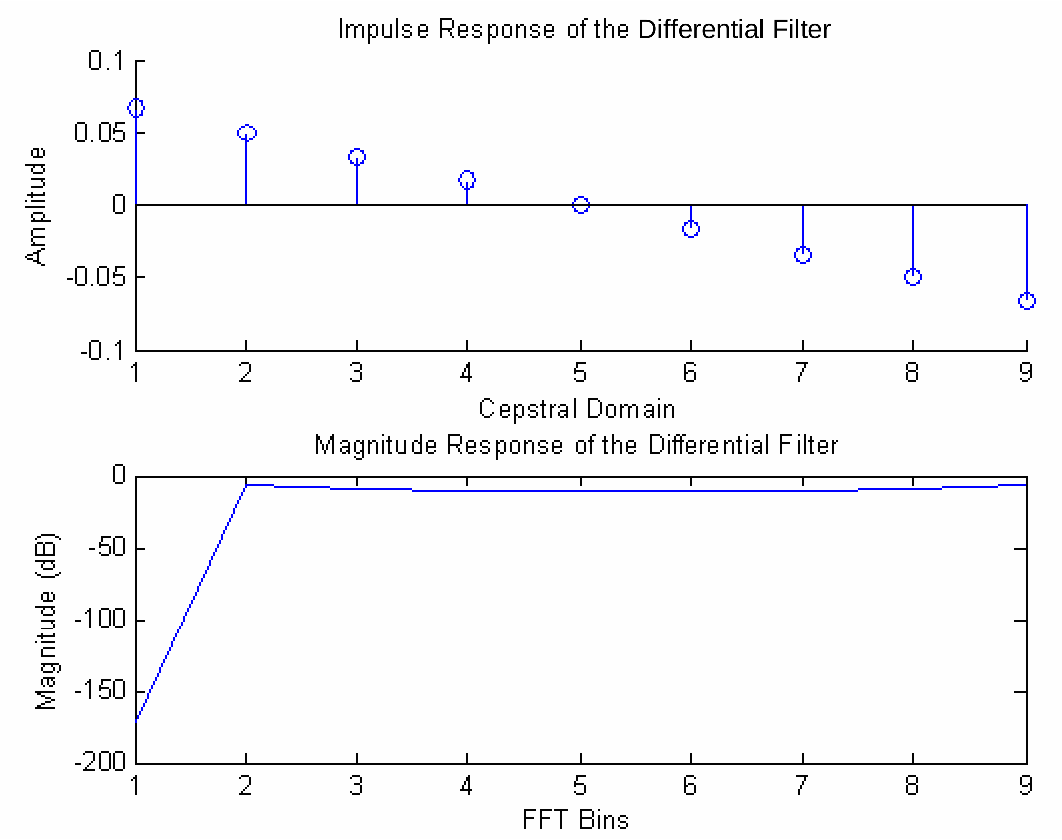


Figure 3.7. Differential Filter Used to Compute Delta-MFCC Parameters.

Figure 3.8 shows a typical example of the resulting 12 MFCC and 12 weighted delta-MFCC parameters extracted from the ear-microphone data for the spoken word "pan." Figure 3.9 plots the cepstrogram for the same 24 parameters.

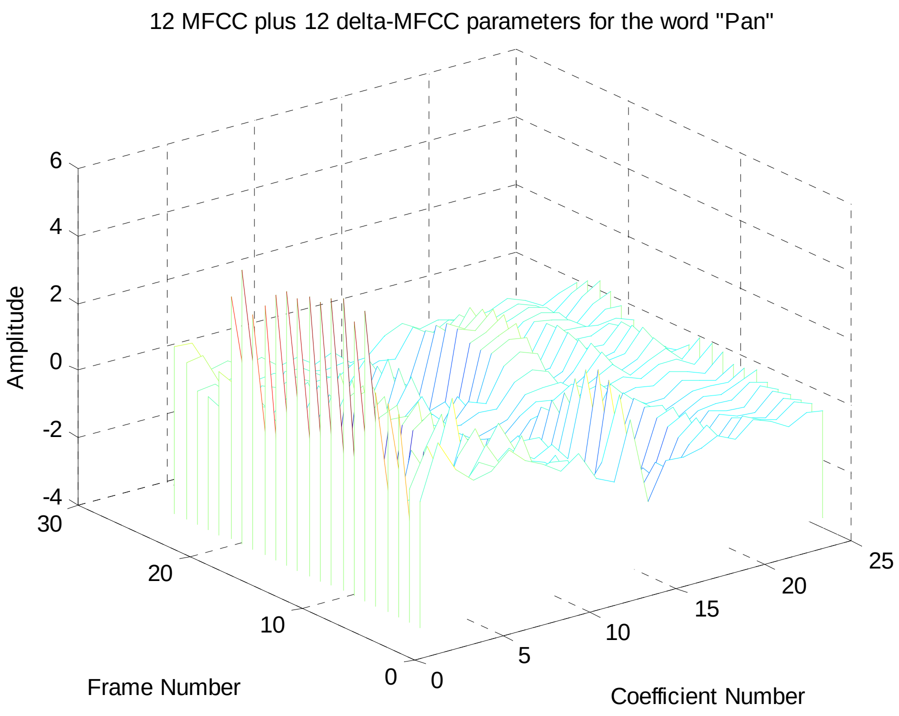


Figure 3.8. MFCC and Delta MFCC Parameters Extracted from the Ear-Microphone Data of the Spoken Word "Pan."

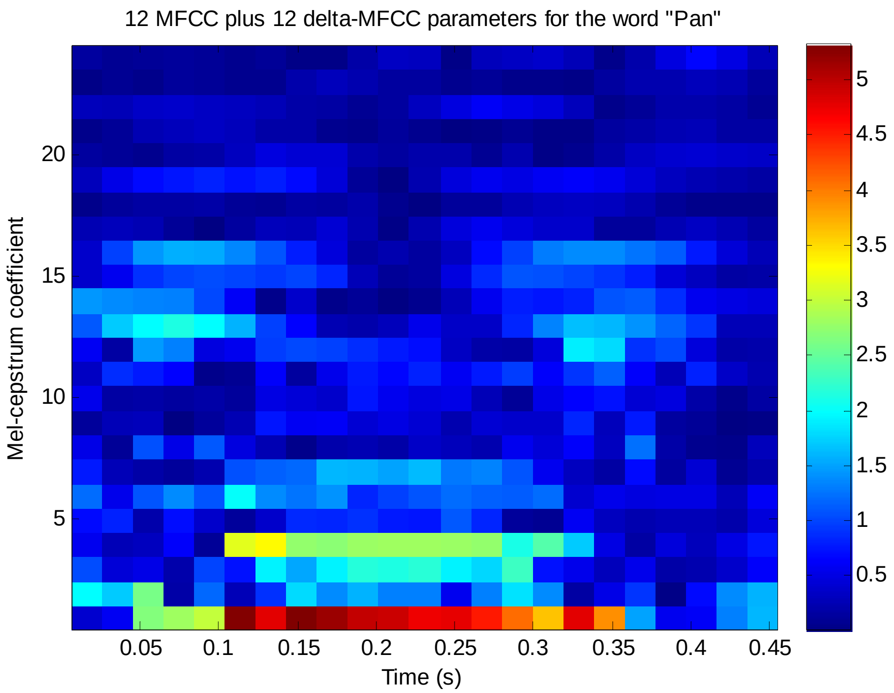


Figure 3.9. Cepstrogram Plot of the MFCC and Delta MFCC Parameters Extracted from the Word "Pan."

2. LPC-derived Cepstral Coefficients

Recall that LPC model can be used to compute a smoothed version of cepstral coefficients, known as LPC-derived Cepstral Coefficients (LPC-CC). In addition to MFCC parameters, LPC-CC parameters were tested for performance comparison in recognition. First, LPC coefficients were computed by using the auto-correlation method and Levinson-Durbin recursion. The use of the auto-correlation method guarantees that the LPC model is inherently stable [Picone, 1993]. Second, the LPC parameters are converted to 12 cepstral parameters by using the following relation [Deller, 2000]:

$$\gamma(n;m) = a(n;m) + \sum_{k=1}^{n-1} \left(\frac{k}{n}\right) \gamma(k;m)\, a(n-k;m), \quad n = 1, 2, \ldots M, \tag{3.14}$$

where $a(n;m)$ represents the LP coefficients and M is the order of the LPC model. Next, these cepstral coefficients were weighted by a window function (the raised sine lifter) $w_\gamma(n)$, which tapers at both ends. Equations (3.15) and (3.16) show this window function $w_\gamma(n)$ and the resulting weighted cepstral coefficients $\gamma_w(n;m)$, respectively:

$$w_\gamma(n) = 1 + \frac{M}{2}\sin\left(\frac{\pi n}{M}\right), \tag{3.15}$$

$$\gamma_w(n;m) = \gamma(n;m)\, w_\gamma(n). \tag{3.16}$$

Finally, 12 delta LPC-CC parameters were computed by using the following difference function:

$$\Delta\gamma_w(n;m) = \frac{1}{2}\left(\gamma_w(n;m+1) - \gamma_w(n;m-1)\right), \quad n = 1, 2, \ldots, M, \tag{3.17}$$

and included in the feature vector as done previously with the MFCC parameters. The MATLAB implementation used for LPC-CC feature extraction was that available at [Sorensen, 2005] and is included in Appendix B.2. Experimental results showed that the set of MFCC parameters outperformed the LPC-CC parameters in isolated word recognition by 7%, on average.

The next section discusses vector quantization and codebook generation.

C. VECTOR QUANTIZATION AND CODEBOOK GENERATION

Pattern recognition provides the technical foundation that underlies vector quantization techniques in speech recognition [Gold, 2000]. The goal of pattern recognition is to classify objects of interest into a number of classes (or clusters). In this case, the objects of interest are sequences of feature vectors extracted from the ear-microphone data using the techniques described in the previous section.

A *supervised pattern recognition* system is trained with labeled samples; that is, each input pattern has a class label associated with it. Pattern recognition can also be performed in an *unsupervised* fashion when information about the class membership of the training vectors is not provided. In the case of speech recognition, the problem of automatically separating training data into classes is often solved by a *clustering algorithm* in an unsupervised fashion [Deller, 2000]. For example, vector quantization is such a technique in which speech features vectors are clustered around some centroid locations. The resulting cluster centers constitute a *codebook*. Then, the codebook can be used to index a new feature vector by finding the cluster center (centroid) that is closest in some norm to the new vector.

1. Vector Quantization (VQ)

Quantization converts a continuous-amplitude signal to one of a set of discrete-amplitude signals, which is different from the original signal by the quantization error. The independent quantization of each signal parameter separately is termed *scalar quantization,* while the joint quantization of a vector is termed *vector quantization* (VQ) [Makhoul, 1985]. Vector quantization can be thought of as a process of redundancy removal that makes the effective use of nonlinear dependency and dimensionality by compression of speech spectral parameters. Generally, the use of vector quantization results in a lower distortion than the use of scalar quantization at the same rate [Sayood, 2000].

Vector quantization has been significantly popular in speech coding and recognition after the introduction of LPC [Makhoul, 1985]. The VQ problem can be summarized simply in the following manner. Suppose that $x = \left[x^{(1)}, x^{(2)}, ..., x^{(L)} \right]^T$ is an L-dimensional column vector whose components $\left\{ x^{(i)}, 1 \le i \le L \right\}$ are real-valued, continuous-amplitude random variables. In vector quantization, the vector x is mapped to another real-valued, discrete-amplitude, *L*-dimensional column vector, y. Thus, the quantization function is of the form

$$y = Q(x). \tag{3.18}$$

Typically, y takes on one of K distinct values such that $Y = \left\{ y_j^{(i)},\ 1 \le j \le K \right\}$, where y_j represents a vector of the form $y_j = \left[y_j^{(1)}, y_j^{(2)}, ..., y_j^{(L)} \right]^T$. The set Y is referred to as the *codebook*, and each y_j is a *code vector*. To generate such a codebook, the *L*-dimensional space S_L of the speech feature vector x is partitioned into K mutually exclusive regions or clusters such that $\left\{ C_j, 1 \le j \le K \right\}$, and a code vector y_j is associated with each cluster C_j. The vector quantizer then assigns the code vector y_j to the feature vector x_i if the feature vector x_i is in the cluster C_j, i.e.,

$$Q\left(x\right) = y_j, \qquad \text{if } x \in C_j. \tag{3.19}$$

For $L = 1$, vector quantization reduces to scalar quantization. Figure 3.10 illustrates a typical example of uniform clustering of two-dimensional space $\left(L = 2\right)$ for vector quantization. This illustration is a uniform clustering in that the two-dimensional space is partitioned into four clusters equally. Unfortunately, speech data are not generally this well-behaved in terms of uniform clustering and low-dimensionality [Gold, 2000].

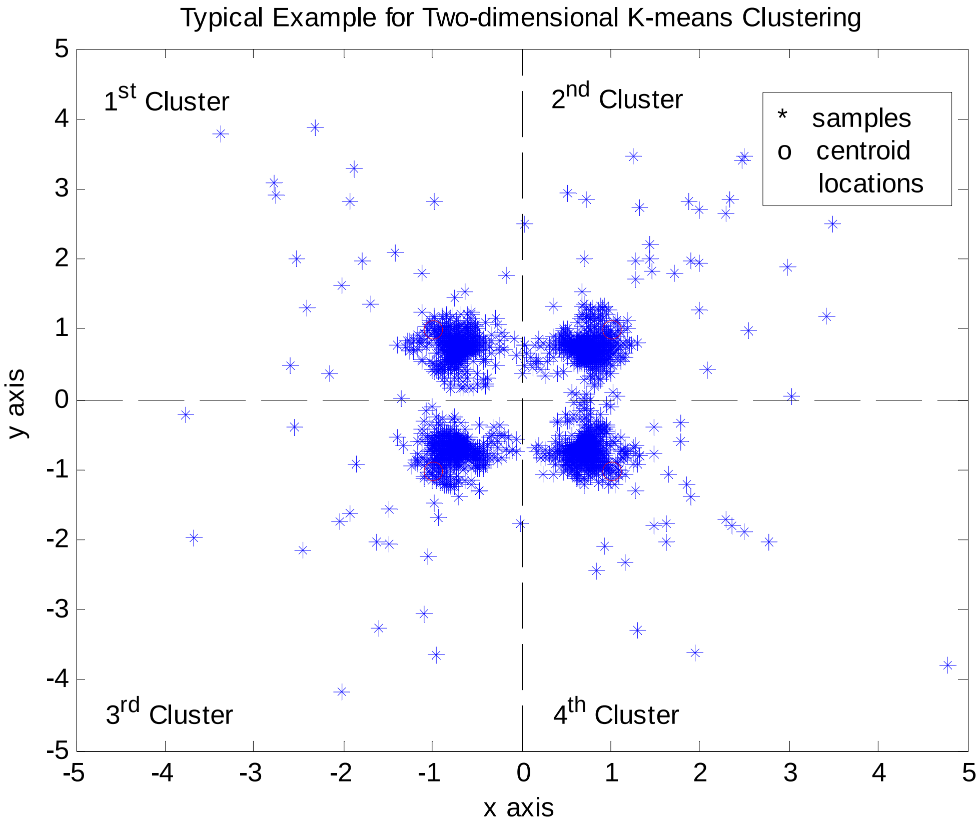


Figure 3.10. A Typical Example of Uniform Clustering of Two-Dimensional Space for Vector Quantization.

2. Distortion Measure

When x is quantized as y_j, a quantization error results and a distortion measure (or a distance measure) $d(x, y_j)$ can be defined between feature vector x and the code vector (or codeword) y_j.

The squared Euclidian distance is one of the most popular distortion measures in speech recognition applications. The quantized code vector is selected to be the closest in Euclidean distance from the input speech feature vector. The *Euclidean distance* is defined by:

$$d\left(x, y_j\right) = \sqrt{\sum_{i=1}^{L}\left(x^{(i)} - y_j^{(i)}\right)^2}, \tag{3.20}$$

where $x^{(i)}$ is the i^{th} component of the input speech feature vector, and $y_j^{(i)}$ is the i^{th} component of the codeword y_j. A vector quantizer is said to be *optimal* if it meets the following two necessary conditions.

- *Nearest Neighbor Condition*: The optimal quantizer is realized by using a minimum distortion or *nearest neighbor condition* such that

$$Q(x) = y_j, \text{ iff } d(x, y_j) \le d(x, y_k) \text{ for } j \ne k \text{ and } \forall k = 1, 2, ..., K. \tag{3.21}$$

  Eq. (3.21) states that the vector quantizer selects the code vector that results in the minimum distortion with respect to x. Thus, the cluster C_j should consist of all feature vectors that are closer to y_j than any of the other code vectors such that:

$$C_j = \left\{ x : \left\| x - y_j \right\|^2 \le \left\| x - y_k \right\|^2 \;\; \forall k = 1, 2, ..., K. \right\} \tag{3.22}$$

- *Centroid Condition*: Each vector y_j is chosen to minimize the average distortion in cluster C_j. Such a vector is called the *centroid* of the cluster C_j, which is equivalent to the statistical mean of the cluster members. Thus, the *centroid* of the cluster C_j is given by:

$$y_j = centroid\left(C_j\right) = \frac{\sum_{x_m \in C_j} x_m}{\sum_{x_m \in C_j} 1} \quad \text{for} \quad j = 1, 2, ..., K. \tag{3.23}$$

A popular method for codebook design is an iterative clustering algorithm known as the *K-means algorithm*. This algorithm divides the set of training feature vectors into *K* clusters C_j in such a way that the two necessary conditions for optimality are satisfied. The *K*-means algorithm is described next.

3. *K*-Means Algorithm

The *K*-means algorithm is widely used in speech processing as a dynamic clustering approach [Deller, 2000]. “*K*” is pre-selected and simply refers to the number of desired clusters. Linde, Buzo and Gray et al. were first to suggest the use of the *K*-means algorithm for vector quantization with a large class of distortion measures [Linde, 1980]. Consequently, the *K*-means algorithm has also been named after Linde, Buzo and Gray as the *generalized LBG algorithm* in speech processing literature.

The LBG algorithm was chosen to execute the vector quantization step for this research. The objective of the LBG algorithm is to define the regions C_j and their representatives y_j so that the overall distortion is minimized. One way to compute the code vectors of the training set is to start with an arbitrary random initial estimate of the code vectors and to apply the nearest neighbor condition and the centroid condition iteratively, until a termination criterion is satisfied [Theodoridis, 2003].

Thus, a set of *K* code vectors is discovered into which all speech feature vectors in the training set can be quantized with minimum distortion. This set of code vectors represents a codebook for the feature space. Figure 3.11 shows the flow diagram for the *K*-means clustering using the generalized LBG algorithm.

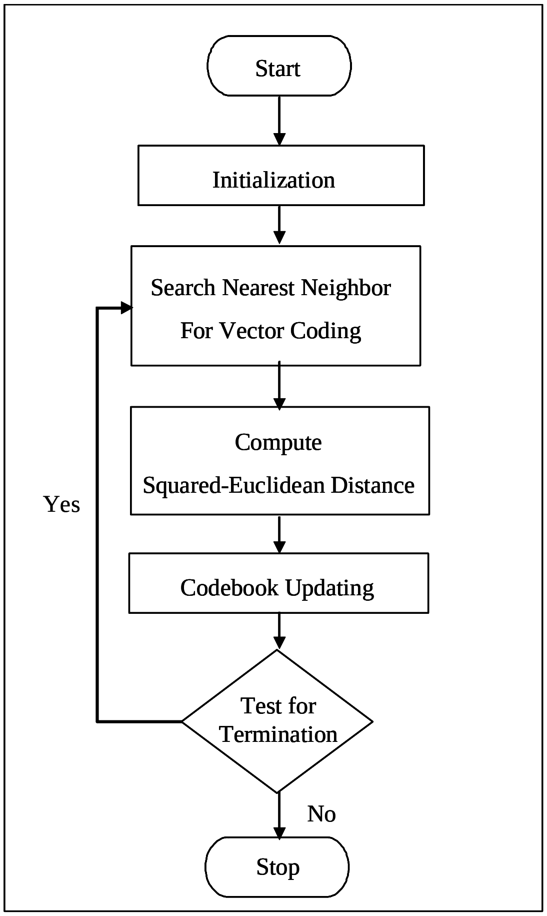

Start
Initialization
Search Nearest Neighbor For Vector Coding
Compute Squared-Euclidean Distance
Yes
Codebook Updating
Test for Termination
No
Stop

Figure 3.11. Flow Diagram for the *K*-means LBG Clustering Algorithm.

The operation of the generalized LBG algorithm is summarized as follows. [Deller, 2000].

- *Initialization.* Choose an arbitrary (random) set of *K* code vectors, say y_j, $j = 1, 2, ..., K$, where y_j represents the initial centroid (or initial mean value) of the feature vectors in cluster C_j.
- *Iteration.*

  1. Apply the nearest neighbor rule for each feature vector x in the training set to quantize x into code vector y_j, where

$$j = \arg\min_i \left[d(x, y_i) \right], \quad i = 1, 2, ...K. \tag{3.24}$$

  Here $d(\cdot,\cdot)$ represents the squared Euclidean distance in the feature space.

  2. Compute the total distortion that has occurred as a result of this quantization in terms of the squared Euclidean distance,

$$D = \sum d\left[x, Q(x) \right], \tag{3.25}$$

  where the sum is taken over all vectors x in the training set and $Q(x)$ indicates the code vector y_j to which x is assigned in the current iteration.

  3. Apply the centroid condition for updating the codebook. For each cluster, compute the centroid of all vectors x such that $y_j = centroid\left(C_j\right) = Q(x)$ during the current iteration. Update the codebook based on the new set of centroids and return to Step 1.

- *Termination*: Iterations in step 1 through 3 repeat until a specific termination criterion is met. Termination criterion might be determined in various ways, such as reaching a maximum iteration number or having a sufficiently small squared Euclidean distance. Specifically, the goal is to terminate the iteration when the difference between the updated code vectors and the previous code vectors is sufficiently small.

Like many other numerical minimization algorithms, the *K*-means algorithm can converge to a local optimum instead of a global optimum based on the starting point (randomly-selected initial codebook). The distortion measure is likely to converge towards the local minimum closest to the initial codebook. For this reason, the algorithm should be executed several times with different initial codebooks to achieve some type global optimality. Then, it is possible to choose the optimum codebook, which results in a minimum overall distortion.

4. Codebook Generation from Speech Features

Since the discrete observation Hidden Markov Models (HMM) classifier will be used to recognize the words of interest, this study is restricted to the production of discrete observation symbols as input to the classifier. In other words, the naturally occurring continuous-valued observation vectors should be quantized into indices of a codebook before classification.

Next, the utilization of vector quantization in this study is detailed. A training matrix of size $(F \text{ by } T)$ is formed by concatenating the first 12 MFCC and 12 delta-MFCC parameters extracted from all the training utterances. Each column of this training matrix represents F speech features of a given frame and T is the number of all frames. Given the size of the codebook K, this randomly-selected training matrix is used to generate the codebook representing the acoustic information of all the words of interest. To train the codebook properly, the training feature vectors should span the anticipated range of acoustic information used for recognition. Two thirds of the overall data is randomly chosen to ensure that the training data for codebook generation is sufficiently large. In addition, it is also necessary to choose K random code vectors as initial conditions.

Each feature vector in the training matrix is then clustered based on the squared Euclidean distance between a training vector and each of the code vectors in the codebook. Once all training vectors have been labeled with the corresponding code vectors, the total distortion is computed by calculating the squared Euclidean distance. Summation of all these distances represents the total quantization error of the codebook.

The mean of each cluster's members is computed to find the centroid location and each centroid location replaces the corresponding old code vector. The absolute difference between the updated code vectors and the previous code vectors determines the number of repetitions required before terminating the algorithm.

At this point, an important issue is to decide upon the number of required code vectors (centroids) in the codebook for a quality representation of speech features. One measure of the quality of a codebook is the average distance of a feature vector in the training data from its corresponding code vector. This figure is often called the *codebook* *distortion*. Obviously, the smaller the distortion, the more accurately the centroids represent the vectors they replace. A plot of the average distortion versus codebook size for this research is shown in Figure 3.12.

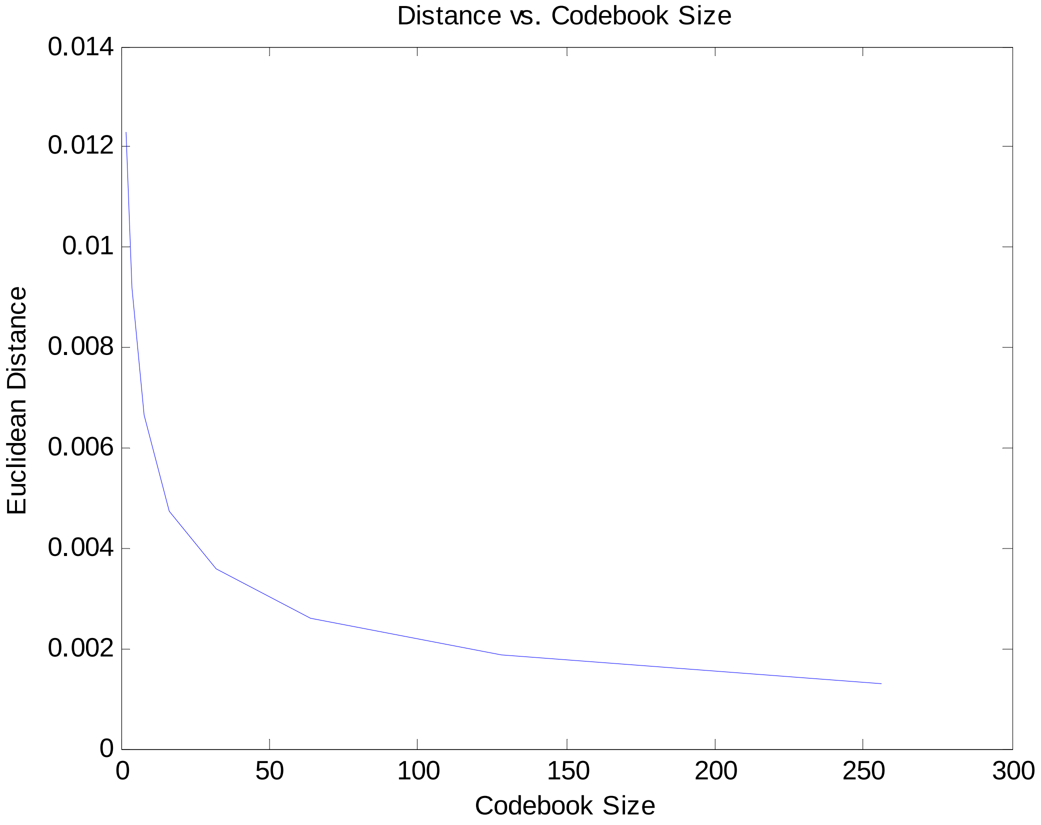


Figure 3.12. Average Distortion versus Codebook Size.

Note that the performance of any codebook would deteriorate if it is used with feature types for which it has not been designed [Makhoul, 1985].

Generally, a codebook with size larger than the number of phonemes (basic speech units) in the English alphabet is preferred to account for the temporal and spectral variability in speech [Rabiner, 1993]. Vector quantization provides a means to reduce storage for speech information, discrete representation and reduced computation for speech modeling. However, Note that vector quantization also causes an inherent distortion in representing the original feature vectors and requires training time to generate a proper codebook.

Reasonably, it should be intended to keep the codebook as small as possible without decreasing performance since a large codebook implies increased computation and training time. Note that the centroids loosely correspond to different acoustic phonemes in the speech. Thus, the proper codebook size can be determined by examining the acoustic complexity of the vocabulary. If the vocabulary is small, fewer symbols might be sufficient [Deller, 2000].

Consequently, considering all the VQ issues discussed above, a codebook of 128 code words was generated by using the *K*-means (LBG) algorithm on the training sequence. The MATLAB implementations used for *K*-means algorithm and codebook generation were those available at [Sorensen, 2005] and are included in Appendices B.3. and B.4., respectively.

Next, the continuous-valued feature vectors on the testing sequence were replaced with the codebook indices so as to obtain discrete symbols for classification, as shown in Figure 3.13.

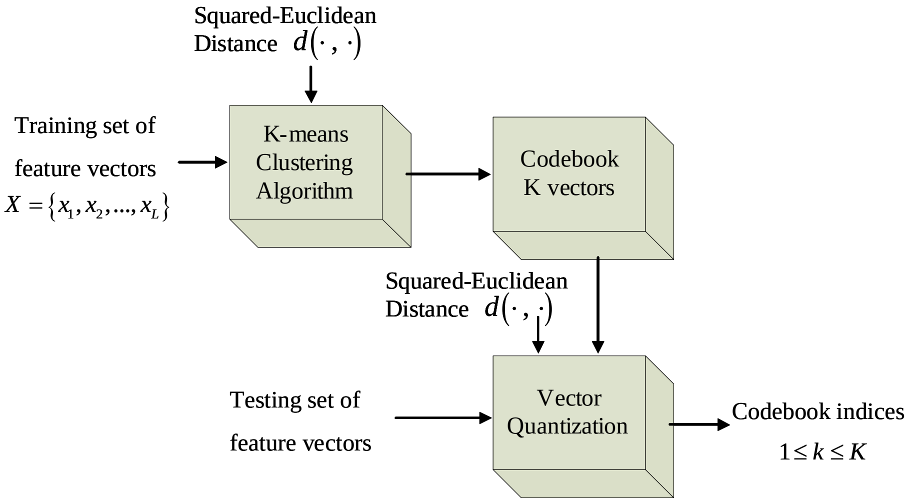

Squared-Euclidean Distance $d(\cdot,\cdot)$
Training set of feature vectors $X=\{x_1,x_2,...,x_L\}$
K-means Clustering Algorithm
Codebook K vectors
Squared-Euclidean Distance $d(\cdot,\cdot)$
Testing set of feature vectors
Vector Quantization
Codebook indices $1\le k\le K$

Figure 3.13. Block Diagram for Vector Quantization of Speech Feature Vectors (After [Rabiner, 1993]).

Vector quantization converts the speech feature vectors (one feature vector per frame) to the discrete observation sequence $O=\{o_1,o_2,...,o_t,...,o_T\}$ where each of the variables o_t may only take integer values k from the trained codebook index for $1\le k\le K$. Now that a discrete set of symbols associated with each speech signal exists, it is possible to build a discrete-symbol Hidden Markov Model (HMM) classifier. The next chapter discusses how to build such a HMM classifier for isolated word recognition.

IV. HIDDEN MARKOV MODELS (HMM) CLASSIFIER FOR ISOLATED WORD RECOGNITION (IWR)

A. INTRODUCTION TO HMM

Natural speech is a time-varying and non-stationary signal. Speech signal modeling helps characterize a set of distinct sounds inherent in speech, such as phonemes, syllables or other subword units thanks to their distinguishing acoustic representations. The fundamental problem in speech signal modeling is that the temporal structure of a specific speech sound is not thoroughly well-defined or easily-predictable due to a wide range of variability in its nature. When all combined, phonetic variability (due to the acoustic differences of the same phoneme in different words), background noise (owing to the microphone or channel distortion), intra-speaker variability (because of the changing physical or emotional condition of the same speaker) and most importantly inter-speaker variability (as a result of different utterances of the same word by different speakers) have tremendous adverse impact on the success of any speech signal model in use. Figure 4.1 illustrates the speaker-variable nature of speech on the different utterances of the same word “Right” spoken by four speakers. Although it is clearly seen that all the waveforms in Figure 4.1 have a common temporal structure, it is possible to observe the significant variations over time in detail from one repetition of the word to the other.

In a broad sense, signal models can be categorized into two classes; namely deterministic models and statistical models [Rabiner, 1989]. In the first class, the Dynamic Time Warping (DTW) approach provides a straight-forward, non-parametric, template-based model which aims to match the incoming speech to the distinct templates generated by normalizing time variations of speech sounds. The templates are generated via training and generally represent coarse speech units like words. Although the DTW approach is relatively simple and straightforward, it has a number of known limitations in speech recognition due to the variability in speech patterns [Gold, 2000; Deller, 2000].

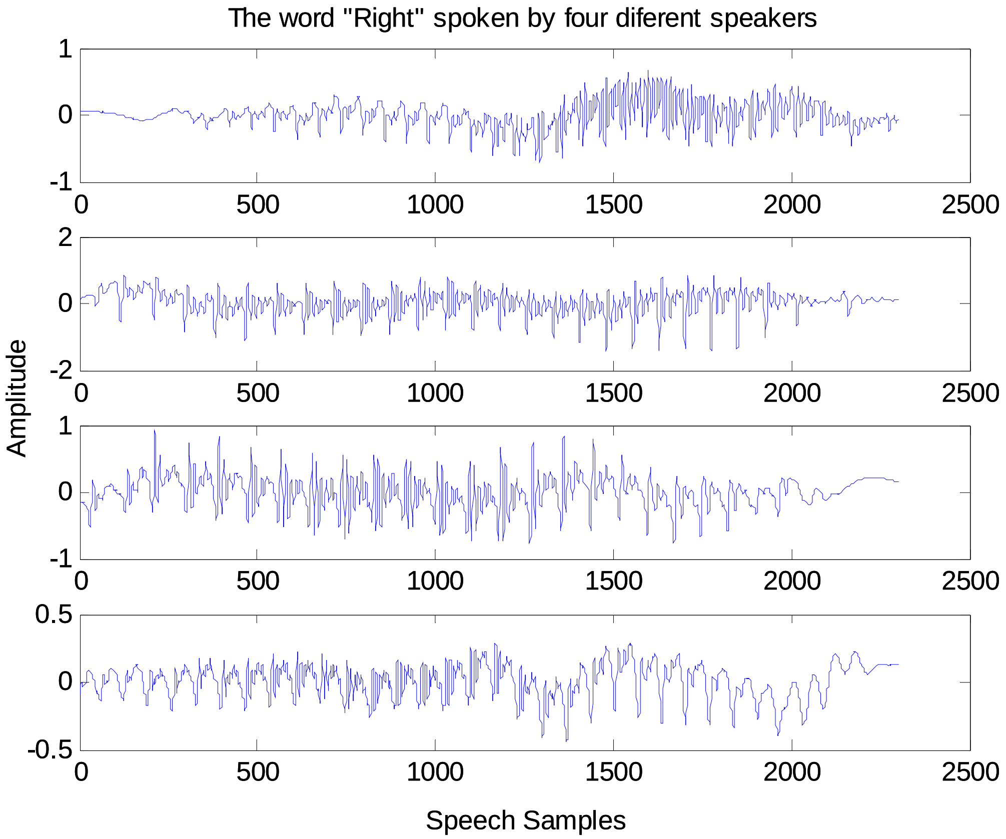


Figure 4.1. Inter-Speaker Temporal Variability in the Word “Right” Spoken by Four Different Speakers.

The second class of statistical speech signal models better accounts for non-linear variability in speech waveform, thereby outperforming the models of the first class. Most statistical approaches inherently model speech as being generated based on some probability distributions. The use of Artificial Neural Networks (ANN) and the Hidden Markov Models technique have mainly constituted relatively elaborate and yet extensively-used statistical models in speech recognition field thus far [Deller, 2000].

The basic theory behind the Hidden Markov Models (HMM) dates back to the late 1900s when Russian statistician Andrei Markov first presented Markov chains. Baum and his colleagues introduced the Hidden Markov Model as an extension to the first-order stochastic Markov process and developed an efficient method for optimizing the HMM parameter estimation in the late 1960s and early 1970s [Deller, 2000]. Baker at Carnegie Mellon University and Jelinek at IBM provided the first HMM implementations to speech processing applications in the 1970s [Rabiner, 1993]. Proper credit should also be given to Jack Ferguson at the Institute for Defense Analysis for explaining the theoretical aspects of three central problems associated with HMMs, which will be further discussed in the following sections [Rabiner, 1989].

The technique of HMM has been broadly accepted in today's modern state-of-the-art ASR systems mainly for two reasons: its capability to model the non-linear dependencies of each speech unit on the adjacent units, and a powerful set of analytical approaches provided for estimating model parameters.

This research strictly deals with a discrete observation HMM implementation of an Isolated Word Recognition (IWR) system to justify the effectiveness of the airflow signals collected within the external air canal for speech recognition. Each word of interest consists of a sequence of phonemes and each phoneme is related to the values of the features extracted from an acoustic speech signal. Ordinarily, the method of HMM makes it possible to capture the non-linear variability in pronunciations (or utterances) using probabilities. The underlying assumption of the HMM approach is that the speech signal can be characterized as a random stochastic process if it is considered to be a sequence of quasi-stationary sounds. In other words, the HMM can be employed as the piecewise stationary model of a nonstationary speech signal [Deng, 2003]. Further, it is then possible to estimate the parameters of this stochastic process in a well-defined linear model structure [Rabiner, 1989].

This chapter first starts with the definition of the isolated word recognition (IWR) and presents the preliminary notations that will be used in the discussions of the Markov Process and Hidden Markov Model. Next, the Discrete-time Markov process is introduced and the ideas to the Hidden Markov Model (HMM) extended.

Understanding the HMM framework is essential to the successful recognition of the airflow signals collected within the external air canal in this research. The three central problems associated with decoding the hidden state sequence, recognition and training of the HMM are discussed. Efficient solution algorithms to these problems are detailed. Also explained are some important implementation issues, such as scaling and training with multiple observation sequences. The chapter concludes with the HMM application to the isolated word recognizer (IWR) used in this research.

1. Definition of the IWR Problem

In an IWR system, the goal is to preserve the temporal and spectral information needed to determine the phonetic identity of speech units and ignore factors like the background noise distortion or undesirable speaker-dependent impacts. In practice, a model is constructed for each word from a small vocabulary. Thus, each word has a probabilistic model of some observable symbol sequence. The task of the Isolated Word Recognition (IWR) system can be defined as finding the best matching word out of an L-word vocabulary whose model is most likely to generate the given observation sequence of feature vectors extracted from a speech signal. Figure 4.2 illustrates the simplified mechanism of a discrete observation HMM classifier for IWR task works.

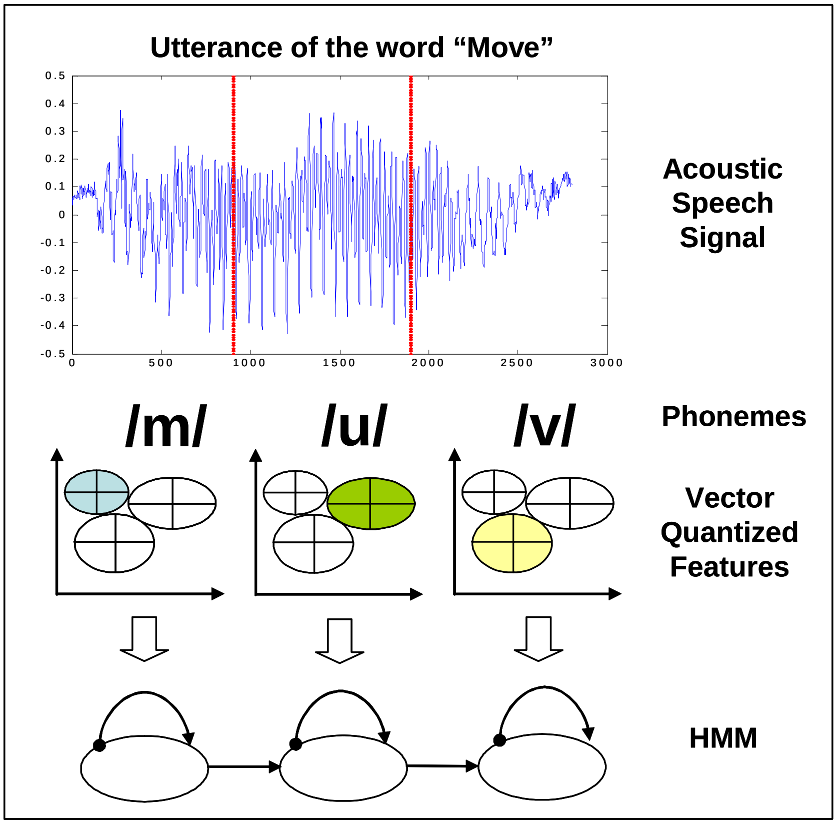


Figure 4.2 An Illustration of a Discrete Observation HMM Classifier for IWR.

Assume that an observation sequence (or a set of symbols) corresponds to a set of feature vectors extracted from an acoustic signal. Let this observation sequence be represented by $O = \{o_1, o_2, ..., o_M\}$. Further, assume that the models of the words to be recognized correspond to $W = \{w_1, w_2, ..., w_N\}$. Then, the following equation estimates the most likely word out of the L-word vocabulary, given the observation sequence O [Jelinek, 1997].

$$\hat{w} = \arg\max_{W \in L} \left[P\left(W \middle| O\right) \right]. \tag{4.1}$$

By applying the Bayes Rule, $P\left(x|y\right) = \frac{P\left(y|x\right)P\left(x\right)}{P\left(y\right)}$ to Eq. (4.1), the recognition task transforms to the following equation:

$$\hat{w} = \arg\max_{W \in L} \left[P\left(W \middle| O\right) \right] = \arg\max_{W \in L} \left[\frac{P\left(O|W\right)P\left(W\right)}{P\left(O\right)} \right]. \tag{4.2}$$

In Eq. (4.2), $P\left(W\right)$ is the prior probability of having that particular model, which produces the word to be recognized. Fortunately, $P\left(O\right)$, the probability of the observation sequence, can be ignored since this probability is constant when attempting to maximize over the possible word models [Jelinek, 1997]. Hence, the resulting equation becomes:

$$\hat{w} = \arg\max_{W \in L} \left[P\left(O|W\right)P\left(W\right) \right]. \tag{4.3}$$

$P\left(O|W\right)$, the probability of having a particular observation sequence O given the word model W, will be obtained by using Hidden Markov Modeling.

2. Notations

Different sets of notations for Hidden Markov Modeling exist in the literature. Table 4.1 lists the preliminary notation that represent the basic HMM concepts. More advanced concepts follow on these notations. These notations are selected from [Deller, 2000; Rabiner, 1989; Dugad, 1996] for convenience and consistency throughout the chapter.

Notation	Definition
$Q = \{q_1, q_2, ..., q_t, ..., q_S\}$	Q is the set of possible states where q_t is the state at time t and S is the number of states.
$O = \{o_1, o_2, ..., o_t, ..., o_K\}$	O is the set of possible observation symbols where K is the number of symbols in the codebook and o_t is the symbol observed at time t.
$A = \begin{pmatrix} a(1\mid 1) & a(1\mid 2) & \dots & a(1\mid S) \\ & \ddots & & \\ \vdots & a(i\mid j) & & \vdots \\ & & \ddots & \\ a(S\mid 1) & a(S\mid 2) & \cdots & a(S\mid S) \end{pmatrix}$	A is the *S-by-S State transition matrix* where $a(i\mid j) = P(q_t = i\mid q_{t-1} = j)$ is the probability of being in state i at time t given the previous state was j at time $t-1$. Columns of A must sum to unity.
$B = \begin{pmatrix} b(1\mid 1) & a(1\mid 2) & \dots & b(1\mid S) \\ & \ddots & & \\ \vdots & \mathrm{b}(k\mid i) & & \vdots \\ & & \ddots & \\ a(K\mid 1) & a(K\mid 2) & \cdots & a(K\mid S) \end{pmatrix}$	B is the *K-by-S Observation probability matrix* where $b(k\mid i) = P(o_t = k\mid q_t = i)$ is the probability of having the observation k at time t given the model is in state i at time t. Columns of B must add up to unity as well.
$\pi = \begin{bmatrix} P(q_t = 1) = \pi_{q_1} \\ \vdots \\ P(q_t = i) = \pi_{q_i} \\ \vdots \\ P(q_t = S) = \pi_{q_t} \end{bmatrix}$	π is the *S-by-1 Initial state probability vector* which shows the probabilities of having any state at time $t = 1$. Elements of the column vector π must also sum to unity.
$\lambda = \{A, B, \pi\}$	λ is the compact notation for a HMM.
$\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = i\mid\lambda).$	The forward variable.
$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T\mid q_t = i, \lambda).$	The backward variable.

Table 4.1. List of Basic Notations for HMMs.

3. Discrete-Time Markov Process

The Markov Chain is the underlying stochastic process of Hidden Markov Models. A finite-state Markov Chain is a stochastic discrete-time Markov process whose output is a set of S distinct states where each state corresponds to an observation.

[Rabiner, 1989]. State transitions (transition back to the original state is also possible) in a discrete-time Markov process can occur based on only two parameters:

- State transition probability, $a(i|j) = P(q_t = i | q_{t-1} = j)$ where $1 \leq i, j \leq S$, and $\sum_{i=1}^{S} a(i|j) = 1$ for $\forall j$.
- Initial state probability, $P(q_1 = i)$ where $1 \leq i \leq S$ and $\sum_{i=1}^{S} P(q_1 = i) = 1$.

There are two distinct characteristics of a Markov process. First, there is one-to-one correspondence between the observation sequence and the Markov Chain state sequence. In other words, observations are deterministic. Second, the Markov Chain state sequence is also observable.

The following example provides a better visualization of a discrete-time Markov process.

a. Example 4.1: Three-State Observable Markov Model

Assume that a Christmas tree is ornamented with a decorative light which takes on three colors in a sequence based on some priori probabilities. Red (R) color corresponds to state 1, green (G) color to state 2 and blue (B) color to state 3. Model parameters are given as follows:

$$A = \begin{bmatrix} a(1|1) = 0.5 & a(1|2) = 0.6 & a(1|3) = 0.3 \\ a(2|1) = 0.3 & a(2|2) = 0.4 & a(2|3) = 0.6 \\ a(3|1) = 0.2 & a(3|2) = 0 & a(3|3) = 0.1 \end{bmatrix} \text{ and}$$

$$\pi = \begin{bmatrix} P(q_1 = 1) = 0.4 \\ P(q_1 = 2) = 0.3 \\ P(q_1 = 3) = 0.3 \end{bmatrix}.$$

This three-state Markov model is illustrated in Figure 4.3.

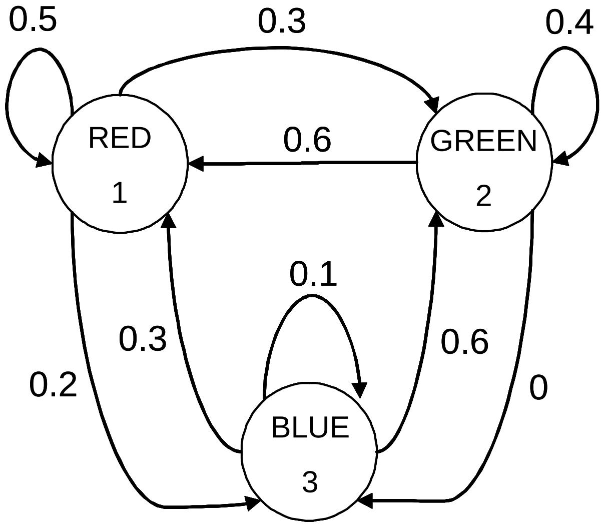


Figure 4.3. Three-State Discrete-Time Markov Chain.

Given a sequence of observed colors when the decorative lights are turned on, $O=\{\mathrm{R}-G-G-R-B-B\}$, the only corresponding state sequence is $Q=\{s_1,s_2,s_2,s_1,s_3,s_3\}$. The probability of having this observation sequence can be computed as

$$
\begin{aligned}
P(O|\lambda)=P(\mathrm{R}-G-G-R-B-B|\lambda)=P(Q|\lambda)=P(s_1,s_2,s_2,s_1,s_3,s_3|\lambda)\\
=P(q_1=1)P(s_2|s_1)P(s_2|s_2)P(s_1|s_2)P(s_3|s_1)P(s_3|s_3)\\
=0.4\times0.3\times0.4\times0.6\times0.2\times0.1=5.76\times10^{-4}.
\end{aligned}
$$

Finally, also note that an *n*th-order discrete-time Markov process is a sequence of random variables which depends only on the preceding *n* variables. A first-order Markov model is typically used in speech recognition [Gold, 2000].

Next, the extension of discrete-time Markov process to Hidden Markov Models (HMMs) is addressed.

4. Extension to Discrete-Symbol HMMs

In a stochastic Markov Model, each state is associated with a deterministically observable event and hence the name *observable Markov Model* is also given. The *Hidden Markov Model* can be considered an extended version of an observable Markov Model where the observation is a probabilistic function of the state [Rabiner, 1993].

Given an observation sequence, there is no direct access to the underlying state sequence which generates the observations in a HMM; hence the state sequence is hidden.

A Hidden Markov Model can be characterized by the following five model parameters:

- S, number of hidden states in the model.
- K, number of distinct observation symbols.
- The state transition probability distribution, $A=\{a(i|j)\}$, where $a(i|j)=P(q_t=i|q_{t-1}=j)$ and $1\le i, j\le S$.
- The observation symbol probability distribution, $B=\{b(k|i)\}$, where $b(k|i)=P(o_t=k|q_t=i)$, $1\le i\le S$, $1\le k\le K$.
- The initial state distribution, $\pi=\{P(q_1=i)\}$, where $1\le i\le S$.

The compact notation, $\lambda=(A,B,\lambda)$ is usually used to define the model parameters of an HMM for convenience.

To better explain the extension of the observable Markov Model to HMM, the following two fundamental assumptions behind HMMs are discussed as well [Deng, 2003].

- *First-order Markov process assumption* which states that the state transition at time t depends only on the previous state at time $t-1$, as expressed in the following equation. Note that state transition probability is not time-variant.

$$P(Q|\lambda)=P(q_1,...,q_t,...,q_T|\lambda)=P(q_1|\lambda)\prod_{t=2}^{T}P(q_t|q_{t-1},\lambda). \tag{4.4}$$

$$=\pi_{q_1}a(q_2|q_1)a(q_3|q_2)...a(q_T|q_{T-1}). \tag{4.5}$$

- *Independent observation assumption* which states that an observation is only dependent on the particular state that generates it and is independent from its preceding observations, as shown in Eq. (4.5).

$$P(O|Q,\lambda)=P(o_1,o_2,...,o_t,...,o_T|q_1,q_2,...,q_t,...,q_T,\lambda)=\prod_{t=1}^{T}P(o_t|q_t,\lambda). \tag{4.6}$$

$$=b(o_1|q_1)b(o_2|q_2)...b(o_T|q_T). \tag{4.7}$$

The following two examples illustrate two different structures of HMMs.

a. *Example 4.2: Three-State Ergodic Hidden Markov Model*

The set-up of our previous example is extended with the Christmas tree light to incorporate the hidden states and the non-deterministic observations in this example. Suppose a Christmas tree light which turns on any one of the three colors; red (R), green (G) and blue (B) each time a button is hit on a remote control device (RCD). Assume that a kid changes the colors of the Christmas tree light by using any one of three RCDs. Each RCD is biased for the generation of colors. There is no direct access to the information about which particular RCD is used whenever a change of color occurs as an observation. Thus, the following three hidden states occur.

- State 1: RCD1 is used.
- State 2: RCD2 is used.
- State 3: RCD3 is used.

The matrix A of state transition probabilities is given as follows:

$$A = \left\{ a\left(i \middle| j\right) \right\} = \begin{bmatrix} P\left(q_t = 1 \middle| q_{t-1} = 1\right) = 0.2 & P\left(1 \middle| 2\right) = 0.3 & P\left(1 \middle| 3\right) = 0.1 \\ P\left(q_t = 2 \middle| q_{t-1} = 1\right) = 0.4 & P\left(2 \middle| 2\right) = 0.4 & P\left(2 \middle| 3\right) = 0.7 \\ P\left(q_t = 3 \middle| q_{t-1} = 1\right) = 0.4 & P\left(3 \middle| 2\right) = 0.3 & P\left(3 \middle| 3\right) = 0.2 \end{bmatrix}.$$

The initial state probability vector, π is

$$\pi = \begin{bmatrix} P\left(q_1 = 1\right) = 0.5 \\ P\left(q_1 = 2\right) = 0.2 \\ P\left(q_1 = 3\right) = 0.3 \end{bmatrix}.$$

The matrix B of observation probabilities is expressed as follows:

$$B = \left\{ b\left(k \middle| i\right) \right\} = \begin{bmatrix} P\left(o_t = R \middle| q_t = 1\right) = 0.6 & P\left(R \middle| 2\right) = 0.3 & P\left(R \middle| 3\right) = 0.4 \\ P\left(o_t = G \middle| q_t = 1\right) = 0.1 & P\left(G \middle| 2\right) = 0.5 & P\left(G \middle| 3\right) = 0.3 \\ P\left(o_t = B \middle| q_t = 1\right) = 0.3 & P\left(B \middle| 2\right) = 0.2 & P\left(B \middle| 3\right) = 0.3 \end{bmatrix}.$$

This HMM is called an *ergodic model* in that any state can be reached from any other state. In other words, an ergodic HMM allows for all possible state transitions at any time, as shown in Figure 4.4.

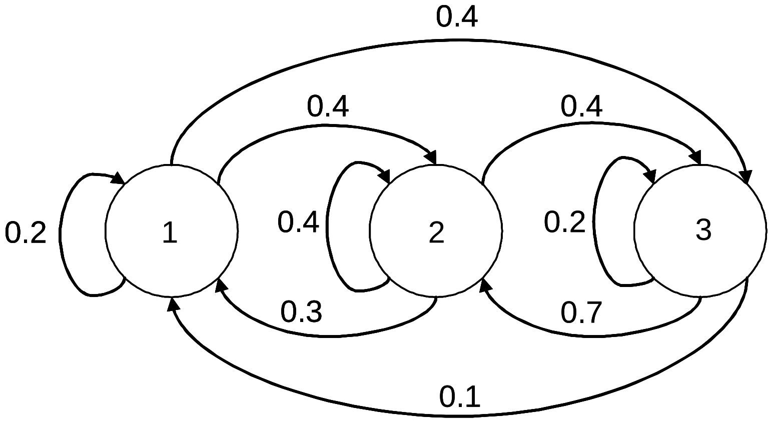


Figure 4.4. Three-State Ergodic HMM.

Given that the colors $O = \{B - G - G - R\}$ appear in a sequence of observations, the following question can be asked: *What is the probability of having this observation sequence given the model parameters?*

In this case, it should be obvious that there are $3 \times 3 \times 3 \times 3 = 81$ possible corresponding state sequences, Q_i which might have generated the given observation sequence, O. Thus, the probability, $P(O|\lambda)$ is

$$P(O|\lambda) = \sum_{i=1}^{81} P(O, Q_i|\lambda) = \sum_{i=1}^{81} P(O|Q_i, \lambda) P(Q_i|\lambda).$$

For example, if it is known that the generating state sequence is presumably $Q = \{s_1 - s_3 - s_2 - s_2\}$, then it should be possible to compute the probability, $P(O|\lambda)$ as follows:

$$P(O|\lambda) = P(O|Q, \lambda) P(Q|\lambda) = 0.0135 \times 0.056 = 7.56 \times 10^{-4}.$$

b. Example 4.3: Five-State Left-to-right Hidden Markov Model

Consider a bag-drawing system consisting of *bags and coins* as another example of HMM. Assume there are five bags and three different types of coins; nickels, dimes and quarters in each bag. Each bag has ten coins in differing numbers from each type of coin. Six drawings from five bags are conducted when blindfolded. Therefore, it is not known which drawing comes from which bag. However, there is a restriction to the sequence of the bags to be used. Once a coin is drawn from any bag, the only choice is to choose from either the same bag or go to the one of the next two larger-numbered bags for the next drawing. For instance, $Q = \{bag_1, bag_2, bag_2, bag_3, bag_5, bag_5\}$ is a valid sequence of bags for the drawing. Thus, the drawing bags correspond to the hidden states whereas the six coins drawn represent the observation sequence in the model.

The model parameters are given as

$$A = \begin{bmatrix} 0.2 & 0 & 0 & 0 & 0 \\ 0.5 & 0.3 & 0 & 0 & 0 \\ 0.3 & 0.4 & 0.4 & 0 & 0 \\ 0 & 0.3 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0.1 & 0.7 & 1 \end{bmatrix}, \; B = \begin{bmatrix} 0.7 & 0.2 & 0.4 & 0.2 & 0.5 \\ 0.1 & 0.5 & 0.3 & 0.6 & 0.4 \\ 0.2 & 0.3 & 0.3 & 0.2 & 0.1 \end{bmatrix} \text{ and } \pi = \begin{bmatrix} 0.4 \\ 0.2 \\ 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}.$$

Note that the upper triangular portion of the state transition matrix A represents the backward state transitions, which is all zero. This type of model structure is a special case of the model known as a *left-to-right model*. In a left-to-right HMM, backward state transitions are not allowed. Speech has inherently a temporal structure and thereby can be modeled to fit to the left-to-right HMM in a well-behaved fashion. Figure 4.5 illustrates the structure of this specific type of HMM.

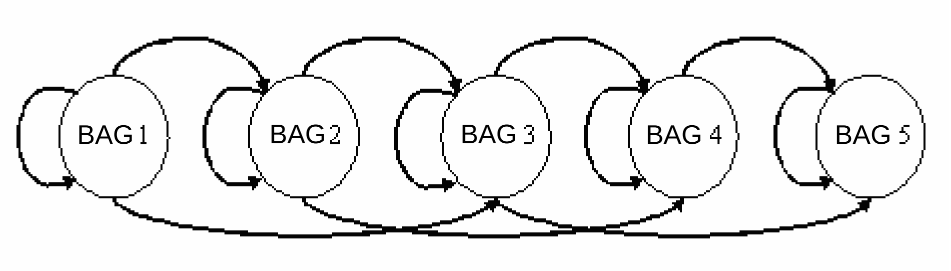


Figure 4.5. Five-State Left-to-Right HMM.

Next, the three fundamental problems associated with the use of HMM and their respective solutions are discussed.

B. THE THREE FUNDAMENTAL PROBLEMS FOR HMM

For a practical implementation of HMM, three basic problems of interest must be discussed whose effective solutions will lead to the training of the model, that is the estimation of the model parameters, $\lambda = \{A, B, \pi\}$, and finally the resulting classifier. Rabiner et al. lists these central problems as follows [Rabiner, 1989]:

- **Problem 1:** Given an observation sequence $O = \{o_1, o_2, ..., o_t, ..., o_T\}$ and a HMM $\lambda = \{A, B, \pi\}$, how is $P(O|\lambda)$, the probability that the observation sequence is generated by the given model, computed? Problem 1 is known as the *evaluation* or *scoring problem*.
- **Problem 2:** Given an observation sequence $O = \{o_1, o_2, ..., o_t, ..., o_T\}$ and a HMM $\lambda = \{A, B, \pi\}$, how is the generating state sequence $Q^* = \{q_1, q_2, ..., q_t, ..., q_T\}$, which accounts for the observations optimally, determined? A solution that would satisfy $Q^* = \arg\max_{All\ Q} \left[P(Q, O|\lambda) \right]$ in this problem is sought. Problem 2 is also known as the *decoding problem* in that the goal is to uncover the hidden part of the model, the optimal state sequence.
- **Problem 3:** How is the model parameter set $\lambda = \{A, B, \pi\}$, which maximizes $P(O|\lambda)$, the probability of the observation sequence given the model, estimated? Problem 3 represents the *training problem*.

1. Solution to the Evaluation Problem

Computing the probability of occurrence of a particular observation sequence given the model can also be considered a scoring problem in which the attempt is to determine the suitability of a given model to generate the particular observation sequence. This standpoint becomes useful when aiming to choose the best model for a designated word among many competing models where each model represents a word from a small vocabulary [Rabiner, 1989].

Basically, there are two options available to evaluate the probability $P(O|\lambda)$; the direct brute-force evaluation and the forward-backward procedure. The former approach, called the direct evaluation, is straightforward albeit impractical for many real-world applications. The latter method, called the forward-backward procedure, is extensively used to address the evaluation problem in HMM.

a. The Direct Evaluation

The brute-force approach to the computation of $P(O|\lambda)$ starts with finding the conditional probability $P(O|Q,\lambda)$ along a candidate path (any allowable state sequence) $Q=\{q_1,q_2,...,q_t,...,q_T\}$ then multiplies it by the probability of such a path, $P(Q|\lambda)$. Summation of these products over all possible state sequences accounts for the probability $P(O|\lambda)$. Applying *the independent output assumption*, the joint output probability of observations along the path Q is computed

$$P(O|Q,\lambda)=\prod_{t=1}^{T}P(o_t|q_t,\lambda)=b(o_1|q_1)b(o_2|q_2)...b(o_T|q_T). \tag{4.8}$$

The probability of the path Q is computed using a *first-order Markov assumption* as follows:

$$P(Q|\lambda)=P(q_1|\lambda)\prod_{t=2}^{T}P(q_t|q_{t-1},\lambda)=\pi_{q_1}a(q_2|q_1)a(q_3|q_2)...a(q_T|q_{T-1}). \tag{4.9}$$

Therefore, the probability of observation sequence given the model is obtained by the following computation:

$$P(O|\lambda)=\sum_{\text{all Q}}P(O|Q,\lambda)P(Q|\lambda) \tag{4.10}$$

$$=\sum_{\text{all Q}}\pi_{q_1}b(o_1|q_1)a(q_2|q_1)b(o_2|q_2)...a(q_T|q_{T-1})b(o_T|q_T). \tag{4.11}$$

A close observation of Eq. (4.11) reveals the complexity of the computational requirements in the brute-force direct evaluation approach. Considering that there are S^T distinct possible state sequences and $2T-1$ multiplications are performed in the summand of Eq. (4.11), there is a total of $(2T-1)S^T \approx 2T\cdot S^T$ multiplications and S^T-1 additions [Dugad, 1996]. Therefore, the brute-force computation is very expensive and it becomes essential to compute $P(O|\lambda)$ more efficiently. Next, we consider the forward-backward procedure which is a very efficient solution to the evaluation problem.

b. The Forward-Backward Procedure

First, *the forward variable* $\alpha_t\left(i\right)$ is introduced and defined as:

$$\alpha_t\left(i\right) = P\left(o_1, o_2, ..., o_t, q_t = i \middle| \lambda\right). \tag{4.12}$$

The forward variable $\alpha_t\left(i\right)$ is described as the joint probability of the partial observation sequence up to instant t, $\left\{o_1, o_2, ...o_t\right\}$ and the state i at instant t, given the model λ. *The Forward Recursion* can be used to solve for $\alpha_t\left(i\right)$ inductively as follows:

1. **Initialization:** for $1 \le i \le S$,

$$\alpha_1\left(i\right) = P\left(o_1, q_1 = i \middle| \lambda\right) = \pi_i b\left(o_1 \middle| q_1\right). \tag{4.13}$$

2. **Recursion:** for $t = 1, 2, ..., T-1$ and $1 \le j \le S$,

$$\alpha_{t+1}\left(j\right) = \left[\sum_{i=1}^{S} \alpha_t\left(i\right) a\left(j \middle| i\right)\right] b\left(o_{t+1} \middle| j\right). \tag{4.14}$$

3. **Termination:** for $1 \le i \le S$,

$$P\left(O \middle| \lambda\right) = \sum_{i=1}^{S} \alpha_T\left(i\right). \tag{4.15}$$

In the first step, the forward probabilities are initialized as the joint probability of having the observation o_1 at the initial state i for all possible states. In the second step, $\alpha_{t+1}\left(j\right)$, the joint probability of the partial observation sequence up to instant $t+1$ and the state j at instant $t+1$ is computed from $\alpha_t\left(i\right)$, probabilities of the previous partial observations, recursively. The Lattice (Trellis) structure shown in Figure 4.6 illustrates how state j at time $t+1$ can be reached independently from any of the S states at time t. The third step of the forward recursion shows that the desired calculation of $P\left(O \middle| \lambda\right)$ is obtained from the summation of S independent final forward variables, $\alpha_T\left(i\right)'s$ which realize the observation sequence [Rabiner, 1993].

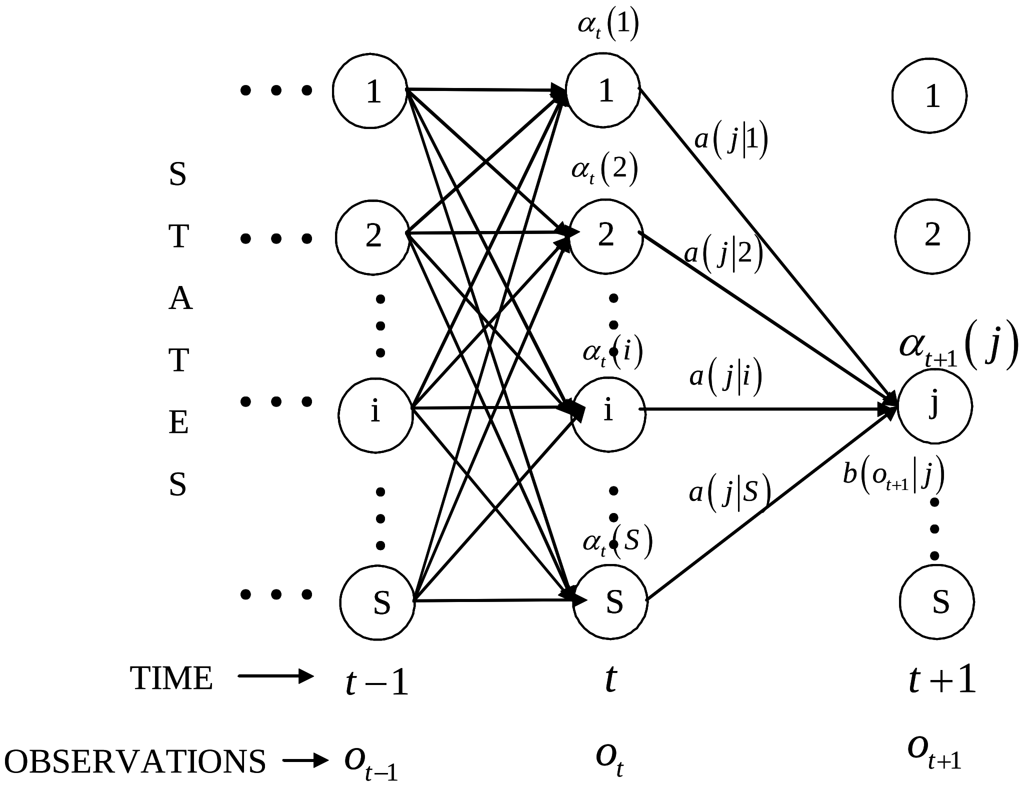


Figure 4.6. The Lattice Structure Used to Derive the Forward Recursion (After [Deller, 2000]).

The proof of the Forward Recursion is provided from [Wang, 2005] by using the HMM assumptions [Eqs. (4.4) through (4.7)], the principal of conditional probability given as:

$$P(A,B|\lambda) = P(A|B,\lambda)P(B|\lambda) = P(A|\lambda)P(B|A,\lambda), \tag{4.16}$$

and the principal of total probability expressed as:

$$P(A) = \sum_{All\ B} P(A,B). \tag{4.17}$$

$$
\begin{aligned}
\alpha_{t+1}(j) &= P\left(o_1,o_2,...,o_t,o_{t+1},q_{t+1}=j\middle|\lambda\right), \\
&= P\left(o_1,o_2,...,o_t,o_{t+1}\middle|q_{t+1}=j,\lambda\right)P\left(q_{t+1}=j\middle|\lambda\right), \text{ [From Eq. (4.16)]} \\
&= P\left(o_1,o_2,...,o_t\middle|q_{t+1}=j,\lambda\right)P\left(o_{t+1}\middle|q_{t+1}=j,\lambda\right)P\left(q_{t+1}=j\middle|\lambda\right), \text{ [From Eqs.(4.6)]} \\
&= P\left(o_1,o_2,...,o_t,q_{t+1}=j\middle|\lambda\right)P\left(o_{t+1}\middle|q_{t+1}=j,\lambda\right), \text{ [From Eq. (4.16)]} \\
&= P\left(o_1,o_2,...,o_t,q_{t+1}=j\middle|\lambda\right)b\left(o_{t+1}\middle|j\right) \\
&= \left[\sum_{i=1}^{S} P\left(o_1,o_2,...,o_t,q_t=i,q_{t+1}=j\middle|\lambda\right)\right]b\left(o_{t+1}\middle|j\right), \text{ [From Eq.(4.17)]} \\
&= \left[\sum_{i=1}^{S} P\left(o_1,o_2,...,o_t,q_t=i\middle|\lambda\right)P\left(q_{t+1}=j\middle|o_1,o_2,...,o_t,q_t=i,\lambda\right)\right]b\left(o_{t+1}\middle|j\right), \text{ [From Eq.(4.16)]} \\
&= \left[\sum_{i=1}^{S} P\left(o_1,o_2,...,o_t,q_t=i\middle|\lambda\right)P\left(q_{t+1}=j\middle|q_t=i,\lambda\right)\right]b\left(o_{t+1}\middle|j\right), \text{ [From Eq.(4.4)]} \\
&= \left[\sum_{i=1}^{S} \alpha_t(i)a\left(j\middle|i\right)\right]b\left(o_{t+1}\middle|j\right) \text{ [From Eq.(4.12)].}
\end{aligned}
$$

It should be noted that there are N multiplications involved in the first step and $S(S+1)(S-1)$ multiplications involved in the second step. There are no multiplications in the third step. Therefore, the Forward Recursion results in a total of $S+S(S+1)(T-1)\approx S^2T$ multiplications in comparison with $2T\cdot N^T$ multiplications required by the direct evaluation approach [Dugad, 1996]. Consequently, the Forward Recursion provides a practically feasible method for the computation of the desired probability $P(O|\lambda)$.

Next, *the backward variable* $\beta_t(i)$ is defined in a similar way as was done with the forward variable as follows:

$$\beta_t(i)=P\left(o_{t+1},o_{t+2},...,o_T\middle|q_t=i,\lambda\right). \tag{4.18}$$

$\beta_t(i)$ is defined as the conditional probability of the observation sequence from the instant $t+1$ to the final observation instant T given the state i at the instant t and the model λ. The probability $P(O|\lambda)$ can be evaluated by computing $\beta_t(i)$ with iterations which are backward in time, hence called *the Backward Recursion*.

Although applying the Forward Recursion is enough for the computation of $P(O|\lambda)$, a combination of forward and backward procedures becomes useful when dealing with the training problem. Thus, the Backward Procedure is presented, as given in [Dugad, 1996].

1. **Initialization:** for $1 \leq i \leq S$,

$$\beta_T(i) = 1, \tag{4.19}$$

2. **Recursion:** for $t = T-1, T-2, ..., 2, 1$ and $1 \leq j \leq S$,

$$\beta_t(i) = \sum_{j=1}^{S} a(j|i) b(o_{t+1}|j) \beta_{t+1}(j), \tag{4.20}$$

3. **Termination:** for $1 \leq i \leq S$,

$$P(O|\lambda) = \sum_{i=1}^{S} \pi_i b(o_1|i) \beta_1(i). \tag{4.21}$$

The computation of the probability $P(O|\lambda)$ by using the Backward Recursion results in multiplications on the order of S^2T just like the Forward Procedure; thereby providing an equally efficient method.

2. Solution to the Decoding Problem

The solution to Problem 1 computes $P(O|\lambda)$ by summing up all possible state sequences through a Trellis structure in a HMM, hence it does not result in the optimal state sequence that accounts for the observation sequence. In Problem 2, the single most-likely state sequence (path) $Q^* = \{q_1, q_2, ..., q_t, ..., q_T\}$ is sought for a given observation sequence and model. In other words, the goal is to find the optimal path that satisfies $Q^* = \arg\max_{All\ Q}\left[P(Q, O|\lambda)\right]$, considering that maximizing the probability $P(Q|O, \lambda)$ is equivalent to maximizing $P(Q, O|\lambda)$. The Viterbi Algorithm can be used to determine such an optimal state sequence as a dynamic programming method applied to HMM [Rabiner, 1993].

a. The Viterbi Algorithm

The Viterbi Algorithm chooses and keeps the most-likely state sequence for each of the S possible states at each instant recursively unlike the Forward Recursion which sums over the transitions from all possible incoming states reaching the same destination state. Once the Viterbi Algorithm reaches the final step at the end of observation sequence, it uses back-tracing to find the most probably path.

Following reformulation of the decoding problem provides an insight into the Viterbi Algorithm as it is applied to the estimation of the optimal state sequence in a HMM [Dugad, 1996]. Ultimately, the goal is to maximize the probability $P\left(Q,O|\lambda\right)$. The principal of conditional probability is employed first using Eqs. (4.10) and (4.11) to obtain

$$P\left(O,Q|\lambda\right)=P\left(O|Q,\lambda\right)P\left(Q|\lambda\right) \tag{4.22}$$

$$=\pi_i b\left(o_1|q_1\right)a\left(q_2|q_1\right)b\left(o_2|q_2\right)...a\left(q_T|q_{T-1}\right)b\left(o_T|q_T\right). \tag{4.23}$$

Then, a distance function associated with any possible state sequence along the path from $t=1$ up to $t=T$ is defined by taking the negative logarithm of Eq. (4.23) as follows:

$$D_T=D\left(q_1,q_2,...,q_T\right)=-\left[\log\left(\pi_{q_1}b\left(o_1|q_1\right)\right)+\sum_{t=2}^{T}\log\left(a\left(q_t|q_{t-1}\right)b\left(o_t|q_t\right)\right)\right]. \tag{4.24}$$

Now, observe that the maximization problem to obtain the most-likely state sequence given in the following equation

$$Q^*=\underset{All\ q_t}{\arg\max}\left[P\left(Q,O|\lambda\right)\right]\text{ for }t=1,2,...,T \tag{4.25}$$

turns out to be a minimization problem over the distance function as expressed in

$$Q^*=\underset{All\ q_t}{\arg\min}\left[D\left(q_1,q_2,...,q_T\right)\right]. \tag{4.26}$$

The term $-\log\left(\pi_{q_1} b\left(o_1 | q_1\right)\right)$ in Eq. (4.24) is called the initial weight of having observation o_1 at the initial state q_1. The weight of a given state sequence is equal to the summation of all the weights along the path. In a similar way, the term $-\log\left(a\left(j|i\right)b\left(o_t|j\right)\right)$ in Eq. (4.24) corresponds to the cost associated with going from state i to state j at time t. Note that the weight function is associated with a state sequence, whereas the cost function is associated with a state transition only.

There are two expressions used in the Viterbi Algorithm. The term $\delta_t(i)$ stands for the accumulated weight along the path as far as proceeding to state i at time t. The term $\Psi_t(j)$ denotes the state at time $t-1$ whose transition to state j at time t has the minimum corresponding cost. It is necessary to keep track of the latter argument so as to backtrack and decode the optimal state sequence in the end. It is now possible to present the Viterbi Algorithm as follows:

1. **Initialization:** for $1 \le i \le S$

$$\delta_1(i) = -\log(\pi_i) - \log\left(b\left(o_1|i\right)\right), \tag{4.27}$$

$$\Psi_1(i) = 0, \tag{4.28}$$

2. **Recursion:** for $2 \le t \le T$ and $1 \le j \le N$

$$\delta_t(j) = \min_{1 \le i \le S}\left[\delta_{t-1}(i) - \log\left(a\left(j|i\right)\right)\right] - \log\left(b\left(o_t|j\right)\right), \tag{4.29}$$

$$\Psi_t(j) = \arg\min_{1 \le i \le S}\left[\delta_{t-1}(i) - \log\left(a\left(j|i\right)\right)\right], \tag{4.30}$$

3. **Termination:**

$$P^* = \min_{1 \le i \le S}\left[\delta_T(i)\right], \tag{4.31}$$

$$q_T^* = \arg\min_{1 \le i \le S}\left[\delta_T(i)\right], \tag{4.32}$$

4. **Backtracking:** for $t = T-1, T-2, ..., 2, 1$

$$q_t^* = \Psi_{t+1}\left(q_{t+1}^*\right). \tag{4.33}$$

The argument P^*, the negative log probability of the path with the minimum weight, is called the *log likelihood*. Although the expression $\exp\left(-P^*\right)$ equals the desired probability strictly, the argument of log likelihood is preferably used instead of the regular probability for convenience in computations without any conversion. Note that small log likelihood measures correspond to large probabilities. In addition, taking the negative logarithm is a means of circumventing the precision and underflow problems which occur when small probability values are multiplied iteratively.

A little thought over the steps involved in the Viterbi Algorithm reveals that the computational requirement for finding the most-likely path is, similar to the Forward Procedure, on the order of S^2T additions.

Figure 4.7 illustrates that the Viterbi Algorithm operates in a Trellis structure in a similar way as that followed in the Forward Recursion. Bold arrows represent the choice of the most-likely state at each time step and the resulting best state sequence is shown as $Q^* = \{S_2, S_1, S_3, S_2, S_2\}$ in the three-state HMM in Figure 4.7.

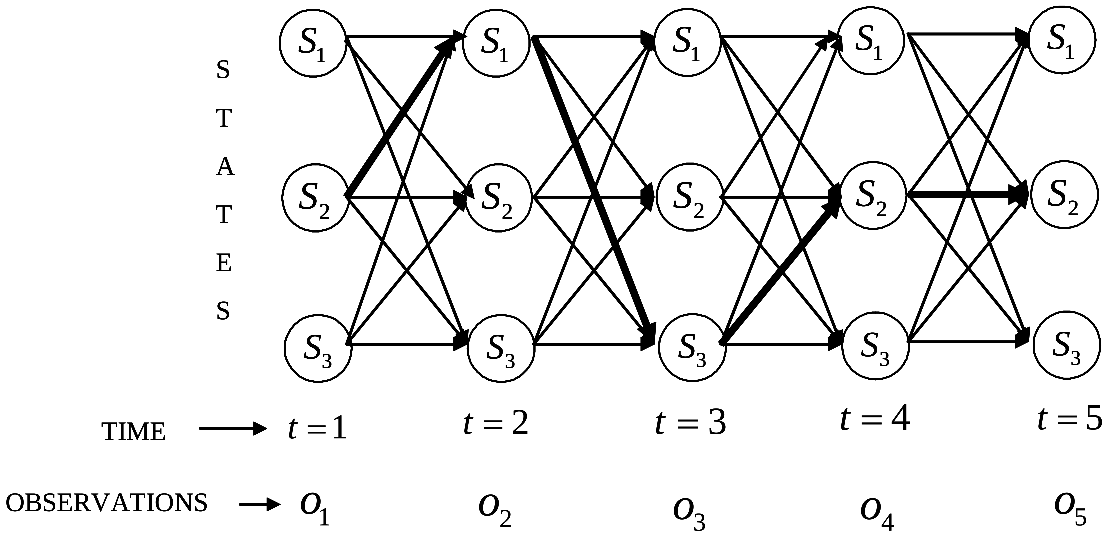


Figure 4.7 Trellis Structure Illustrating the Viterbi Algorithm for a Three-State HMM

3. Solution to the Training Problem

The training problem deals with adjusting (or re-estimating) the model parameters $\lambda\left(A,B,\pi\right)$ in an effort to learn and encode the characteristics of the observation sequence into the model such that the model should identify a similar observation sequence in future [Dugad, 1996].

Unfortunately, optimizing the training parameters of a HMM is difficult to address as there is no known analytical method that maximizes the joint probability of the training data in a closed form [Rabiner, 1993]. Two of the most popular methods to handle this problem are discussed. First, it is possible to optimize the parameters $\lambda\left(A,B,\pi\right)$ such that the likelihood of occurrence for the training data $P\left(O\middle|\lambda\right)$ is locally maximized by using the iterative *Baum-Welch algorithm*, also known as the *Forward-Backward algorithm* [Baum, 1966; Baum 1970]. Second, the problem can be solved by first applying the Viterbi algorithm, and next maximizing the likelihood $P\left(O\middle|\lambda\right)$ over the single most-likely state sequence.

a. The Baum-Welch Re-Estimation Algorithm (B-W Algorithm)

This method starts with an initial arbitrary model λ and searches for a new model $\hat{\lambda}$ which improves the probability that the given observation sequence is generated by the new model, at each iteration until a maximum is reached, such that:

$$P\left(O\middle|\hat{\lambda}\right)\geq P\left(O\middle|\lambda\right). \tag{4.34}$$

This optimization criterion is known as the maximum likelihood criterion. Some new notations from [Rabiner, 1993; Dugad, 1996] are first introduced to explain the B-W re-estimation algorithm. The term $\gamma_t\left(i\right)$ denotes the conditional probability of being in state i at time t, given the full observation sequence $O=\left\{o_1,o_2,...,o_t,...,o_T\right\}$ and the model $\lambda\left(A,B,\pi\right)$, as defined in the following form:

$$\gamma_t\left(i\right)=P\left(q_t=i\middle|O,\lambda\right). \tag{4.35}$$

By using the definitions of the Forward variable $\alpha_t(i)$ from Eq. (4.12), the Backward variable $\beta_t(i)$ from Eq. (4.18) and the Bayes Rule, the following relation is obtained:

$$\gamma_t(i) = \frac{P\left(q_t = i, O \middle| \lambda\right)}{P\left(O \middle| \lambda\right)} = \frac{\alpha_t(i)\beta_t(i)}{P\left(O \middle| \lambda\right)}. \tag{4.36}$$

The conditional probability of being in state i at time t and making a transition to state j at time $t+1$, $\xi_t(i,j)$, is also defined, given the observation and the model, i.e.,

$$\xi_t(i,j) = P\left(q_t = i, q_{t+1} = j \middle| O, \lambda\right). \tag{4.37}$$

By using the forward and backward variables and the Bayes' Rule, the probability $\xi_t(i,j)$ can be written in the form [Rabiner, 1993]:

$$\begin{aligned}\xi_t(i,j) &= \frac{P\left(q_t = i, q_{t+1} = j, O \middle| \lambda\right)}{P\left(O \middle| \lambda\right)} \\ &= \frac{\alpha_t(i)a\left(j \middle| i\right)b\left(o_{t+1} \middle| j\right)\beta_{t+1}(j)}{P\left(O \middle| \lambda\right)} \\ &= \frac{\alpha_t(i)a\left(j \middle| i\right)b\left(o_{t+1} \middle| j\right)\beta_{t+1}(j)}{\sum_{i=1}^{S}\sum_{j=1}^{S}\alpha_t(i)a\left(j \middle| i\right)b\left(o_{t+1} \middle| j\right)\beta_{t+1}(j)}.\end{aligned} \tag{4.38}$$

The derivation involved in Eq. (4.38) can be seen intuitively in Figure 4.8, which illustrates how the probability $\xi_t(i,j)$ is computed. In this illustration, $\alpha_t(i)$ and $\beta_{t+1}(j)$ account for the partial observation sequences $\{o_1, o_2, ..., o_t\}$ and $\{o_{t+1}, o_{t+2,} ..., o_{T-1}, o_T\}$, respectively. For completeness, $a\left(j \middle| i\right)$ explains the transition from state i to state j and $b\left(o_{t+1} \middle| j\right)$ represents the probability of selecting the symbol o_{t+1} from state j. All these probabilities are multiplied to compute the numerator term in Eq. (4.38) because their respective events are statistically independent. Summation of these products over all possible states j and i accounts for the denominator term in Eq. (4.38).

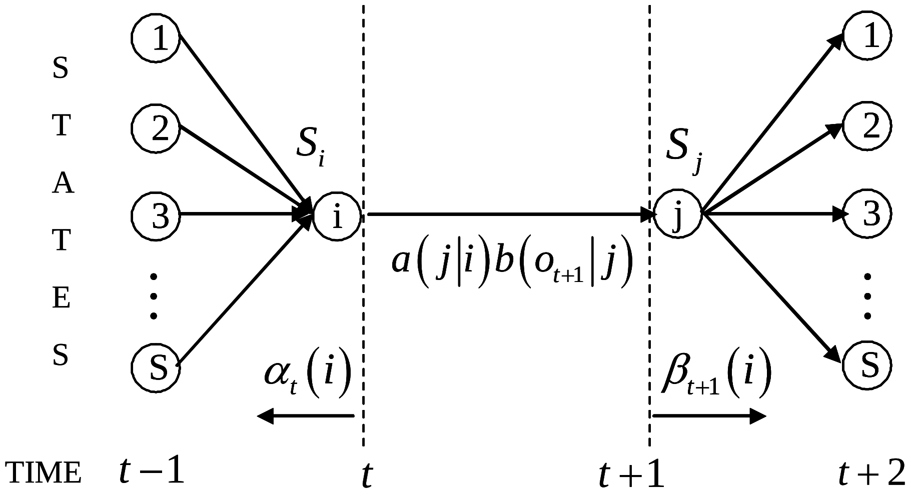


Figure 4.8. Illustration for the Computation of the Probability $\xi_t\left(i, j\right)$ (After [Rabiner, 1993]).

It should also be pointed out that $\gamma_t\left(i\right)$ is related to $\xi_t\left(i, j\right)$ by summation over j as shown below:

$$\gamma_t\left(i\right) = \sum_{j=1}^{S} \xi_t\left(i, j\right). \tag{4.39}$$

Summation of $\gamma_t\left(i\right)$ over the observation time t provides a useful quantity, the expected number of transitions from state i

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{ Expected number of transitions from state } i. \tag{4.40}$$

By way of similar reasoning, the expected number of transitions from state i to state j is obtained by summing $\xi_t\left(i, j\right)$ over the time index t

$$\sum_{t=1}^{T-1} \xi_t\left(i, j\right) = \text{ Expected number of transitions from state } i \text{ to state } j. \tag{4.41}$$

Now that all the underlying concepts have been discussed and their respective notations given, Eqs. (4.40) and (4.41) can be used in the Baum-Welch re-estimation formulas as follows [Rabiner, 1993]:

$$\begin{aligned}\hat{\pi} &= \text{ Expected number of times having state } i \text{ at time } t = 1. \\ &= \gamma_1(i), \;\; 1 \le i \le S.\end{aligned} \tag{4.42}$$

$$\begin{aligned}\hat{a}(j|i) &= \frac{\text{Expected number of transitions from state } i \text{ to state } j}{\text{Expected number of transitions from state } j} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t}.\end{aligned} \tag{4.43}$$

$$\begin{aligned}\hat{b}(k|j) &= \frac{\text{Expexted number of times in state } j \text{ and observing symbol } o_t = k}{\text{Expected number of times in state } j} \\ &= \frac{\sum_{\substack{t=1 \\ o_t=k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}.\end{aligned} \tag{4.44}$$

Based on the formulas given in Eqs. (4.42) through (4.43), an updated model $\hat{\lambda}$ is obtained until the new model satisfies the condition:

$$P\left(O\middle|\hat{\lambda}\right) - P\left(O|\lambda\right) \ge \varepsilon, \tag{4.45}$$

where the tolerance ε denotes a very small number.

However, the B-W Re-estimation algorithm improves towards the nearest maxima in the proximity of the initial model parameters owing to the complex nature of the joint likelihood function of the training set $P(O|\lambda)$ which has many local maxima. One way to circumvent this problem is to repeat the B-W re-estimation algorithm a few times with different random initial model parameters and select the best re-estimated model which returns the maximum $P(O|\lambda)$.

b. Viterbi Re-Estimation Algorithm

The Viterbi decoding algorithm has been applied to ascertain the optimal hidden state sequence so far. This algorithm can also be applied to the training problem to re-estimate the model parameters over the most-likely state sequence, unlike the B-W re-estimation algorithm which works over all possible state sequences.

In a left-to-right HMM set-up, the initial state can be designated as state one to save from re-estimating the initial state probabilities. First, the optimal state sequence Q^* is estimated and then the probability $P\left(O\middle|Q^*,\lambda\right)$ is evaluated. The following arguments are also tracked along the most probable path [Deller, 2000]:

$$n\left(j \mid i\right) = \text{ Number of transitions from state } i \text{ to state } j \text{ along the path } Q^*, \quad (4.46)$$

$$n\left(\bullet \mid i\right) = \text{ Number of transitions from state } i \text{ along the path } Q^*, \quad (4.47)$$

$$n\left(j \mid \bullet\right) = \text{ Number of transitions to state } j \text{ along the path } Q^*, \quad (4.48)$$

$$n\left(o_t = k, q_t = j\right) = \text{ Number of times observation } k \text{ and state } j \text{ occur concerrently along the path } Q^*. \quad (4.49)$$

The Viterbi re-estimation formulas are presented as follows:

$$\hat{a}\left(j|i\right) = \frac{n\left(j \mid i\right)}{n\left(\bullet \mid \mathrm{i}\right)}, \quad (4.50)$$

$$\hat{b}\left(k|j\right) = \frac{n\left(o_t = k, q_t = j\right)}{n\left(j \mid \bullet\right)}. \quad (4.51)$$

Note that although the Viterbi re-estimation algorithm seems simpler and more computationally-efficient than the B-W re-estimation, it does not incorporate all possible state sequences into the model update.

C. HMM IMPLEMENTATION ISSUES

This section discusses two important implementation issues for training and recognition of HMMs. First, there is a numerical underflow problem inherent in the computation of the B-W re-estimation formulas. Second, it is necessary that HMMs be trained by multiple observation sequences so as to obtain a more rigorous representation of the statistical variations inherent in the observations (or utterances of words for IWR purpose.).

1. Scaling the B-W Re-estimation Formulas

It has been demonstrated that the B-W re-estimation formulas given in Eqs (4.42) through (4.44) can be used to update the model parameters. A potential problem with the forward, backward variables and the B-W re-estimation formulas is that as the time index t increases, the large number of multiplications of probabilities between 0 and 1 may exceed the precision range of numerical computations, hence causing underflow problems. As a result, it becomes necessary to scale the forward and backward variables to circumvent this problem [Rabiner, 1993]. First, the re-estimation formula for $\hat{a}(j|i)$ must be rewritten in terms of the forward and backward variables by using Eqs. (4.38) and (4.39).

$$\hat{a}(j|i) = \frac{\sum_{t=1}^{T-1} \alpha_t(i) a(j|i) b(o_{t+1}|j) \beta_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{S} \alpha_t(i) a(j|i) b(o_{t+1}|j) \beta_{t+1}(j)}. \tag{4.52}$$

Next, the forward variable $\alpha_t(i)$ is scaled by the scaling coefficient c_t, that is:

$$c_t = \frac{1}{\sum_{j=1}^{S} \alpha_t(j)}, \tag{4.53}$$

$$\begin{aligned} \bar{\alpha}_t(i) &= c_t \alpha_t(i) \\ &= \frac{\alpha_t(i)}{\sum_{j=1}^{S} \alpha_t(j)}, \end{aligned} \tag{4.54}$$

where the notation $\bar{\alpha}_t(i)$ denotes the scaled version of the forward variable. Similarly, the backward variable is scaled and shown as

$$\bar{\beta}_t(j) = c_t \beta_t(j). \tag{4.55}$$

Note that the computations are kept within a reasonable dynamic range and the following scaled re-estimation formulas obtained by replacing the un-scaled forward and backward variables with the scaled variables in B-W re-estimation formulas, resulting in [Rabiner, 1993]:

$$\bar{a}\left(j|i\right)=\frac{\sum_{t=1}^{T-1}\bar{\alpha}_t\left(i\right)a\left(j|i\right)b\left(o_{t+1}|j\right)\bar{\beta}_{t+1}\left(j\right)}{\sum_{t=1}^{T-1}\sum_{j=1}^{S}\bar{\alpha}_t\left(i\right)a\left(j|i\right)b\left(o_{t+1}|j\right)\bar{\beta}_{t+1}\left(j\right)}, \tag{4.56}$$

$$\bar{b}\left(k|j\right)=\frac{\sum_{\substack{t=1\\o_t=k}}^{T-1}\bar{\alpha}_t\left(j\right)\bar{\beta}_t\left(j\right)}{\sum_{t=1}^{T-1}\bar{\alpha}_t\left(j\right)\bar{\beta}_t\left(j\right)}. \tag{4.57}$$

Furthermore, the probability $P\left(O|\lambda\right)$ can be computed in terms of the scaling factor c_t. The following property is introduced first [Rabiner, 1993]:

$$\prod_{t=1}^{T}c_t\sum_{i=1}^{S}\alpha_T\left(i\right)=1. \tag{4.58}$$

From Eqs. (4.15) and (4.58),

$$\prod_{t=1}^{T}c_t\cdot P\left(O|\lambda\right)=1, \tag{4.59}$$

and

$$P\left(O|\lambda\right)=\frac{1}{\prod_{t=1}^{T}c_t}. \tag{4.60}$$

By taking the negative logarithm on both sides of Eq. (4.60), the log likelihood function is computed given as [Deller, 2000]:

$$\log\left[P\left(O|\lambda\right)\right]=-\sum_{t=1}^{T}\log c_t. \tag{4.61}$$

Finally, it should be noted that the Viterbi decoding and re-estimation algorithms, as described in the previous sections, are not inherently susceptible to an underflow problem due to the logarithmic operations involved. Therefore, it is not necessary to scale the Viterbi algorithm.

The MATLAB implementation for the scaled B-W re-estimation algorithm is that available at [Sorensen, 2005] and given in Appendix C.1. The MATLAB implementation for the Viterbi training algorithm is provided in Appendix C.2.

2. Training with Multiple Observation Sequences

The training algorithms and the method of scaling the re-estimation formulas for HMM so far have been discussed, provided that there is only a single observation sequence of interest $O = \left\{ o_1, o_2, ..., o_T \right\}$ available. However, training a HMM using a single observation sequence is not practical in most real-world applications because it is possible to build a more rigorous model which encodes many statistical variations of the same event to be modeled by using multiple observations. Such a model is more likely to extract a set of characteristics relevant to the recognition of a particular word. Therefore, a modification to the re-estimation procedure is essential to account for multiple observations [Rabiner, 1993; Deller 2000].

Thus, the study uses a left-to-right model in which state transitions from state 1 at time $t = 1$ to state S at time $t = T$ occur sequentially, assuming that there are L observation sequences denoted as,

$$\tilde{O} = \left\{ O^{(1)}, O^{(2)}, ..., O^{(l)}, ..., O^{(L)} \right\}, \tag{4.62}$$

where $O^{(l)} = \left(o_1^{(l)}, o_2^{(l)}, ..., o_{T_l}^{(L)} \right)$ represents the l^{th} observation sequence. By assuming that all observation sequences are statistically independent, the joint probability of L observation sequences results in

$$P\left(O \middle| \lambda\right) = \prod_{l=1}^{L} P\left(O^{(l)} \middle| \lambda\right). \tag{4.63}$$

Recall that the re-estimation formulas are calculated by enumerating the occurrences for each event. Hence, the unscaled version of the re-estimation formulas for multiple observation sequences should add up the expected number of occurrences for each individual observation sequence as follows:

$$\tilde{a}\left(j|i\right)=\frac{\sum_{l=1}^{L}\frac{1}{P\left(O^{(l)}|\lambda\right)}\sum_{t=1}^{T_l-1}\alpha_t^l\left(i\right)a\left(i|j\right)b\left(o_{t+1}^{(l)}|j\right)\beta_{t+1}^l\left(j\right)}{\sum_{l=1}^{L}\frac{1}{P\left(O^{(l)}|\lambda\right)}\sum_{t=1}^{T_l-1}\alpha_t^l\left(i\right)\beta_t^l\left(i\right)}, \tag{4.64}$$

and

$$\tilde{b}\left(k|j\right)=\frac{\sum_{l=1}^{L}\frac{1}{P\left(O^{(l)}|\lambda\right)}\sum_{\substack{t=1\\ o_t^{(l)}=k}}^{T_l-1}\alpha_t^l\left(j\right)\beta_t^l\left(j\right)}{\sum_{l=1}^{L}\frac{1}{P\left(O^{(l)}|\lambda\right)}\sum_{t=1}^{T_l-1}\alpha_t^l\left(j\right)\beta_t^l\left(j\right)}. \tag{4.65}$$

Equation (4.60) is modified to obtain the relation between $c_t^{(l)}$, the scaling factor for the l^{th} observation and $P\left(O^{(l)}|\lambda\right)$ for the multiple-observation scenario in Eq. (4.66)

$$P\left(O^{(l)}|\lambda\right)=\left(\prod_{t=1}^{T_l}c_t^{(l)}\right)^{-1}. \tag{4.66}$$

Rewriting Eqs. (4.64) and (4.65) in terms of the scaled variables $\bar{\alpha}_t^l\left(i\right)$ and $\bar{\beta}_t^l\left(i\right)$ the scale factor $c_t^{(l)}$ and the term $\frac{1}{P\left(O^{(l)}|\lambda\right)}$ causes each other to be canceled out of the nominator and the denominator, due to the relation given in Eq. (4.66). Consequently, the modified re-estimation formulas in terms of scaled variables become:

$$\tilde{a}\left(j|i\right)=\frac{\sum_{l=1}^{L}\sum_{t=1}^{T_l-1}\bar{\alpha}_t^l\left(i\right)a\left(j|i\right)b\left(o_{t+1}^{(l)}|j\right)\bar{\beta}_{t+1}^l\left(j\right)}{\sum_{l=1}^{L}\sum_{t=1}^{T_l-1}\bar{\alpha}_t^l\left(i\right)\bar{\beta}_t^l\left(i\right)}, \tag{4.67}$$

$$\tilde{b}\left(k|j\right)=\frac{\sum_{l=1}^{L}\sum_{\substack{t=1\\ o_t^{(l)}=k}}^{T_l-1}\bar{\alpha}_t^l\left(j\right)\bar{\beta}_t^l\left(j\right)}{\sum_{l=1}^{L}\frac{1}{P\left(O^{(l)}|\lambda\right)}\sum_{t=1}^{T_l-1}\bar{\alpha}_t^l\left(j\right)\bar{\beta}_t^l\left(j\right)}, \tag{4.68}$$

and

$$\tilde{\pi}_i(1) = P(q_1 = i) = \frac{1}{L}\sum_{l=1}^{L}\frac{\bar{\alpha}^l(o_1, i)\bar{\beta}^l(o_2, i)}{c_1^{\,l}}. \tag{4.69}$$

Note that it is not necessary to re-estimate π_i in a left-to-right HMM structure by assuming $\pi_1 = 1$ and $\pi_i = 0$ when $i \neq 1$. Finally it should be pointed out that the same rationale behind the modification of B-W re-estimation formulas to account for multiple observation sequences also applies to the Viterbi re-estimation approach.

The MATLAB implementation for the HMM training with the multiple observations (speech feature vectors) is that available at [Sorensen, 2005] and presented in Appendix C.3.

D. ISOLATED WORD RECOGNIZER IMPLEMENTATION USING HMMS

Hidden Markov modeling was discussed, as well as the underlying assumptions of HMMs, the model parameters, their common structures and the basic implementation issues. Also presented were two popular training algorithms for estimating optimal model parameters in this chapter. From an acoustic signal modeling point of view, a HMM can be considered a combination of two stochastic processes. These are a hidden Markov Chain, which represents the temporal variability of an acoustic signal in the hidden state sequence, and an observable process which explains the spectral properties of an acoustic signal. Based on this background knowledge, an isolated word recognizer using discrete symbol HMM classification is now introduced, so as to identify seven distinct sets of acoustic signals collected from within the external ear canal.

The discrete symbol HMM works with a finite set of integer symbols from a codebook. The K-means clustering algorithm makes it possible to generate such a codebook from the feature vectors of a training data set. The discrete observation sequences are derived from the indices of the codebook by using the feature vectors of a designated word from the vocabulary. The goal is to identify the designated unknown words that these observation sequences represent, by using Hidden Markov modeling. A left-to-right HMM structure attempts to model the sequential pattern likely to be present within the symbols of a given observation set. Needless to say, there are temporal variations in the nature of speech utterances. Thus, it should be pointed out that a left-to-right model is more appropriate for isolated word recognition [Rabiner, 1993]. An additional common restriction on such a model structure is that no more than two-state jumps at a time are allowed in forward transitions. This is also known as a Bakis model [Deller, 2000].

Therefore, the model structure chosen for recognition is a left-to-right HMM with discrete observation symbols. It is also noteworthy to point out that even an ergodic HMM trained with speech utterances often fits into a sequential structure eventually with backward transition probabilities turning out to be zero [Deller, 2000].

Unfortunately, there is not a widely-accepted method to determine the proper size of the model, that is, the number of states necessary to encode the relevant characteristics of the observations into the model so as to result in the best recognition accuracy. Needless to say, the relationship between the number of states and the performance of HMM is crucial.

One of the basic linguistic units commonly used in Hidden Markov modeling is the phoneme. As a matter of fact, a phoneme is nothing more that an abstract unit that might have various acoustic forms from one context to another or sometimes from speaker to speaker due to the naturally occurring variations in the utterance.

Some researchers suggest the number of states be at least equal to the number of phonemes contained in the designated word. Some others report it is better to assign two or sometimes three states per phoneme taking into account the phoneme itself and the transitions to and from it [Deng, 2003; Becchetti, 1999]. In most real-world HMM applications, experimental results suggest the proper model size [Deller, 2000]. Selecting too small a number of states would result in poor classification due to the lacking capability in modeling properly. Also, it should be noted that large models suggest increased computational and memory cost. In the training phase, model sizes of five through eight states were experimented with to determine the proper model size. This study’s experimental results showed that eight states are sufficient to model the linguistic units of phonemes present in the vocabulary. The assumption is that the number of states should correspond roughly to the average number of distinct sounds (phonemes or syllabuses) in each word. Thus, an eight-state HMM was selected.

There are mainly two approaches to implement an isolated word recognizer using HMMs. The first technique models each word in a small vocabulary with a separate HMM. The second is more appropriate for larger vocabularies whose goal is to model subword basic units such as phonemes, diphones or triphones. In this method, all these models should be combined to account for the whole word. The former technique was chosen by modeling each one of the seven words in the vocabulary by a separate HMM as shown in Figure 4.9.

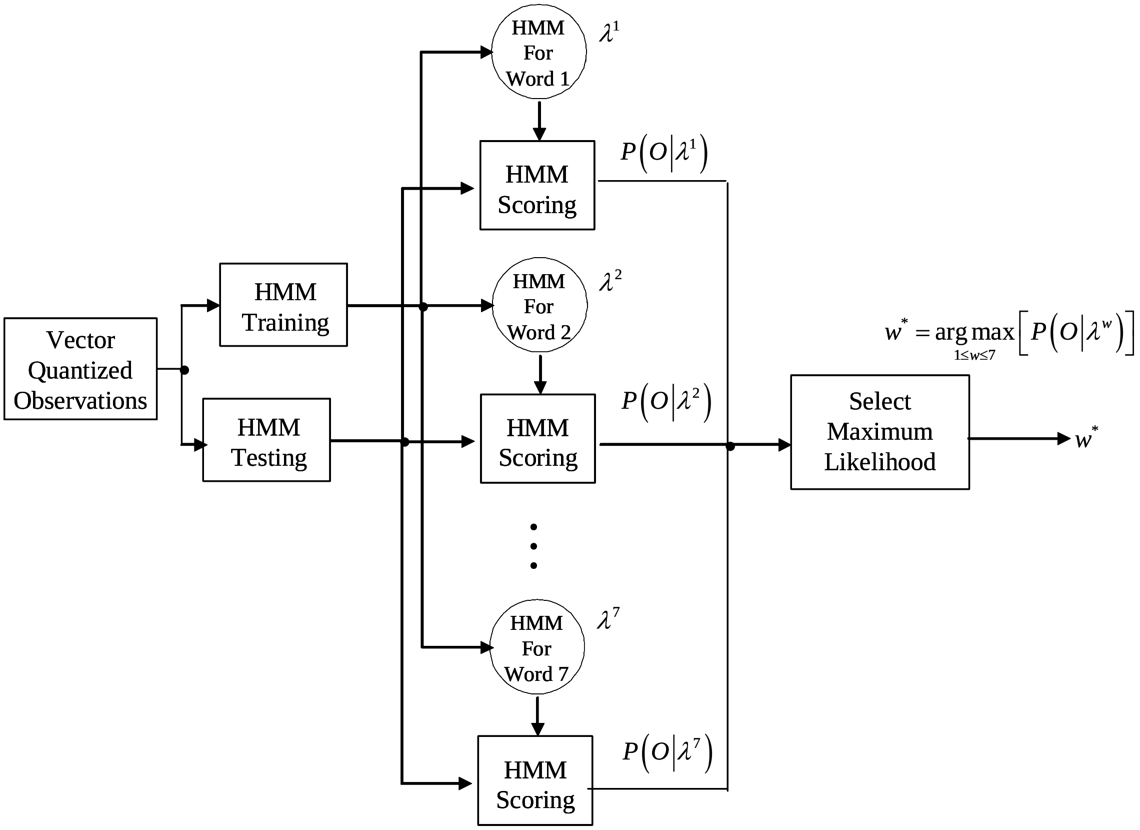

HMM For Word 1
λ^1
HMM Scoring
$P\left(O|\lambda^1\right)$
HMM For Word 2
λ^2
Vector Quantized Observations
HMM Training
HMM Testing
HMM Scoring
$P\left(O|\lambda^2\right)$
$w^* = \arg\max_{1\le w\le 7}\left[P\left(O|\lambda^w\right)\right]$
Select Maximum Likelihood
w^*
HMM For Word 7
λ^7
HMM Scoring
$P\left(O|\lambda^7\right)$

Figure 4.9. Block Diagram of the Isolated Word HMM Recognizer ([After (Rabiner, 1993)].

The procedure followed in the isolated word recognition may be divided into two steps [Rabiner, 1993]:

1. First, a separate HMM is built for each word in the vocabulary, where. each word model has the same number of states. This step involves the training phase to estimate the model parameters $\lambda^w = (A, B, \pi)$ using either the B-W re-estimation technique or the Viterbi re-estimation algorithm described in Section (IV.B.3).

2. Second, the trained HMMs are used to identify each unknown word in the testing data set. The recognition phase involves the computation of the likelihood $P\left(O\middle|\lambda^w\right)$ for all possible models given the observation sequence belonging to the designated unknown word. Specifically, the Forward-Backward recursion described in Section (IV.B.1) is used to calculate $P\left(O\middle|\lambda^w\right)$ for the trained seven models and then the index of the model with the maximum likelihood is identified as the designated word such that:

$$w^* = \underset{1\le w\le 7}{\arg\max}\left[P\left(O\middle|\lambda^w\right)\right]. \tag{4.70}$$

In training the models, both the Baum-Welch re-estimation and the Viterbi re-estimation algorithms were used for comparison. The Baum-Welch re-estimation algorithm outperformed the Viterbi re-estimation algorithm by resulting in slightly better recognition rates after the training. Therefore, the B-W re-estimation scheme, which uses multiple observation sequences as described in Section (IV.C.2), were chosen to train the models. This scheme implements training using scaled forward and backward variables to prevent underflow problems in numerical computations. Initial model parameters (A, B, π) are randomly chosen before training. Then, these models are trained until they have an optimized set of parameters, which produces the maximum likelihood $P\left(O\middle|\lambda^w\right)$ for the given observation sequence of the designated word w. It was observed that these models do not necessarily converge to the same parameters when the training for each model is repeated.

In recognition, the MATLAB implementations available at [Sorensen, 2005] were used. The MATLAB routine in the Appendix C.4. computes the log likelihood function for a given observation sequence and trained model parameters. The MATLAB routine in Appendix C.5. implements the HMM-based recognition by selecting the model which has the maximum log likelihood as the identified word.

The whole data set was consisted of 50 repetitions for each of the seven words $\{left, right, up, down, kill, pan, move\}$ spoken by 20 adult subjects. Two thirds of the data set was randomly selected for training and the remaining one third of the data was used for testing.

The next chapter starts with a brief overview of the entire isolated word recognition system, describing each step from the in-ear microphone data collection to the classification of the words under consideration. Finally, HMM-based average classification results are presented for the isolated word recognition system used in this study.

THIS PAGE INTENTIONALLY LEFT BLANK

V. HMM CLASSIFICATION RESULTS

This chapter presents a system overview of the speaker independent, small-vocabulary, isolated word HMM recognizer before discussing classification results.

A. SYSTEM OVERVIEW

The main goal of the study was to recognize a set of seven spoken words, in which the acoustic signals were collected within the external ear canal using an isolated word recognizer. The small vocabulary consists of seven words: {*left*, *right*, *up*, *down*, *move*, *pan*, *kill*} and the speech database was generated using 20 adult subjects who uttered each word 50 times by wearing the in-ear microphone discussed earlier. A discrete-symbol HMM recognizer was chosen because the vocabulary under consideration is small and consists of short words. The isolated word recognition system assumes that the speech signal is a realization of some message composed of basic subword units, e.g., phones (or phonemes), which can be considered as a sequence of symbols from a unique codebook. Figure 5.1 represents the overall block diagram of the isolated word HMM recognizer used in the study.

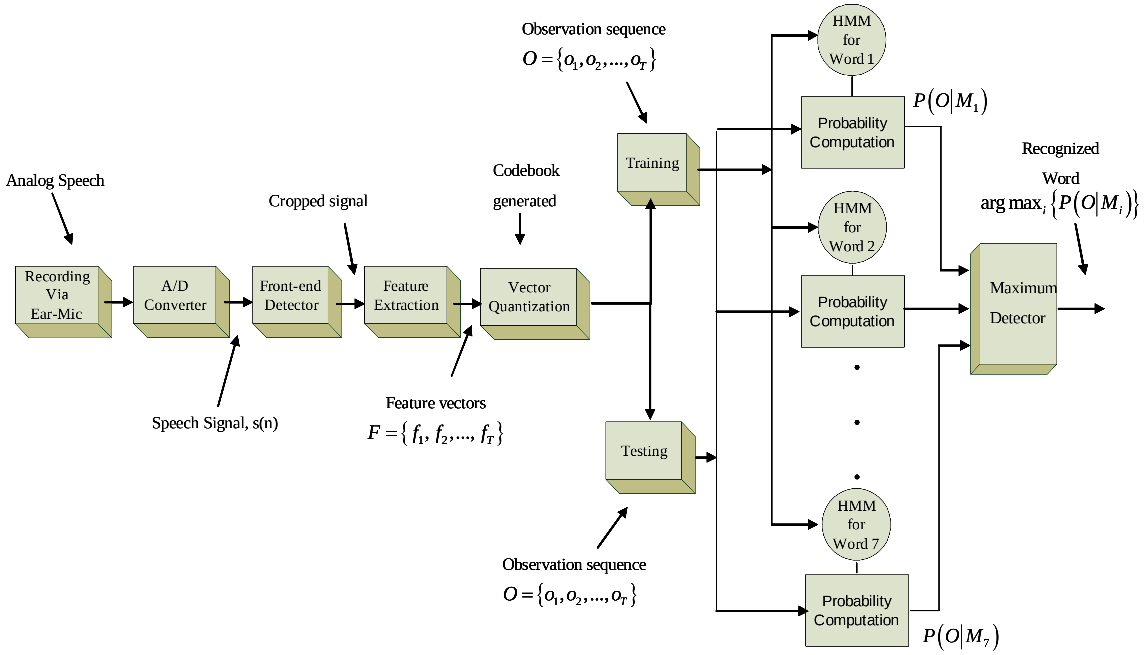


Figure 5.1. System Block Diagram of the Isolated Word HMM Recognizer (After [Sorensen, 2005].).

The IWR system includes the following six phases:

1. *Recording step*: Audio recordings containing the utterances were first collected using the in-ear microphone in an earlier study by [Newton, 2006] as described in Section I.C. Recall that a total of $50 \times 20 \times 7 = 7000$ isolated utterances of seven words were collected within the external ear canal. In addition, 1,000 isolated utterances of the word "Kill" were collected from the same 20 adult subjects within the external ear canal while a piece of loud music was played in the same room to simulate noisy environment conditions. This dataset was named "Kill_noise" in the database. Finally, 400 isolated utterances of the word "Right" were collected from the same adult subjects with the in-ear microphone placed outside the mouth to investigate potential distortions due to the ear canal. This dataset was named "Right_outside."

   An A/D converter digitized the speech recordings with 8 kHz sampling rate and 16-bit quantization scheme. Next, the digitized speech was transferred to and stored in a notebook PC via the PCMCI card (DAQ Card-Al-16E-4 by National Instruments).

2. *Front-end detection step*: One of the hardest challenges encountered in this study was to accurately extract the speech utterances from the audio. First, the mean of the digital signal was normalized to zero, so as to eliminate any bias introduced by the microphone. Second, the signal was passed through a high-pass filter with a passband of 100 Hz to 4000 Hz in order to remove equipment-related disturbances such as 50/60 Hz humming noise. Third, a search scheme based on short-time energy and zero-crossing rate was used to identify beginning and end of each utterance, as discussed earlier in Chapter II. A rectangular window was applied and the signal partitioned into 50% overlapping frames of 10 ms for short-term processing, as discussed in Chapter II. Note that a second detection algorithm using frame-based modified Teager's energy was also considered [Ying, 1993]. However, the latter scheme produced comparable results to the first detection scheme at a higher computational load, and as a result, was not used in the final speech endpoint detection phase. Finally, the speech portion of each signal was cropped and stored based on the estimated boundaries.

3. *Feature extraction step*: Feature extraction was performed to obtain the spectral and temporal characteristics of the speech waveform in a compact representation. First, the signal was divided into frames of 256 samples (corresponding to 32 ms) with an overlap of 100 samples (roughly corresponding to 40% overlapping) from frame to frame and a Hamming window applied to each frame. Two types of speech feature were considered: LPC-derived Cepstral Coefficients (LPC-CCs) and Mel-frequency Cepstral Coefficients (MFCCs). Results conducted on a small portion of the database showed classification results obtained with MFCC parameters were better than those obtained with the LPC-CC parameters, thus the preference was to use MFCC parameters as speech features for

the study. A filter bank of 24 triangular filters was used for the mel-scale conversion in the MFCC computation. The first 12 MFCC (static parameters) and 12 delta-MFCC (dynamic parameters) parameters were extracted from each frame of the cropped speech signal. Finally, the mean of the static MFCC parameters was normalized to zero and the delta-MFCC parameters were scaled to the range of the static MFCC parameters by multiplying them by a factor of six.

4. *Vector quantization (VQ) step*: Two thirds of the database was selected randomly to form the training set, while the reminder was used as the testing test. Feature vectors were generated from the training set and used to generate a codebook of length equal to 128 with the K-means clustering, as discussed in Chapter III. Note that a randomly generated codebook was also considered in an effort to significantly reduce the computational load by eliminating the K-means algorithm training phase. Kinnunen et al. reported that there is “no clusters” in the speech feature space based on the Euclidean distance metric and the cepstrum-based parameters (RCC, MFCC and LPCC) [Kinnunen, 2001]. This research study concluded that the role of vector quantization is only to remove redundancy from speech features and detecting clusters does not play a key role in codebook generation. Therefore, any fast method of choosing uniformly-spaced linear code vectors among the sample feature vectors should establish a codebook and work out for speech recognition. Motivated by the findings from [Kinnunen, 2001] a random codebook was considered using 256 symbols uniformly spaced in the range of speech features. However, experimental results showed that this linear codebook resulted in poor classification performances, and was not considered further. Note that our results did not agree with the findings in that earlier paper by [Kinnunen, 2001]. It is surmised that the differences could arise from the in-ear microphone data characteristics due to low-pass filtering in the ear canal. Both training and testing feature sets were vector-quantized and provided inputs to the DHMM classifier set-up, as explained next.

5. *Discrete-symbol Hidden Markov Model training step*: A discrete-symbol HMM was selected with eight states for each word in the vocabulary. The training sequence of vector-quantized symbols was used to train each model and the model parameters that maximize the likelihood of the training set were estimated using the Baum-Welch re-estimation, as discussed in Chapter IV. Also applied was the Viterbi training algorithm for comparison and through experiments it was discovered that the Baum-Welch re-estimation algorithm resulted in better classification rates than those obtained with the Viterbi algorithm for the data considered.

6. *Hidden Markov Model recognition step*: Recall that endpoint detection, feature extraction and vector quantization were performed for each word in the testing set, as illustrated in Figure 5.1. Once model parameters are determined from the training set, the recognizer is designed to decide that unlabelled data is most likely to have come from a specific model M_i by

selecting the model which maximizes the a-posteriori probability that the data was generated by the given model, i.e.,

$$\arg\max_i \left[P\left(O \middle| M_i\right) \right] \text{ for } i = 1, 2, ..., 7. \quad (5.1)$$

That is, the recognizer identifies the model, which is most-likely, given the discrete observation sequence.

Next, the overall HMM classification results obtained with the data under consideration are presented.

B. DHMM RECOGNIZER CLASSIFICATION RESULTS

Eighty experiments were conducted to investigate the overall classification results obtained with a DHMM classifier approach on the database, where two thirds of the data were randomly selected for the training phase and the reminder one third of the data were used for testing. Several models were considered by varying the frame length, the number of MFCC coefficients, and implementations with and without the delta-MFCC coefficients. The combination leading to the best average classification results is shown below:

a.	Frame length:	256 samples.
b.	Increments:	156 samples (40% overlapping frames).
c.	Speech features:	12 MFCCs and 12 delta-MFCCs.
d.	Mel-scale filter bank:	24 triangular filters.
e.	Delta-MFCC scaling factor:	6.
f.	Codebook size:	128 symbols.
g.	Model size of HMMs:	8 states.

First, the overall average classification results for the isolated words under consideration are discussed. Second, the experimental findings for the second dataset (Kill_noise and Right_outside) are presented to show the recognition performance in noisy environment conditions and the potential distortions due to the ear canal investigated. Finally, the timing issues regarding the simulation run-times were examined.

1. Overall Recognition Results

Results show the overall classification rate to be equal to 92.77%. Table 5.1 shows the confusion matrix of the average classification results obtained on the testing sets over 80 experiments. Figure 5.2 plots the average classification rates obtained for each word.

WORD	**AVERAGE CLASSIFICATION RATES FOR THE TESTING SET (%)**						
	Up	**Down**	**Left**	**Right**	**Kill**	**Pan**	**Move**
Up	**92.5500**	3.2125	2.3375	0.4375	0.3458	0.8208	0.2958
Down	0.3042	**92.3333**	1.7958	0.3250	2.3292	2.3500	0.5625
Left	2.1375	1.6875	**87.6833**	7.0792	0.6542	0.3625	0.3958
Right	0.4875	1.0750	2.6417	**94.6708**	0.4000	0.3792	0.3458
Kill	0.3125	1.7250	0.7917	0.4667	**94.2292**	2.2500	0.2250
Pan	0.0917	3.5500	0.9375	0.3375	1.8708	**91.9458**	1.2667
Move	0.6875	0.1000	1.4542	1.2875	0.2917	0.2167	**95.9625**
Average Classification: 92.77%							

Table 5.1. Confusion Matrix for the Average Classification Rates for Testing Sets; 80 Experiments, 62.5% Training, 37.5% Testing.

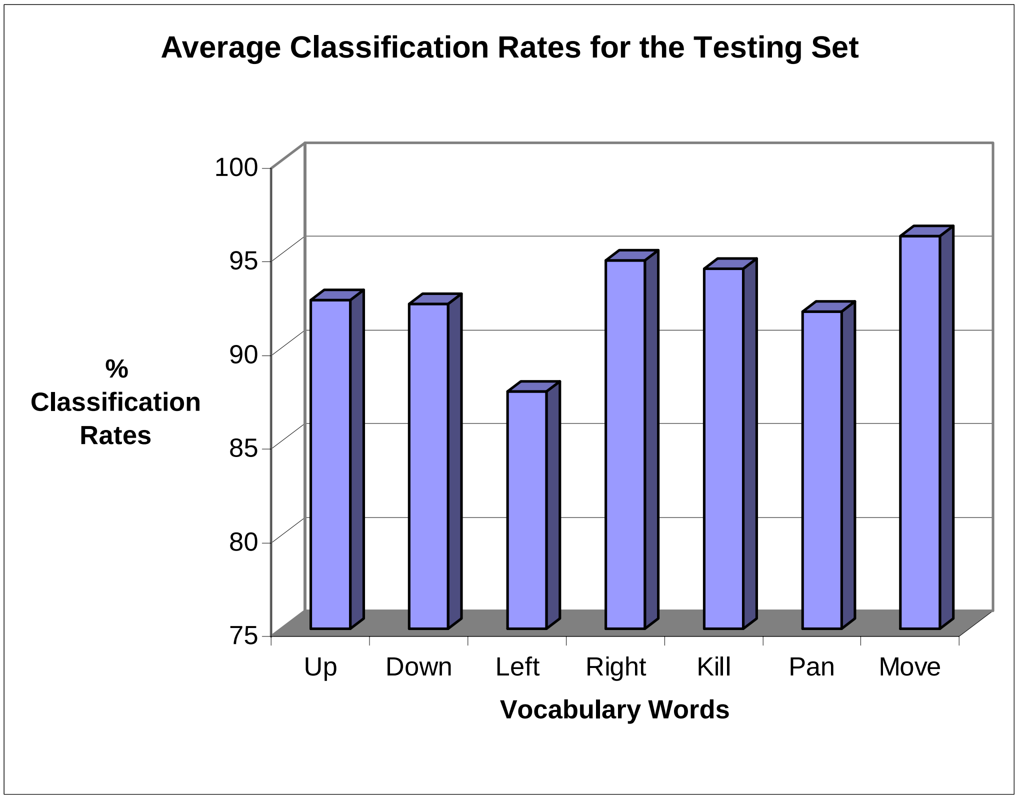


Figure 5.2. Average Classification Rates for Testing Sets; 80 Experiments, 62.5% Training, 37.5% Testing.

Table 5.2 lists the resulting 95% confidence level intervals for the averaged classification results obtained for 80 experiments on testing sets. Figure 5.3 illustrates the 95% confidence level intervals and the average classification rates for the testing set.

WORD	95% CONFIDENCE INTERVALS FOR TESTING SETS, in %
Up	[86.0000 - 96.0000]
Down	[86.0000 - 95.3333]
Left	[79.3333 - 93.3333]
Right	[92.3333 - 97.0000]
Kill	[91.0000 - 97.3333]
Pan	[86.6667 - 96.0000]
Move	[93.0000 - 98.0000]
Overall Classification	[91.0952 - 94.2857]

Table 5.2. 95% Confidence Level Intervals for Testing Sets; 80 Experiments.

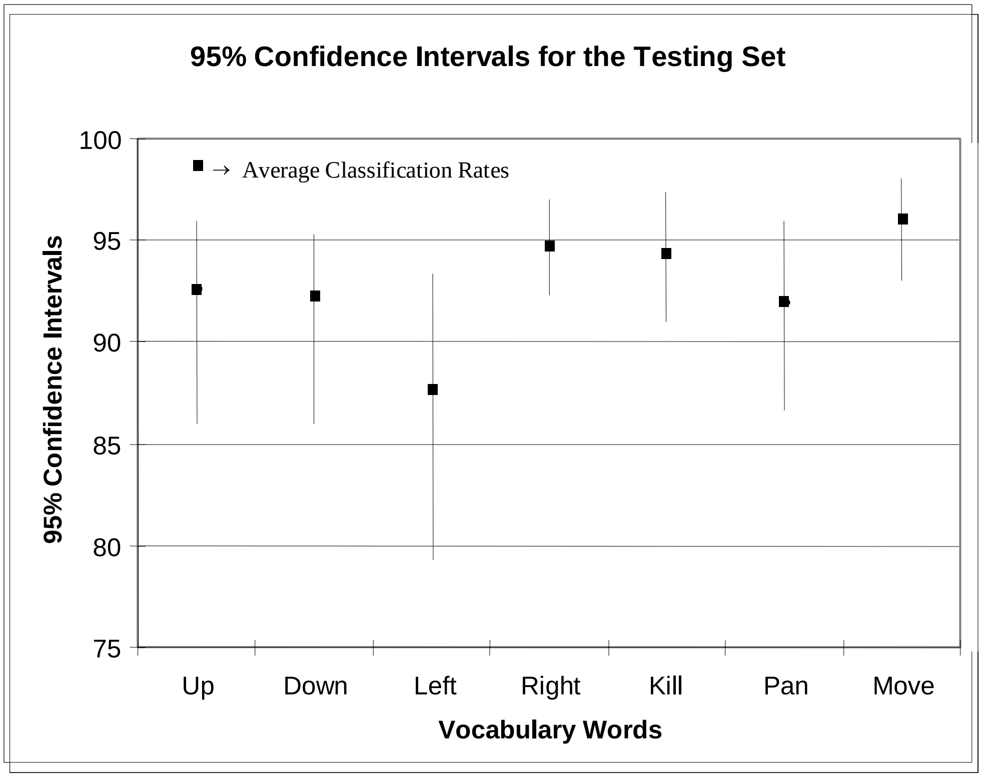


Figure 5.3. 95% Confidence Intervals and the Average Classification Rates for Testing Sets; 80 Experiments.

Figure 5.2 shows that the misclassification rate (or the word error rate) of the word "Left" stands out as the worst among all words. In an effort to find the cause of the performance degradation for this word, the misclassified "Left" utterances were traced back. We noted that misclassification errors occurred for trials where ending boundaries did not include the low energy "t" sound which was supposed to be present at the end of the word "Left." As a result, we surmise these errors were mostly caused by the incorrect detection of the speech boundaries, due to the hard-to-detect unvoiced low-energy ending sound "t" in the word "Left."

2. Experimental Results for the Second Dataset

Eighty experiments were also run with the second data set consisting of 1,000 utterances of "Kill_noise" and 400 utterances of "Right_outside." Table 5.4 shows the average classification rates for the second data set while Table 5.5 lists the 95% confidence level intervals for the second data set.

WORD	**AVERAGE CLASSIFICATION RATES FOR THE SECOND DATA SET (%)**						
	Up	**Down**	**Left**	**Right**	**Kill**	**Pan**	**Move**
Right_outside	0.8747	4.1858	9.5897	**83.5146**	0.8969	0.9192	0.0191
Kill_noise	2.0082	5.3058	4.1641	2.6784	**79.2431**	6.1595	0.4409

Table 5.4. Average Classification Rates for the Second Data Set; 80 Experiments.

WORD	**95% CONFIDENCE INTERVALS FOR THE SECOND DATA SET**
Right_outside	[63.1043 - 94.6565]
Kill_noise	[72.6809 - 84.5056]

Table 5.5. 95% Confidence Level Intervals for the Second Data Set; 80 Experiments.

Figure 5.4 illustrates the 95% confidence level intervals and the average classification rates for the second data set consisting of the words "Right_outside" and "Kill_noise". Note that hidden Markov models were trained with the first data set of words $\{up,\ down,\ left,\ right,\ kill,\ pan,\ move\}$ and tested on the second data set in these experiments.

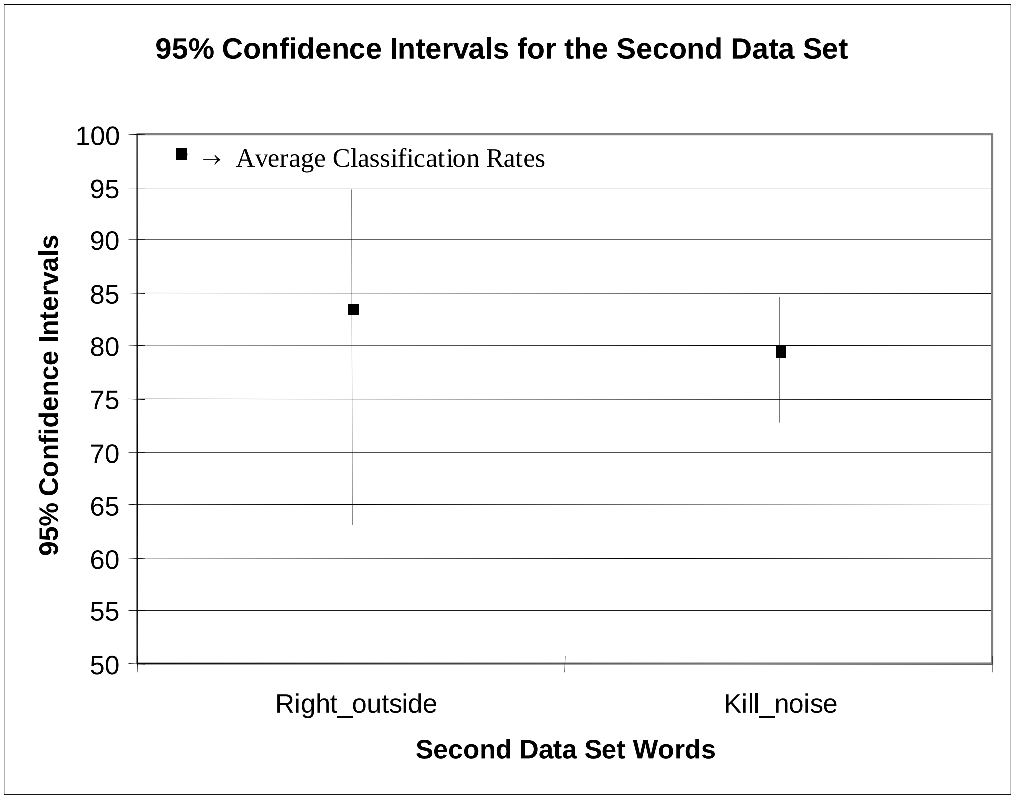


Figure 5.4. 95% Confidence Intervals and the Average Classification Rates for the Second Data Set; 80 Experiments

Recall that the isolated utterances of the words “Right_outside” and “Kill_noise” were not included in the training population which was used to estimate the model parameters. The following comments can be made.

1. Average classification rates for the word “Kill” collected within the ear canal while the subject is placed in a severely distorted environment show the ear-canal and the foam casing noise limit recognition degradations over what would be observed from the same signal collected in front of the mouth. However, they also indicate that recognition performance still worsens, thereby showing the need for some noise canceling pre-processing to minimize these noise distortions prior to the recognition stage.
2. Average classification rates obtained for the word “Right” collected outside the mouth shows significant performance degradation (down to 63%), over that observed with the same word collected from within the ear canal (down to 86%). Recall that the ear canal dampens high frequencies, as shown in Figure 2.1 which presents spectrograms of two trials of the word “Right” collected within the ear canal and outside the mouth, respectively. As a result, the HMM model derived for the data with dampened higher frequencies does not accurately fit the data collected outside the mouth, resulting in degraded recognition performances.

3. Timing Issues

Table 5.3 shows the average training times per utterance of a word for the following four computational tasks which are part of the classifier set-up: 1) the K-means clustering step which transforms the continuous valued feature vectors into a codebook, 2) the Baum-Welch re-estimation algorithm step which is applied to compute the HMM parameters, 3) the overall average training time, and 4) the average time needed to make a decision for a word from the testing set. The specifications shown are based on average MATLAB simulation run-times obtained with a PC with 3 GHz Intel Pentium IV processor and 1 GB of RAM. Table 5.3 shows that the K-means clustering algorithm step represents more than 50% of the overall training time needed to generate the codebook, while the computational time of the Baum-Welch re-estimation algorithm step needed to train one of the HMMs is about 44% less. Finally, we note that the recognition time on a per word basis is about 1/3 of that required at the training phase.

SIMULATION TYPE	AVERAGE MATLAB SIMULATION RUN-TIME (seconds)
K-means algorithm training time per word:	0.0771
Baum-Welch algorithm training time per word:	0.0431
Overall training time per word:	0.1202
Recognition time per word:	0.0428

Table 5.3. Average Simulation Run-Times for 80 Experiments.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis study extended the earlier work by Newton [Newton, 2006] who collected a database of isolated word utterances of seven words from 20 adult subjects by using the in-ear microphone. The objective of this thesis was to study this database and demonstrate the feasibility of using acoustic speech signals collected within the external ear canal for isolated word recognition. The intent was also to investigate the performance and noise-robustness of the in-ear microphone data for speech recognition in noisy operational environments. These objectives were successfully accomplished by implementing a discrete-observation HMM-based isolated word recognizer in MATLAB.

A. SIGNIFICANT RESULTS AND CONCLUSIONS

The following specific conclusions were drawn through experimental results:

- Three types of digital filters: a first-order pre-emphasis filter (FIR), a sixth-order bandpass filter (IIR) and a sixth-order high-pass filter (IIR) were implemented and applied to the database in the pre-processing step. The use of high-pass filter resulted in the best classification performances in the isolated word HMM recognizer.

- Two different speech segmentation procedures for endpoint detection were studied. The first approach used the short-term energy (STE) and the zero-crossing rate (ZCR), while the second was based on the frame-based modified Teager energy. Results showed similar detection accuracies were obtained for both. The Teager energy approach was found to be computationally more expensive than the alternative, and as a result, it was not used in the detection/cropping step. However, we also noted the *STE and ZCR detection approach* still produced misdetection of word boundaries for some of the words with weak fricative sounds.

- Two popular types of speech features, specifically LPC-derived Cepstral coefficients (LPC-CC) and Mel-frequency Cepstral coefficients (MFCC) were extracted from speech frames separately and their performances compared. The use of MFCC parameters for feature extraction resulted in better recognition rates. In addition, augmentation of the static MFCC parameters by the dynamic delta-MFCC parameters improved recognition rates. Finally, we also noted that Cepstral mean correction and delta parameters scaling had a significant impact for feature enhancement, improving the classification accuracy further.

- K-means clustering was implemented for codebook generation and vector quantization of continuous valued speech features. We observed that increasing the codebook size diminished the quantization distortion. However, large codebook sizes (over 128 codewords) were not practical for this study’s small-vocabulary recognition system, considering the increased training time associated with a larger codebook. We also investigated the use of a linear codebook uniformly spaced along the range of speech features, which resulted in poor clustering and classification. It was observed that the K-means clustering algorithm with the selection of a sufficiently-large codebook size can be effectively used to obtain discrete speech feature sequences for training the discrete-observation HMMs.

- The selection of proper HMM parameters such as the number of discrete symbols (or equivalently codebook size) and the number of hidden states to set up the HMM plays a key role in recognition performance. Although there are only four phonemes per word in this study’s vocabulary on average, experimental results showed the state size of eight was required to model the speech characteristics of in-ear microphone data for high classification accuracy.

The average classification rate for this study’s small-vocabulary, multiple-speaker isolated word HMM recognition system was 92.77%. Results show that the acoustic signals originated from speech organs and collected within the external ear canal via an in-ear microphone can be effectively used for isolated word recognition.

The second dataset collected under low SNR conditions with additive noise resulted in 79.24% recognition accuracy in the HMM-based classifier. This result is consistent with the theoretical noise-shielding property of the human ear, thus justifying the use of in-ear microphone data for speech enhancement and improved recognition under low-SNR conditions.

Finally, we investigated using the in-ear microphone outside the mouth and comparing classification results obtained for the data collected within the ear canal and outside the mouth. Average classification rates obtained for the data collected outside the mouth shows significant performance degradation (down to 63%), over that observed with the data collected from within the ear canal (down to 86%). Recall that the ear canal dampens high frequencies. As a result, the HMM model derived for the data with dampened higher frequencies does not accurately fit the data collected outside the mouth, resulting in degraded recognition performances.

These initial results are considered highly encouraging for the development of a human-machine interface using the in-ear microphone for the tele-operation of military robots in noisy operational environments.

B. RECOMMENDATIONS FOR FUTURE WORK

Results indicate that there is still room for recognition rate improvements. Four main areas for further work are identified.

First, the current database may be expanded to include additional speakers and the vocabulary size may be increased to include possible voice commands considering the development of a human-computer interface under consideration. In addition, a "clean" database needs to be collected inside a noise-controlled room to more formally investigate the impact of noise on the in-ear microphone data. Next, a set of standard noise databases with known SNR levels should be added synthetically to distort the data in order to investigate the performance degradation under known SNR levels.

Second, it was observed that some word errors, especially for the word "Left" in the vocabulary, were due to the misdetection of speech endpoints. Further improvement should be possible with an improved speech segmentation algorithm development. A possible alternative segmentation algorithm may include entropy-based measures to locate the speech boundaries in noisy environments better.

Third, speech features other than the MFCC and LPC-CC may be studied in an effort to determine the proper set of features for the in-ear microphone data. An alternative to the MFCC parameters is the use of Perceptual Linear Prediction (PLP) coefficients. Further, to augment the spectral speech features an energy term such as the log of frame-based speech energy can be appended to the speech parameters. The use of second order differential coefficients (delta-delta parameters) may also be investigated.

Finally, performance improvement may be achieved by implementing different model structures. The current isolated word recognizer was a simple discrete-observation HMM set-up because discrete systems require low run-time computation. However, vector quantization reduces the accuracy by introducing some distortion. Therefore, a continuous-density HMM structure using Gaussian mixture densities could be selected for investigation. Another limitation of the current HMM set-up is due to the first-order Markov chain assumption which provides a temporal structure for modeling the time evolution of speech spectral characteristics from one state into the next by state transitions. Ideally, the observation vectors are assumed to be independent and identically distributed (IID) within each state. The IID assumption suggests that there is no correlation between successive observation vectors (or speech feature vectors), which is not strictly true in terms of basic linguistic units. Therefore, a higher-order model structure with discrete observations could also be investigated.

Alternatively, the incorporation of state duration densities into the HMMs, applying the error corrective training method or Bayesian Network adaptation might initiate a wide range of research topics and improve system performance.

APPENDIX A. SPEECH FRONT-END DETECTION MATLAB PROGRAMS

This appendix includes the MATLAB programs used in the front-end detector of the isolated word recognition system. These programs determine the endpoints of speech boundaries and crop speech signals based on the detected endpoints. References show the specific endpoint detection algorithms implemented in the codes.

1. SPEECH ENDPOINT DETECTION FROM THE SHORT-TERM ENERGY AND ZERO-CROSSING RATE

```
%   Filename              : endpoint1.m
%   Written by            : R.S. Kurcan, LTJG. TU NAVY.
%   Date last revised     : 10\18\2005.
%   Purpose               : This function reads in a speech file and applies  an endpoint
%  detection algorithm based on Short-term Energy and Zero-crossing Rate to determine
%  the speech boundaries.
%
function [datacropped,speechlength,speechflag] = endpoint1(datadir,fs,win,fl,inc);
%
%   Endpoint Detection Using Short-term Energy and Zero-crossing Rate.
%  <Inputs>
%       datadir : Directory of the txt file containing speech signal.
%       fs      : Sampling frequency in Hz (Default 8000).
%       win     : Window type 'R' Rectangular window in time.
%                       'N' Hannning window in time.
%                       'H' Hamming window in time.
%       fl      :Frame length (Default 80 samples corresponding to 10 ms).
%       inc     :Increment in num of samples (Default 40).
%  <Outputs>
%       datacropped     : This returns the cropped speech signal based on endpoint
% detection.
%       speechflag      : If any speechflag is set for warning  any single one or
% combination of the following problems might have occurred.
%                   1. Split data file may be corrupt if norm(s)<0.1
%                   2. Infinite loop in raw search algorithm for speech
% boundaries if there is no signal detected that has 20 frames or longer
% duration.
%                   3. Speech portion of the signal starting too early
% might indicate a possible noise sequence in the beginning due to
% bad recording.
%       All above cases need further investigation after cropping speech to determine if
% the cropped data is corrupt.
%        speechlength   : It is the length of the cropped speech signal.
%
```

```
%  <References>
% [1] L. R. Rabiner and M. R. Sambur, “An algorithm for determining the endpoints
%       of isolated utterances,” The Bell System Technical Journal, February 1975.
%
%-----------------------------------------------------------------------------------------------------

clc, clear('s');
if nargin < 2 fs = 8000; end
if nargin < 3 win = 'R'; end     % Default Rectangular Window.
if nargin < 4 fl = fs/100; end   % Default frame length is 80 for 10 ms.
if nargin < 5 inc = fl/2; end    % Default inc.in overlapping window is 40.
if win == 'R'
   % Default Rectangular Window of length 80 for 10 ms frames.
   w = rectwin(fl);
   else if win =='N' w = hann(fl);
      else if win =='H' w = hamming(fl); end
      end
end

s = load(datadir);
sm = s-mean(s);                  % Takes the mean out of data before processing.
speechflag = 0;

% IIR Elliptical BPF Design
%d = fdesign.bandpass(100/4000,150/4000,2000/4000,2400/4000,60,0.5,60);
%hd = ellip(d);
%sf = filter(hd,sm);              % Filters the speech data.

% Design of a first-order pre-emphasis filter
%a = 1;
%b = [1, -15/16];
%sf = filter(b,a,sm);

% IIR Elliptical HPF Design
d = fdesign.highpass('n,fst,fp,ap',10,60/4000,100/4000,0.5);
hd = ellip(d);
sf = filter(hd,sm);

% Checking bad split trials below if any.
if norm(sf) < 1.5; datacropped = []; speechflag = 1; speechlength = 0;
else

% Truncating the data to make it divisible by frame length.
m = floor(length(sf)/inc);
sf = sf(1:m*inc);
```

```
% Short-term energy per frame is STE and Zero crossing count
% per frame is ZCR. % 50 overlaping frames taken by default.
for n=1:m-1;
   sw = sf(inc*(n-1)+1:inc*(n-1)+fl).*w; STE(n) = sum(abs(sw));
   for  j= 2:fl;
      zc(j)=abs(sign(sw(j))-sign(sw(j-1)))/2;
   end
   ZCR(n) = sum(zc);
end

% Assuming there is no speech during the first 100 ms of recordings,
% Mean and standard deviation of ZCR and STE for the first ten frames.
avgzcr = mean(ZCR(1:10)); stdzcr = std(ZCR(1:10));
avgste = mean(STE(1:10)); stdste = std(STE(1:10));

% Setting up STE Upper and Lower Thresholds and ZCR Threshold.
IF = fl/4;          % 20 crossings per 10 ms is a fixed value when N=80;
IZCT = min(IF,avgzcr+stdzcr);           % Zero-crossing Rate Threshold.
IE = 0.25;          % Upper level for avgste in case high noise
% present in first 20 frames.
minste = min(IE,avgste+stdste);
ITL = 8*minste;                        % Lower threshold for STE.
ITU = 32*minste;                        % Upper threshold for STE.
FT = (ITU-ITL)/2 + ITL;                % Fine threshold for STE.

% Following algorithm firsts makes a raw search for endpoint detection
% based on STE, ITU and UTL. Then it refines the speech boundaries using
% ZCR and IZCT for the successive and preceding 6 frames forth and back.
% If the interval btwn rawstart and rawend is less than 100 ms (20 frames)
% algorithm assumes that it is just a click noise (or a spike), not speech.

duration = 0; rawend = 0; rawstart = 0; loopn =0; d = 1; refindex = 0;
while duration < 21;       % Raw search for speech boundaries starts.
   for n = d:m-1
      if STE(n) > ITU;
         refindex = n; rawstart = n;
         for l = refindex:-1.0:1
            if STE(l) < FT; rawstart = l; break; end
         end
   break;
% If STE of speech signal does not exceed upper threshold, ITU is set to ITU = ITU/2
% and makes one more search.
      else if n == m-1;
         ITU = ITU/2; FT = FT/2;
         for v = 1:m-1
            if STE(v) > ITU;
```

```
               refindex = v; rawstart = v;
                 for p = refindex:-1.0:1
                    if STE(p) < FT; rawstart = p; break; end
                 end
         break;
            end
         end
      end
      end
   end
   if refindex ~= 0;
   for k = refindex:m-1
      if STE(k) < FT; rawend = k; break; end
   end
   end
   duration = rawend - rawstart; % If duration < 20 frames, it keeps scanning.
   if (duration == 0); speechflag =2;  break; end
   if (duration < 0); speechflag =3;  break; end
   loopn = loopn+1;
   if loopn > 10; speechflag = 4; speechlength = 0; break; end  % Prevents infinite loop.
   d = rawend+1;
end

% Fine search using total number of intervals crossing threshold of ZCR.
finestart = rawstart; fineend = rawend;

if (duration>23)&&(duration<50)
nzc = 0; update = 0;
if (rawstart-6) > 0;
for n = rawstart:-1:(rawstart-6)
   if ZCR(n) > IZCT; nzc = nzc+1; update = n; end
end
if nzc > 3; finestart = update; end
end

nzc = 0; update = 0;
if ((rawend+6) < length(ZCR))&&(rawend ~= 0);
for n = rawend:(rawend+6)
   if ZCR(n) > IZCT; nzc = nzc+1; update = n; end
end
   if nzc > 3; fineend = update; end
end
end

starti = inc*(finestart)+1;
endi = inc*(fineend);
```

```
if (endi-starti) > 878;
speechlength = endi-starti;
datacropped = sf(starti:endi);
speechflag = 0;
else speechlength = 0; datacropped = sf;
end

% Plots of STE, ZCR, and original speech signal, thresholds and resulting
% endpoints on the speech signal.
%figure(1);
%subplot(3,1,1); plot(STE); hold on,
%subplot(3,1,1), plot(1:length(STE),ITU,'r'), hold on;
%subplot(3,1,1), plot(1:length(STE),ITL,'g');
%title('Short-term Energy'); ylabel('Magnitude'); xlabel('Frame Number');

%subplot(3,1,2); plot(ZCR); hold on;
%subplot(3,1,2); plot(1:length(ZCR),IZCT,'r');
%title('Zero-crossing Rate'); ylabel('Zero-crossings');
%xlabel('Frame Number');

%subplot(3,1,3); plot(s); hold on;
%subplot(3,1,3); plot(starti,-2:0.1:2,'r'); hold on;
%subplot(3,1,3); plot(endi,-2:0.1:2,'r');
%title(datadir); ylabel('Magnitude');
%xlabel('Sample Number');

%figure(2);
%subplot(2,1,1); plot(s(starti:endi)); title('Original Speech Cropped');
%subplot(2,1,2); plot(sf(starti:endi)); title(datadir);   % filtered speech cropped.

end

%-----------------------------------------------------------------------------------------------------
% End of function endpoint1.m
%-----------------------------------------------------------------------------------------------------
```

2. SPEECH ENDPOINT DETECTION FROM THE FRAME-BASED MODIFIED TEAGER ENERGY

```
%   Filename                    : endpoint2.m
%   Written by                  : R.S. Kurcan, LTJG. TU NAVY.
%   Date last revised           : 11\08\2005.
%   Purpose                     : This function reads in a speech file and applies an endpoint
%  detection algorithm based on the frame-based modified Teager's energy to determine
%  the speech boundaries.
%
function [datacropped,speechlength,speechflag] = endpoint2(datadir,fs,win,fl,inc);
```

```
% Endpoint Detection Using Frame-based Modified Teager’s Energy.
% <Inputs>
%       datadir   :Directory of the txt file containing speech signal.
%       fs        :Sampling frequency in Hz (Default 8000).
%       win       :Window type 'R' Rectangular window in time.
%                       'N' Hannning window in time.
%                       'H' Hamming window in time.
%       fl       :Frame length (Default 80 samples corresponding to 10 ms).
%       inc      :Increment in num of samples (Default 40).
% <Outputs>
%       datacropped     :it returns the cropped speech signal based on endpoint
% detection.
%       speechflag       :If speechflag is set to 1 it returns a warning regarding
% any single one or combination of the following problems.
%                 1. Split data file may be corrupt if norm(s)<3
%                 2. Infinite loop in raw search algorithm for speech
% boundaries if there is no signal detected that has 20 frames or longer
% duration.
%                 3. Speech portion of the signal starting too early
% might indicate a possible noise sequence in the beginning due to
% bad recording.
%       All above cases need further investigation after cropping to make sure
% the cropped data is not corrupt.
%  <References>
% [1] G. S. Ying, C. D. Mitchell, L. H. Jamieson, “Endpoint detection of isolated
%      utterances based on a modified Teager energy measurement,” Proceedings of 1993
%       IEEE International Conference on Acoustics, Speech, and Signal Processing
%       (ICASSP-93), Vol.2, pp.732-735, April 1993.
%-----------------------------------------------------------------------------------------------------

if nargin < 2 fs = 8000; end
if nargin < 3 win = 'R'; end    % Default Rectangular Window.
if nargin < 4 fl = fs/50; end    % Default frame length is 160 for 20 ms.
if nargin < 5 inc = fl/2; end    % Default inc.in overlapping window is 80.
if win == 'R'
   % Default Rectangular Window of length 160 for 20 ms frames.
   w = rectwin(fl);
   else if win =='N' w = hann(fl);
       else if win =='H' w = hamming(fl); end
       end
end

s = load(datadir);
sm = s-mean(s);           % Takes the mean out of data before processing.
speechflag = 0;
if norm(sm) < 1.5; datacropped = []; speechflag =1;   % Checking bad split trials if any.
```

```
else

% IIR Elliptical BPF Design
%d = fdesign.bandpass(100/4000,150/4000,2000/4000,2400/4000,60,0.5,60);
%hd = ellip(d); sf = filter(hd,sm);        % Filters the speech data.

% First-order pre-emphasis filter
%a = 1; b = [1, -15/16];
%sf = filter(b,a,sm);

% IIR Elliptical HPF filter
d = fdesign.highpass('n,fst,fp,ap', 6, 100/4000, 150/4000, 0.5);
hd = ellip(d); sf = filter(hd,sm);

% Frame-based Teager Energy Measure
% 1. Calculate the power spectrum
% 2. Weight each sample in the power spectrum with the square of the frequency
% 3. Take the square root of the sum of the weighted power spectrum.

% Zero padding signal if necessary before % 50 overlapping windowing.
m1 = (length(sf)/inc);
m1 = ceil(m1); sf(length(sf):inc*m1) = zeros;

end

% Frame-based Teager's Energy Measurement, FTE
f = fs*(0:64)/128;
Wf = f.^2;            % Weighting factor for power spectrum of each frame.
for n=1:m1-1;
   sw = sf(80*(n-1)+1:80*(n-1)+160).*w;
   swfft = fft(sw,128);
   Ps = swfft.*conj(swfft) / 128;
   WPs = transpose(Ps(1:65)).*Wf;  % Power Spect. weighted by the square of the freq.
   FTE(n) = sqrt(sum(WPs));
end
% Endpoint detection using Teagar's Frame-based Modified Energy.
maxft = mean(FTE(1:10))+std(FTE(1:10));
maxfte = min(12,maxft);
ITL_T = 6*maxfte;
ITU_T = 18*maxfte;
dur = 0; rend = 0; rstart = 0; dt = 1; loopn = 0;
while dur < 20;
   for n = dt:(m1-1)
      if FTE(n) > ITU_T;
         rindex = n;
         for l = rindex:-1.0:1
```

```
            if FTE(l) < ITL_T; rstart = l; break; end
         end
         break
      end
   end
   for k = rindex:m1-1
      if FTE(k) < ITL_T; rend = k; break; end
   end
   dt = rend+1;
   dur = rend - rstart;
   loopn = loopn +1;
   if loopn > 5; speechflag =0; speechlength =0; break; end
end
startt = 80*rstart;
endt = 80*(rend-1);
speechlength = startt-endt;
datacropped = sf(startt:endt);

% Plots of ste, zcr, FTE, original speech signal, thresholds and resulting
% endpoints on the speech signal.
figure(1); subplot(2,1,1); plot(FTE); hold on;
subplot(2,1,1), plot(1:length(FTE),ITU_T,'r'), hold on;
subplot(2,1,1), plot(1:length(FTE),ITL_T,'g');
title('Frame-based Teagers Energy'); ylabel('Magnitude'); xlabel('Frame Number');
subplot(2,1,2); plot(s); hold on;
subplot(2,1,2); plot(startt,-2:0.1:2,'g'); hold on;
subplot(2,1,2); plot(endt,-2:0.1:2,'g');
title('MOVE'); ylabel('Magnitude'); xlabel('Sample Number');

end
%-------------------------------------------------------------------------------------------------
% End of function endpoint2.m
%-------------------------------------------------------------------------------------------------
```

3. SPEECH CROPPING BASED ON ENDPOINT DETECTION

```
%  Filename             : cropspeech.m
%  Written by           : R.S. Kurcan, LTJG. TU NAVY.
%  Date last revised  : 11\15\2005.
%  Purpose               : This function crops the individual speech files
%  in the the “Split” folder and saves them in the “Cropped” folder based
%  on an endpoint detection algorithm.
%   <Description>
%
% This function crops the split speech files located in the user-specified
% 'Data' directory and saves them in the subfolder, “Cropped.”
% User specifies the directory of the folder where the speech data is
% stored. This routine first lists the personal folders, then it finds the
% split folder under each person and then lists the folders of 7 words.
% Afterward, the algorithm calls for the endpoint.m function for each
% word (txt file). Cropped data file is saved under the folder 'Cropped'
% unless it is corrupt. If it is corrupt it is saved under folder 'Corrupt'
%-----------------------------------------------------------------------------------------------

w = cd;
user_entry = input('Specify the directory of the folder for the speech data:\n','s');
ind_folder = dir(user_entry);
max_length = 3000; min_length = 2500;

for n = 3:length(ind_folder)
    parentdir = strcat(user_entry,'\',ind_folder(n).name);
    cd(parentdir);
    mkdir('Cropped'); mkdir('Corrupt');
    cd(strcat(parentdir,'\Cropped'));
    mkdir('down'); mkdir('kill'); mkdir('kill_noise'); mkdir('left');
    mkdir('move'); mkdir('pan'); mkdir('right'); mkdir('right_outside');
    mkdir('up');
    cpath = strcat(parentdir,'\Split');
    cd(cpath);
    split_folder = dir;
    flags = []; flags.stat = []; flags.name = []; g = 1;

        for m = 3:length(split_folder)
        wpath = strcat(cpath,'\',split_folder(m).name);
        cd(wpath);
        word_folder = dir;
        cd(w);
        %zlen = []; p = 1;
            for k = 3:length(word_folder)
            datapath = strcat(wpath,'\',word_folder(k).name);

            [datacropped,speechlength,speechflag] = endpoint1(datapath);  % Calls the
            % endpoint detection function.
            fprintf(1,'... Cropping file: %s and length, %6.2f\n',datapath, speechlength);

            if (speechflag == 0) % If cropped data is not a bad one.
               if speechlength > max_length;
                   max_length = speechlength;
                   max_file = word_folder(k).name;
               else if (speechlength < min_length)&&(speechlength ~= 0);
                       min_length = speechlength;
                       min_file = word_folder(k).name;
                   end
               end
               savepath =strcat(parentdir,'\Cropped\',split_folder(m).name,...
                   '\',word_folder(k).name);
               save(savepath, 'datacropped','-ASCII');
               %zlen(p) = speechlength;
               %p = p+1;
            else          % If cropped data is a bad trial.
               savepath =strcat(parentdir,'\Corrupt\',word_folder(k).name);
               save(savepath, 'datacropped','-ASCII');
               flags(g).name = word_folder(k).name;
               flags(g).stat = speechflag;
               g = g+1;
            end
            clear('datacropped','speechlength','speechflag');
            end
            %savepath =strcat(parentdir,'\Cropped\',split_folder(m).name,'\zlen.txt');
            %save(savepath, 'zlen','-ASCII'); clear('zlen');

        end
        savepath =strcat(parentdir,'\Corrupt');
        cd(savepath);
        save('flagstat.mat', 'flags');
        clear('flags');
end
max_file
max_length
min_file
min_length
%-----------------------------------------------------------------------------------------------
% End of function endpoint2.m
%-----------------------------------------------------------------------------------------------
```

APPENDIX B. SPEECH FEATURE EXTRACTION AND VECTOR QUANTIZATION MATLAB PROGRAMS

This appendix includes the MATLAB programs used for feature extraction (MFCC and LPC-CC computation), vector quantization and codebook generation. All the MATLAB codes originated from those available at [Sorensen, 2005] and [Brookes, 1997]. Either the original codes were used or they were modified and the source of code cited in the header of each program.

1. MEL-FREQUENCY CEPSTRAL COEFFICIENT (MFCC) COMPUTATION

```
%  Filename              : melcepst.m
%  Written by            : Mike Brookes (VOICEBOX).
%  Date last revised     : 02/21/2005.
%  Modified by           : R.S. Kurcan, LTJG. TU NAVY.
%  Date last modified : 10/25/2005.
%  Purpose               : This routine computes the Mel-cepstrum Cepstral
%  Coefficients (MFCC) of a speech signal on a frame-by-frame basis. This
%  routine calls the following subroutines: enframe.m, melbankm.m,
%  rfft.m and rdct.m which were also written by Mike Brookes and appended
%  to the end of the routine.
%
function c = melcepst(s,fs,w,nc,p,n,inc)
%
% Simple use: c=melcepst(s,fs)         % calculate mel cepstrum with 12 coefs,
% 256 sample frames
% c=melcepst(s,fs,'e0dD')              % include log energy, 0th cepstral coef,
% delta and delta-delta coefs
%
% <Inputs>
%    s     :speech signal
%    fs    :sample rate in Hz (default 11025)
%    nc    :number of cepstral coefficients excluding 0'th coef. (default 12)
%    n     :length of frame (default power of 2 <30 ms))
%    p     :number of filters in filterbank (default floor(3*log(fs)) )
%    inc   :frame increment (default n/2)
%
%        w   :any sensible combination of the following:
%
%                        'R'     :rectangular window in time domain
%                        'N'     :Hanning window in time domain
%                        'M'     :Hamming window in time domain (default)
%
```

```
%            't'   :triangular shaped filters in mel domain (default)
%            'n'  :hanning shaped filters in mel domain
%            'm' :hamming shaped filters in mel domain
%
%                        'p'       :filters act in the power domain
%                        'a'       :filters act in the absolute magnitude domain (default)
%
%                  '0'  :include 0'th order cepstral coefficient
%                  'e'  :include log energy
%                  'd'  :include delta coefficients (dc/dt)
%            'D'   :include delta-delta coefficients (d^2c/dt^2)
%
%            'z'   :highest and lowest filters taper down to zero (default)
%            'y'   :lowest filter remains at 1 down to 0 frequency and
%                         highest filter remains at 1 up to nyquist freqency
%
%   If 'ty' or 'ny' is specified, the total power in the fft is preserved.
%
% <Output>    c     :mel cepstrum output: one frame per row
%
%     Copyright (C) Mike Brookes 1997
%
%   VOICEBOX is a MATLAB toolbox for speech processing. Home page is at
%   http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
-----------------------------------------------------------------------------------------------------

format long;
if nargin<2 fs=8000; end
if nargin<3 w='M'; end
if nargin<4 nc=12; end
if nargin<5 p=floor(3*log(fs)); end
if nargin<6 n=pow2(floor(log2(0.03*fs))); end
if nargin<7 inc=floor(n/2); end

fh=0.5;            % fh  high end of highest filter as a fraction of fs.
fl=0;              % fl  low end of the lowest filter as a fraction of fs.
if length(w)==0; w='M'; end
if any(w=='R')
  z=enframe(s,n,inc);
elseif any (w=='N')
  z=enframe(s,hanning(n),inc);
else
  z=enframe(s,hamming(n),inc);       % Hamming window in time domain (default).
end
f=fft(z.');
```

```
[m,a,b]=melbankm(p,n,fs,fl,fh,w);
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-6;
if any(w=='p')                          % filters act in the power domain.
   y=log(max(m*pw,pth));
else                                    % filters act in the absolute magnitude domain (default).
   ath=sqrt(pth);
   y=log(max(m*abs(f(a:b,:)),ath));
end
c=dct(y).';
nf=size(c,1);
nc=nc+1;
if p>nc
   c(:,nc+1:end)=[];
elseif p<nc
   c=[c zeros(nf,nc-p)];
end
if ~any(w=='0')      % It is better not to include 0th order mel cepstrum
% coefficient due to its more pronounced high amplitude.
   c(:,1)=[];
   nc=nc-1;
end
if any(w=='e')       % include log energy.
   c=[log(sum(pw)).' c];
   nc=nc+1;
end

% calculate derivative

if any(w=='D')
  vf=(4:-1:-4)/60;
  af=(1:-1:-1)/2;
  ww=ones(5,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+10,nc);
  vx(1:8,:)=[];
  ax=reshape(filter(af,1,vx(:)),nf+2,nc);
  ax(1:2,:)=[];
  vx([1 nf+2],:)=[];
  if any(w=='d')
     c=[c vx ax];
  else
     c=[c ax];
  end
elseif any(w=='d')
  vf=(4:-1:-4)/60;
  ww=ones(4,1);
  cx=[c(ww,:); c; c(nf*ww,:)];
  vx=reshape(filter(vf,1,cx(:)),nf+8,nc);
  vx(1:8,:)=[];
  c=[c vx];
end

if nargout<1
   [nf,nc]=size(c);
   t=((0:nf-1)*inc+(n-1)/2)/fs;
   ci=(1:nc)-any(w=='0')-any(w=='e');
   imh = imagesc(t,ci,c.');
   axis('xy');
   xlabel('Time (s)');
   ylabel('Mel-cepstrum coefficient');
   map = (0:63)'/63;
   colormap(jet(64));
   % colormap([map map map]);
   colorbar;
end

%-----------------------------------------------------------------------------------------------------
% End of routine melcepst.m
%-----------------------------------------------------------------------------------------------------

%  SUBROUTINE
function f=enframe(x,win,inc)
%  ENFRAME split signal up into (overlapping) frames: one per row. F=(X,WIN,INC)
%
%   F = ENFRAME(X,LEN) splits the vector X up into
%   frames. Each frame is of length LEN and occupies
%   one row of the output matrix. The last few frames of X
%   will be ignored if its length is not divisible by LEN.
%   It is an error if X is shorter than LEN.
%
%   F = ENFRAME(X,LEN,INC) has frames beginning at increments of INC
%   The centre of frame I is X((I-1)*INC+(LEN+1)/2) for I=1,2,...
%   The number of frames is fix((length(X)-LEN+INC)/INC)
%
%   F = ENFRAME(X,WINDOW) or ENFRAME(X,WINDOW,INC) multiplies
%   each frame by WINDOW(:)

%      Copyright (C) Mike Brookes 1997
%      Version: enframe.m,v 1.3 2005/02/21 15:22:12
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
-----------------------------------------------------------------------------------------------------

nx=length(x);
nwin=length(win);
if (nwin == 1)
   len = win;
else
   len = nwin;
end
if (nargin < 3)
   inc = len;
end
nf = fix((nx-len+inc)/inc);
f=zeros(nf,len);
indf= inc*(0:(nf-1)).';
inds = (1:len);
f(:) = x(indf(:,ones(1,len))+inds(ones(nf,1),:));
if (nwin > 1)
    w = win(:)';
    f = f .* w(ones(nf,1),:);
end

%--------------------------------------------------------------------------------------------------
% End of subroutine enframe.m
%--------------------------------------------------------------------------------------------------

%  SUBROUTINE
function [x,mn,mx]=melbankm(p,n,fs,fl,fh,w)
%     MELBANKM     determine     matrix     for     a     mel-spaced     filterbank
%     [X,MN,MX]=(P,N,FS,FL,FH,W)
%
% <Inputs>
%       p      :number of filters in filterbank
%       n      :length of fft
%       fs     :sample rate in Hz
%       fl     :low end of the lowest filter as a fraction of fs (default = 0)
%       fh     : high end of highest filter as a fraction of fs (default = 0.5)
%       w      :any sensible combination of the following:
%            't'   :triangular shaped filters in mel domain (default)
%            'n'   :hanning shaped filters in mel domain
%            'm'   :hamming shaped filters in mel domain
%
%            'z'   :highest and lowest filters taper down to zero (default)
%            'y'   :lowest filter remains at 1 down to 0 frequency and
%                 highest filter remains at 1 up to nyquist freqency
%
%           If 'ty' or 'ny' is specified, the total power in the fft is preserved.
%
% <Outputs>
%       x        :a sparse matrix containing the filterbank amplitudes
%            If x is the only output argument then size(x)=[p,1+floor(n/2)]
%            otherwise size(x)=[p,mx-mn+1]
%       mn      :the lowest fft bin with a non-zero coefficient
%       mx      :the highest fft bin with a non-zero coefficient
%
%  <Usage>
%       f=fft(s);                          f=fft(s);
%       x=melbankm(p,n,fs);                [x,na,nb]=melbankm(p,n,fs);
%       n2=1+floor(n/2);                   z=log(x*(f(na:nb)).*conj(f(na:nb)));
%       z=log(x*abs(f(1:n2)).^2);
%       c=dct(z); c(1)=[];
%
%   To plot filterbanks e.g.    plot(melbankm(20,256,8000)')
%
%     Copyright (C) Mike Brookes 1997
%     Version: melbankm.m,v 1.3 2005/02/21 15:22:13
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
--------------------------------------------------------------------------------------------------------

if nargin < 6
  w='tz';
  if nargin < 5
    fh=0.5;
    if nargin < 4
      fl=0;
    end
  end
end
f0=700/fs;
fn2=floor(n/2);
lr=log((f0+fh)/(f0+fl))/(p+1);
% convert to fft bin numbers with 0 for DC term
bl=n*((f0+fl)*exp([0 1 p p+1]*lr)-f0);
b2=ceil(bl(2));
b3=floor(bl(3));
if any(w=='y')
  pf=log((f0+(b2:b3)/n)/(f0+fl))/lr;
  fp=floor(pf);
  r=[ones(1,b2) fp fp+1 p*ones(1,fn2-b3)];
  c=[1:b3+1 b2+1:fn2+1];
  v=2*[0.5 ones(1,b2-1) 1-pf+fp pf-fp ones(1,fn2-b3-1) 0.5];
  mn=1;
  mx=fn2+1;
else
  b1=floor(bl(1))+1;
  b4=min(fn2,ceil(bl(4)))-1;
  pf=log((f0+(b1:b4)/n)/(f0+fl))/lr;
  fp=floor(pf);
  pm=pf-fp;
  k2=b2-b1+1;
  k3=b3-b1+1;
  k4=b4-b1+1;
  r=[fp(k2:k4) 1+fp(1:k3)];
  c=[k2:k4 1:k3];
  v=2*[1-pm(k2:k4) pm(1:k3)];
  mn=b1+1;
  mx=b4+1;
end
if any(w=='n')
  v=1-cos(v*pi/2);
elseif any(w=='m')
  v=1-0.92/1.08*cos(v*pi/2);
end
if nargout > 1
  x=sparse(r,c,v);
else
  x=sparse(r,c+mn-1,v,p,1+fn2);
end

%----------------------------------------------------------------------------------------------------
%  End of subroutine melbankm.m
%-----------------------------------------------------------------------------------------------------

%  SUBROUTINE
function y=rfft(x,n,d)
%  RFFT     FFT of real data Y=(X,N)
%  Data is truncated/padded to length N if specified.
%   N even:     (N+2)/2 points are returned with
%                   the first and last being real
%   N odd:      (N+1)/2 points are returned with the
%                   first being real
% In all cases fix(1+N/2) points are returned
%
```

```
%      Copyright (C) Mike Brookes 1998
%      Version: rfft.m,v 1.3 2005/02/21 15:22:14
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
-----------------------------------------------------------------------------------------------------

s=size(x);
if prod(s)==1
   y=x
else
   if nargin <3
      d=find(s>1);
      d=d(1);
      if nargin<2
         n=s(d);
      end
   end
   if isempty(n)
      n=s(d);
   end
   y=fft(x,n,d);
   y=reshape(y,prod(s(1:d-1)),n,prod(s(d+1:end)));
   s(d)=1+fix(n/2);
   y(:,s(d)+1:end,:)=[];
   y=reshape(y,s);
end

%----------------------------------------------------------------------------------------------------
% End of subroutine rfft.m
%----------------------------------------------------------------------------------------------------

%  SUBROUTINE
function y=rdct(x,n,a,b)
%RDCT     Discrete cosine transform of real data Y=(X,N,A,B)
% Data is truncated/padded to length N.
%
% This routine is equivalent to multiplying by the matrix
%
%   rdct(eye(n)) = diag([sqrt(2)*B/A repmat(2/A,1,n-1)]) * cos((0:n-1)'*(0.5:n)*pi/n)
%
% Default values of the scaling factors are A=sqrt(2N) and B=1 which results in an
%  orthogonal matrix. Other common values are A=1 or N and/or B=1 or sqrt(2).
%  If b~=1 then the columns are no longer orthogonal.
%
```

```
%      Copyright (C) Mike Brookes 1998
%      Version: rdct.m,v 1.4 2005/02/21 15:22:14 dmb
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
----------------------------------------------------------------------------------------------------

fl=size(x,1)==1;
if fl x=x(:); end
[m,k]=size(x);
if nargin<2 n=m;
end
if nargin<4 b=1;
    if nargin<3 a=sqrt(2*n);
    end
    end
if n>m x=[x; zeros(n-m,k)];
elseif n<m x(n+1:m,:)=[];
end

x=[x(1:2:n,:); x(2*fix(n/2):-2:2,:)];
z=[sqrt(2) 2*exp((-0.5i*pi/n)*(1:n-1))].';
y=real(fft(x).*z(:,ones(1,k)))/a;
y(1,:)=y(1,:)*b;
if fl y=y.'; end

%----------------------------------------------------------------------------------------------------
%  End of subroutine rdct.m
%----------------------------------------------------------------------------------------------------
```

2. LPC-DERIVED CEPSTRAL COEFFICIENT (LPC-CC) COMPUTATION

```
%  Filename             : hmmfeatures.m
%  Written by           : Peter S.K. Hansen, IMM, Technical University of Denmark.
%  Date last revised    : 09/30/2000.
%  Purpose              : This routine computes the LPC-derived Cepstral Coefficients
%  (LPC-CC) for feature extraction. This routine calls the subroutine durbin.m which is
%  also written by Peter S.K. Hansen so as to implement the Levinson-Durbin recursion.
%  This subroutine is appended to the end of the routine.

function y = hmmfeatures(s,N,deltaN,M,Q)
%  hmmfeatures --> Feature extraction for HMM recognizer.
%
%  <Synopsis>
%    y = hmmfeatures(s,N,deltaN,M,Q)
%
```

```
%  <Description>
%    A frame based analysis of the speech signal, s, is performed to
%    give observation vectors (columns of y), which can be used to train
%    HMMs for speech recognition.
%
%    The speech signal is blocked into frames of N samples, and
%    consecutive frames are spaced deltaN samples apart. Each frame is
%    multiplied by an N-sample Hamming window, and Mth-order LP analysis
%    is performed. The LPC coefficients are then converted to Q cepstral
%    coefficients, which are weighted by a raised sine window. The result
%    is the first half of an observation vector, the second half is the
%    differenced cepstral coefficients used to add dynamic information.
%    Thus, the returned argument y is an 2Q-by-T matrix, where T is the
%    number of frames.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%
%------------------------------------------------------------------------------------------------

Ns = length(s);                     % Signal length.
T  = 1 + fix((Ns-N)/deltaN);           % No. of frames.

a       = zeros(Q,1);
gamma   = zeros(Q,1);
gamma_w = zeros(Q,T);

win_gamma = 1 + (Q/2)*sin(pi/Q*(1:Q)'); % Cepstral window function.

for (t = 1:T)                       % Loop frames.
  % Block into frames.
  idx = (deltaN*(t-1)+1):(deltaN*(t-1)+N);

  % Window frame.
  sw = s(idx).*hamming(N);

  % Short-term autocorrelation.
  [rs,eta] = xcorr(sw,M,'biased');

  % LP analysis based on Levinson-Durbin recursion.
  [a(1:M),xi,kappa] = durbin(rs(M+1:2*M+1),M); % LP analysis.

  % Cepstral coefficients.
  gamma(1) = a(1);
  for (i = 2:Q)
```

```
    gamma(i) = a(i) + (1:i-1)*(gamma(1:i-1).*a(i-1:-1:1))/i;
  end

  % Weighted cepstral sequence for frame t.
  gamma_w(:,t) = gamma.*win_gamma;
end

% Time differenced weighted cepstral sequence.
delta_gamma_w = gradient(gamma_w);

% Observation vectors.
y = [gamma_w; delta_gamma_w];

%-----------------------------------------------------------------------------------------------------
% End of routine hmmfeatures.m
%-----------------------------------------------------------------------------------------------------

%  SUBROUTINE
function [a,xi,kappa] = durbin(r,M)
%  durbin --> Levinson-Durbin Recursion.
%
%  <Synopsis>
%    [a,xi,kappa] = durbin(r,M)
%
%  <Description>
%    The function solves the Toeplitz system of equations
%
%       [  r(1)   r(2)  ...  r(M)  ] [  a(1)  ] = [  r(2)  ]
%       [  r(2)   r(1)  ... r(M-1) ] [  a(2)  ] = [  r(3)  ]
%       [    .      .          .   ] [    .   ] = [    .   ]
%       [ r(M-1) r(M-2) ...  r(2)  ] [ a(M-1) ] = [  r(M)  ]
%       [  r(M)  r(M-1) ...  r(1)  ] [  a(M)  ] = [ r(M+1) ]
%
%    (also known as the Yule-Walker AR equations) using the Levinson-
%    Durbin recursion. Input r is a vector of autocorrelation
%    coefficients with lag 0 as the first element. M is the order of
%    the recursion.
%
%    The output arguments are the M estimated LP parameters in the
%    column vector a, i.e., the AR coefficients are given by [1; -a].
%    The prediction error energies for the 0th-order to the Mth-order
%    solution are returned in the vector xi, and the M estimated
%    reflection coefficients in the vector kappa.
%
%    Since kappa is computed internally while computing the AR coefficients,
%    then returning kappa simultaneously is more efficient than converting
%    vector a to kappa afterwards.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, p. 300, (2000).
%
%  <Revision>
%    Peter S.K. Hansen, IMM, Technical University of Denmark
%
%    Last revised: September 30, 2000
%----------------------------------------------------------------------------------------------------

% Initialization.
kappa = zeros(M,1);
a     = zeros(M,1);
xi    = [r(1); zeros(M,1)];

% Recursion.
for (j=1:M)
  kappa(j) = (r(j+1) - a(1:j-1)'*r(j:-1:2))/xi(j);
  a(j)     = kappa(j);
  a(1:j-1) = a(1:j-1) - kappa(j)*a(j-1:-1:1);
  xi(j+1)  = xi(j)*(1 - kappa(j)^2);
end

%----------------------------------------------------------------------------------------------------
% End of subroutine durbin.m
%----------------------------------------------------------------------------------------------------
```

3. K-MEANS CLUSTERING ALGORITHM

```
%   Filename              : k_means.m
%   Written by            : Peter S.K. Hansen, IMM, Technical University of Denmark.
%   Date last revised     : 09/30/2000.
%   Purpose               : This routine implements the K-means clustering algorithm for
%   the speech feature vectors.
function [Yc,c,errlog] = k_means(Y,K,maxiter)
%  kmeans --> Trains a k-means cluster model.
%
%  <Synopsis>
%    [Yc,c,errlog] = k_means(Y,K,maxiter)
%
%  <Description>
%    The function uses the k-means algorithm to set the centroids of
%    a cluster model. The matrix Y represents the data which is being
%    clustered, with each row corresponding to a vector, and K is the
%    desired number of clusters. The cluster centroids are returned in
%   the matrix Yc (rows), and the cluster number for each data vector
%   are returned in the vector c. The sum of squares error function is
%   used in the algorithm, and a log of the error values after each
%   iteration is returned in errlog. The maximum number of iterations
%   is specified by maxiter.
%
% <References>
% [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%     Processing of Speech Signals”, IEEE Press, p. 71, (2000).
%----------------------------------------------------------------------------------------------

[M,N] = size(Y);            % Number of vectors and vector dim.
if (K > M)
  error('More centroids than data vectors.')
end

errlog = zeros(maxiter,1);   % Log of error value after each iteration.

% Initialize centroids Yc by random selection of K vectors in Y.
perm = randperm(M);
Yc   = Y(perm(1:K),:);

% Constant term in squared Euclidean distance between rows in Y and Yc.
d2y = (ones(K,1)*sum((Y.^2)'))';

for (i = 1:maxiter)
  % Save old centroids to check for termination.
  Yc_old = Yc;

  % Squared Euclidean distance (M-by-K matrix) between rows in Y and Yc.
  d2 = d2y + ones(M,1)*sum((Yc.^2)') - 2*Y*Yc';

  % Assign each vector in Y to nearest centroid.
  [errvals,c] = min(d2');

  % Adjust the centroids based on the new assignments.
  for (k = 1:K)
    if (sum(c==k)>0)
      Yc(k,:) = sum(Y(c==k,:))/sum(c==k);
    end
  end

  % Error value is the total squared distance from cluster centroids.
  errlog(i) = sum(errvals);
  fprintf(1,'... Iteration %4d --- Error %11.6f\n',i,errlog(i));
  % Test for termination.
  if (max(max(abs(Yc - Yc_old))) < 10*eps)
    errlog = errlog(1:i);
    return
  end
end

%-----------------------------------------------------------------------------------------------------
% End of routine k_means
%-----------------------------------------------------------------------------------------------------
```

4. CODEBOOK GENERATION FOR VECTOR QUANTIZATION

```
%   Filename             : hmmcodebook.m
%   Written by           : Peter S.K. Hansen, IMM, Technical University of Denmark.
%   Date last revised    : 09/30/2000.
%   Modified by          : R.S. Kurcan, LTJG. TU NAVY.
%   Date last modified : 12/18/2005.
%   Purpose              :This routine generates a codebook using the subroutine
%    k_means.m. This codebook is used for vector quantization of the continuous-valued
%    speech feature vectors.
function [cb,K,T,dist,index] = hmmcodebook(datadir,ntrial,Kmax,maxiter);
%  hmmcodebook.m --> Codebook generation for HMM recognizer.
%
%  <Synopsis>
%    [cb,K,T,dist] = hmmcodebook(datadir,ntrial,Kmax,maxiter)
%
%  <Inputs>
%  Kmax         : Maximum cluster number for k_means.m subfunction
%  maxiter      : Maximum iteration number in k-means clustering algorithm
%  datadir      : Directory of the speech data
%  ntrial        : Number of occurrences of each word to be used for
%           generating the codebook
%
%  <Outputs>
%  cb   : Codebook vectors (Locations of cluster centroids)
%  K    : Number of vector-quantized clusters
%  T    : Number of observation vectors (rows) used to generate codebook.
%  dist : VQ distortion based on Euclidean distance
%
%  <Description>
%  After user specifies the initial directory of the data folder,
%  this routine reads the data in from the cropped folder. Next it computes
%  the 12 MFCC parameters and 12 delta-MFCC parameters. Zero-order MFCC
%  coefficient is excluded from the features. Cepstral Mean Substraction is
%  applied to the static MFCC parameters. Dynamic MFCC parameters are
%  weighted by six to fit them into the range of static parameters.
%  A feature matrix of 24 by T is computed where T shows the frame length.
%  The feature vectors for all sequences are concatenated and then vector
%  quantized (clustered) into K < Kmax feature vector (centroids).
%  The number of observation vectors used to generate the codebook is
%  returned in T, and the VQ average distortion is returned in dist.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
% ---------------------------------------------------------------------------------------------------
format long;
if nargin<4 maxiter=1000; end
if nargin<3 Kmax=128; end
if nargin<2 ntrial=15; end
if nargin<1 datadir='D:\MATLAB7\work\Data'; end

cw = cd;
ind_folder = dir(datadir);

% The structure array words.name excludes the words : 'right_outside' and
% 'kill_noise'
words(1).name='down'; words(2).name='kill'; words(3).name='left';
words(4).name='move'; words(5).name='pan'; words(6).name='right';
words(7).name='up';

% Before feature extraction MFCC parameters are set below.
fs = 8000;    % Sampling freq.
w ='Mtd';     % Hamming window in time domain, triangular shaped filters
% in mel domain. All filters act in the absolute magnitude domain. Delta
% parameters are computed as well.
nc = 12;           % 12 MFCC plus 12 delta-MFCC computed excluding 0'th coef.
p = 24;            % number of filters in critical-band filterbank (default 26).
n = 256;           % length of frame.
inc = 156;         % increments in the speech frames.
coef = [];         % feature matrix
index = randperm(40);          % Trial index splits the data into training set and test set.
k = index(1:ntrial);           % Training index
cntr = 1;

for a = 3:length(ind_folder)
   parentdir = strcat(datadir,'\',ind_folder(a).name,'\Cropped\');
   for g = 1:7
      wpath = strcat(parentdir,words(g).name);
      cd(wpath);
      wfolder = dir;
           for m = 1:ntrial
```

```
         dpath = strcat(wpath,'\',wfolder(k(m)+2).name);
         s = load(dpath,'-ascii');

         cd(cw);
         f = melcepst(s,fs,w,nc,p,n,inc); % returns each row as a frame
         % Cepstral mean substraction applied to the 12 MFCC parameters.
         f1 = f(:,1:12); f1 = f1 - mean(f1(:));
         f = [f1 6*f(:,13:24)];
         f = f';                          % Each column is set as a frame.

         fprintf(1,'... Reading file: %4d, %s, T = %4d\n',cntr,…
                           wfolder(k(m)+2).name,length(f));

         cntr = cntr+1;
         coef = [coef f];                 % Feature matrix containing all
         % training occurences of words.
         clear('f','f1','s');
      end
   end
end

spath = 'D:\MATLAB7\work\Thesis\';
cd(spath);
mkdir('Training');

% K-means search reads in rows of input as MFCC features and
% columns of the data correspond to speech frames.

[Yc,c,errlog] = k_means(coef',Kmax,maxiter); % Vector quantization.

cb= Yc(unique(c),:)';             % Only use clusters with assignments.

[N,K] = size(cb);                 % Codebook symbol size K.
[N,T] = size(coef);               % Vector dim and number of vectors.
dist  = sqrt(errlog(end))/T;      % VQ average distortion measure.
savepath = strcat(spath,'Training\cb.txt');
save(savepath, 'cb','-ascii');
savepath = strcat(spath,'Training\trialindex.txt');
save(savepath, 'index','-ascii');
cd(cw);

%---------------------------------------------------------------------------------------------------
% End of routine hmmcodebook.m
%---------------------------------------------------------------------------------------------------
```

APPENDIX C. HIDDEN MARKOV MODEL MATLAB PROGRAMS

This appendix includes various MATLAB programs used for HMM-based recognition of the isolated words. Most of these MATLAB codes originated from those available at [Sorensen, 2005]. Either the original codes were used or they were modified and the source of code cited in the header of each program.

1. HMM BAUM-WELCH RE-ESTIMATION

```
%  Filename              : hmmfb.m
%  Written by            : Peter S.K. Hansen, IMM, Technical University of Denmark.
%  Date last revised     : 09/30/2000.
%  Purpose               : This subroutine implements Baum-Welch re-estimation algorithm
%  (or Forward-Backward algorithm) and is called in the routine “hmmtrain.m” for the
%  training of HMMs.
%
function [A,B,pi1,loglike] = hmmfb(ytrain,S,K,maxiter,tol)
%
%  <Synopsis>
%    [A,B,pi1,loglike] = hmmfb(ytrain,S,K,maxiter,tol)
%
%  <Description>
%    The function implements the F-B reestimation algorithm used
%    to train a HMM based word classifier. The input arguments are
%    the L training sequences (same word), i.e., vectors in the
%    cell-array ytrain, the HMM parameters, S and K, which are the
%    number of states and the number of symbols (codebook vectors),
%    and maxiter and tol are stopping criteria in the F-B algorithm,
%    e.g., 5000 and 1e-3.
%    The output arguments are the S-by-S state transition matrix A,
%    the K-by-S observation probability matrix B, and the initial state
%    probability vector pi1. Finally, the output vector loglike is
%    the log-likelihood as function of the iteration number in the
%    F-B reestimation algorithm.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%----------------------------------------------------------------------------------------------------
 L = length(ytrain);                          % Number of training sequences.

A = rand(S,S); A = A./repmat(sum(A),S,1);   % Initial random model
B = rand(K,S); B = B./repmat(sum(B),K,1);   % with probability sum
pi1 = rand(S,1); pi1 = pi1/sum(pi1);        % normalized to one columnwise.
```

```
loglike = zeros(maxiter,1);                     % Log-likelihood measures.
loglike_old = -inf;                             % From previous iteration.
loglike_dif = inf;                              % Difference.

k = 0;
while (k < maxiter) & (loglike_dif >= tol)
 k = k + 1;                                     % Iteration counter.
 pi_c = 0; A_c = 0; B_c = 0;                    % Initialize sum over
 loglike_c = zeros(L,1);                        % sequences.

 for (l = 1:L)
  y = ytrain{l};                                % l'th sequence.
  T = length(y);                                % Length of l'th seq.
  alpha = zeros(S,T);                           % Forward recursion.
  beta  = zeros(S,T);                           % Backward recursion.
  scale = zeros(T,1);                           % Scale factors.
  alpha(:,1) = pi1.*B(y(1),:)';                 % Alpha for t=1.
  scale(1)   = sum(alpha(:,1));                 % Scale factor for t=1.
  alpha(:,1) = alpha(:,1)/scale(1);             % Scaled alpha for t=1.
  for (t = 2:T)
   alpha(:,t) = A*alpha(:,t-1).*B(y(t),:)';
   scale(t)   = sum(alpha(:,t));                % Scale factor for t.
   alpha(:,t) = alpha(:,t)/scale(t);            % Scaled alpha for t.
  end

  beta(:,T) = 1/scale(T);                       % Beta for t=T.
  for (t = (T-1):-1:1)                          % Scaled beta for t.
   beta(:,t) = A'*(beta(:,t+1).*B(y(t+1),:)')/scale(t);
  end

  loglike_c(l) = sum(log10(scale));             % Log-likelihood.
  gamma_c = (alpha.*beta).*repmat(scale',S,1);
  A_c  = A_c  + (alpha(:,1:T-1)*(B(y(2:T),:)'.*beta(:,2:T))').*A';
  B_c  = B_c  + (gamma_c*(repmat([1:K],T,1)==repmat(y(:),1,K)))';
  pi_c = pi_c + gamma_c(:,1);
 end

 loglike(k)  = sum(loglike_c);
 loglike_dif = loglike(k) - loglike_old;
 loglike_old = loglike(k);

 pi1 = pi_c/sum(pi_c);
 A   = A_c./repmat(sum(A_c),S,1);
 B   = B_c./repmat(sum(B_c),K,1);
 fprintf(1,'... Log likelihood: %6.3f\n',loglike(k));
end
```

```
loglike = loglike(2:k);

%-----------------------------------------------------------------------------------------------------
% End of function hmmfb.m
%-----------------------------------------------------------------------------------------------------
```

2. HMM VITERBI TRAINING ALGORITHM

```
%   Filename             : hmmviterbi.m
%   Written by           : R.S. Kurcan, LTJG. TU NAVY.
%   Date last revised    : 11/28/2005.
%   Purpose              : This subroutine implements the Viterbi Training Algorithm.
%
function [A,B, pi1,loglike] = hmmviterbi(ytrain,S,K,maxiter,tol)
%
%  <Synopsis>
%    [A,B,loglike] = hmmviterbi(ytrain,S,K,maxiter,tol)
%
%  <Description>
%    This function is loosely based on the file “hmmfb.m” written by Peter S.K. Hansen,
%    in an effort to follow similar notations. The function implements the Viterbi          %
algorithm used to train a HMM-based word classifier as an alternative to F-B          %
algorithm. The input arguments are the L training sequences of the same word, i.e., %
vectors in the cell-array ytrain, the HMM parameters, S and K, which are the number %
of states and the number of symbols (codebook vectors). Maxiter and tol are          %
stopping criteria in the Viterbi algorithm, e.g., 1000 and 1e-5.
%    The output arguments are the S-by-S state transition matrix A, the K-by-S
%    observation probability matrix B. The initial state probability vector pi1 is not
%    estimated assuming the initial state is state one in a left-to-right HMM. Finally, the
%    output vector loglike is the log-likelihood as function of the iteration number in the
%    Viterbi  algorithm.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%-----------------------------------------------------------------------------------------------------
L = length(ytrain);                % Number of training sequences.

A = rand(S,S);
Au = tril(A);                      % Lower triangular part of A.
Al = triu(A,1);                    % Upper triangular part of A.
A = Au + eps*Al;                   % Upper tri. part set to 'eps' for a left-to-right HMM.
A = A./repmat(sum(A),S,1);   % Columns of A and B must sum
B = rand(K,S); B = B./repmat(sum(B),K,1); % to unity.
pi1 = [1; zeros(S-1,1)];           % a particular state designated as initial state.

loglike = zeros(maxiter,1);     % Log-likelihood measures.
```

```
loglike_old = inf;              % From previous iteration.
loglike_dif = inf;              % Difference.

k = 0;
while (k < maxiter) & (loglike_dif >= tol)
 k = k + 1;                     % Iteration counter.
 loglike = zeros(L,1);

  for (l = 1:L)
  y = ytrain{l};                % l'th sequence.
  T = length(y);                % Length of l'th seq.

  % Initialization
  Dmin = zeros(T,S);            % Dmin(t,it) is distance from (0,0) to (t,it)
                                % over the best path.
  Dmin(1,:) = pi'.*B(y(1),:);   % Dmin(1,i) at time t=1.
  gamma = zeros(T,S);           % The best final state on the optimal partial
                                % path ending at (t,it).
  % Recursion
   for t = 2:T
    for i = 1:S
     D = Dmin(t-1,:)+(-log10(A(i,:)))+(-log10(B(y(t),:)));
     [Dmin(t,i),gamma(t,i)] = min(D);
    end
   end

   % Termination
   bestseq = zeros(1,T);
   [mindist,argmin] = min(Dmin(T,:));
   bestseq(T) = argmin;

   % Backtracking
   for t = (T-1):-1:1
     bestseq(t) = gamma(t+1, bestseq(t+1));   % Best state sequence.
   end
   loglike(l)  = mindist;
  end

  loglike(k)  = sum(loglike);
  loglike_dif = loglike_old - loglike(k);
  loglike_old = loglike(k);

  % Viterbi reestimation
  for i = 1:S
    X = find(bestseq == i);
    nfromi(i) = length(X);      % Number of transitions from state i.
  end
  ntoj = nfromi;                   % Number of transitions to state j.
  nfromi(bestseq(end)) = nfromi(bestseq(end))-1;     % First and last states
  ntoj(bestseq(1)) = ntoj(bestseq(1))-1                % don't count transitions.

  nu_ji = zeros(S,S);              % Number of transitions from state i to state j.
  for m =1:S-1
    i = bestseq(m);
    j = bestseq(m+1);
    nu_ji(j,i) = nu_ji(j,i)+1;
  end

  nu_kj = zeros(K,S);   % Number of times observation k and state j occur jointly.
  for p = 1:S
    for t = 1:T
    nu_kj(y(t),bestseq(p)) = nu_kj(y(t),bestseq(p))+1;
    end
  end

  % Model parameters updated
  A_m = nu_ji./repmat(nfromi,S,1);
  B_m = nu_kj./repmat(ntoj,K,1);

  fprintf(1,'... Log likelihood: %6.3f\n',loglike(k));

end
  loglike = loglike(2:k);

%---------------------------------------------------------------------------------------------------
% End of function hmmviterbi.m
%---------------------------------------------------------------------------------------------------
```

3. MULTIPLE-OBSERVATION HMM TRAINING ALGORITHM

```
%   Filename              : hmmtrain.m
%   Written by            : Peter S.K. Hansen, IMM, Technical University of Denmark.
%   Date last revised     : 09/30/2000.
%   Modified by           : R.S. Kurcan, LTJG. TU NAVY.
%   Date last modified : 01/05/2006.
%   Purpose               : This routine implements the multiple-observation  training of
%   HMMs using the Baum-Welch re-estimation algorithm. (see also hmmfb.m)
%
function [A_m,B_m,pi_m,loglike_m,testidx] = hmmtrain(S,O,R,maxiter,tol,datadir);
%
%  <Inputs>
%   S             : Number of hidden states in HMMs
%   O             : Number of observation sequences taken from each speaker for training of
%                   the same word.
%   R             : Number of words to be recognized
%   maxiter     : Max. iteration number to terminate training
%   tol             : Tolerance to terminate training
%   datadir      : Directory of the cropped speech data
%
%  <Outputs>
%   A_m         : Cell array for the state transition matrices of HMMs
%   B_m         : Cell array for the observation probability matrices of HMMs
%   pi_m        : Cell array for the state probability vectors of HMMs
%   loglike_m : Cell array for the log likelihood of the observation
%   sequences given the models.
%
%  <Description>
%   The function implements training of multiple HMMs. The training
%   sequences are defined by the cell-array data. The length of the
%   cell-array corresponds to the number of words, and each cell is a
%   new cell array with filenames for occurrences of this spoken word.
%   For each sequence, a frame based analysis is performed using the
%   function y = melcepst(s,fs,w,nc,p,n,inc)to give observation
%   vectors, which are then vector quantized into the possible
%   codebook vectors given by cb. For each word, the (quantized)
%   observation sequences are then used to train a HMM for this word,
%   using the F-B reestimation algorithm. Here, S is the number of
%   states in the HMM, and maxiter and tol are stopping criteria in
%   the F-B algorithm, e.g., 1000 and 1e-5. The output probability
%   measures for the HMMs are returned in the cell-arrays A_m, B_m,
%   pi_m, and loglike_m.
%
%  <References>
%   [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%       Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%------------------------------------------------------------------------------------------------------

format long;

% Default input parameterts
if nargin < 6 datadir='C:\Program Files\MATLAB71\work\Data'; end
if nargin < 5 tol = 1e-5; end
if nargin < 4 maxiter = 1000; end
if nargin < 3 R = 7; end
if nargin < 2 O = 25; end
if nargin < 1 S = 8; end

cb = load('C:\Program Files\MATLAB71\work\Training\cb.txt','-ascii');
index = load('C:\Program Files\MATLAB71\work\Training\trialindex.txt','-ascii');
```

```
% The structure array words.name includes the seven words to be recognized.
words(1).name='up'; words(2).name='down'; words(3).name='left';
words(4).name='right'; words(5).name='kill'; words(6).name='pan';
words(7).name='move';

% MFCC parameters are set before feature extraction.
fs = 8000;      % Sampling freq.
w ='Mtd';       % Hamming window in time domain, triangular shaped filters.
% in mel domain. All filters act in the absolute magnitude domain. Delta
% parameters are specified for computation as well.
nc = 12;        % 12 MFCC plus12 delta-MFCC computed excluding 0'th coef.
p = 24;         % number of filters in critical-band filterbank (default 26).
n = 256;        % length of frame.
inc = 156;      % increments in the speech frames.
coef = [];      % feature matrix with each column representing a frame.

[dim,K] = size(cb);       % Get symbol size K from the codebook.
A_m  = cell(R,1);         % Output arguments in cell-arrays.
B_m  = cell(R,1);
pi_m = cell(R,1);
loglike_m = cell(R,1);

cw = cd;
ind_folder = dir(datadir);
L = (length(ind_folder)-2)*O;   % Total number of occurrences of i'th word.
trnidx = index(1:O);            % Random trial indeces for training.
testidx = index(O+1:end);       % Random trial indeces for testing.
cntr = 1;
for i = 1:R                     % Loop words.
   ytrain = cell(L,1);          % Generate cell-array for train seq.
   j = 1;

   for a = 3:length(ind_folder)
      parentdir = strcat(datadir,'\',ind_folder(a).name,'\Cropped\', words(i).name);
      cd(parentdir);
      wfolder = dir;

      for m = 1:O

         s = load(wfolder(trnidx(m)+2).name,'-ascii');   % Load word file
         cd(cw);
         y = melcepst(s,fs,w,nc,p,n,inc);            % Extract MFCC feature vectors.

         % Cepstral mean-substraction applied to the 12 MFCC parameters.
         y1 = y(:,1:12); y1 = y1 - mean(y1(:));
         y = [y1 6*y(:,13:24)];
         y = y';                  % Each column is set as feature vector of a frame.
         [dim,T] = size(y);       % Number of vectors T.
         yk = zeros(T,1);

         fprintf(1,'... Reading file: %4d, %s, T = 4d\n',cntr,wfolder(trnidx(m)+2).name,T);
         cntr = cntr + 1;
         for (t = 1:T)            % Vector quantization.
            [dist,k] = min(sum((cb-repmat(y(:,t),1,K)).^2));
            yk(t) = k;            % Symbol at time/frame t.
         end;

         ytrain(j) = {yk};
         j = j+1;
         clear('y','y1','s','yk','k');
         cd(parentdir);
      end
   end
   cd(cw);
   [A,B,pi1,loglike] = hmmfb(ytrain,S,K,maxiter,tol);
   A_m(i)  = {A};
   B_m(i)  = {B};
   pi_m(i) = {pi1};
   loglike_m(i) = {loglike};
end

%---------------------------------------------------------------------------------------------
% End of function hmmtrain.m
%---------------------------------------------------------------------------------------------
```

4. HMM LOGLIKELIHOOD COMPUTATION ALGORITHM

```
%   Filename            : loglike.m
%   Written by          : Peter S.K. Hansen, IMM, Technical University of Denmark.
%   Date last revised   : 09/30/2000.
%   Purpose             : This routine computes the log-likelihood for a given
%   observation sequence y consisting of vector quantized speech feature
%   vectors by using Hidden Markov Model parameters.

function loglike = hmmlogp(y,A,B,pi1)
%  hmmlogp --> Log-likelihood for given observation sequence and HMM.
%
%  <Description>
%    The function calculates the log-likelihood for a given observation sequence y,
%    and Hidden Markov Model A, B, and  pi1. Here, A is the S-by-S state transition
%    matrix, B is the K-by-S observation probability matrix, and pi1 is the initial state
%    probability vector.
%
```

```
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%-----------------------------------------------------------------------------------------------

format long;
T = length(y);                    % Length of observation sequence.
S = length(A);                    % Number of hidden states.

alpha = zeros(S,T);               % Forward recursion.
beta  = zeros(S,T);               % Backward recursion.
scale = zeros(T,1);               % Scale factors.

alpha(:,1) = pi1.*B(y(1),:)' + eps;      % Alpha for t=1.
scale(1)   = sum(alpha(:,1));            % Scale factor for t=1.
alpha(:,1) = alpha(:,1)/scale(1);        % Scaled alpha for t=1.
for (t = 2:T)
  alpha(:,t) = A*alpha(:,t-1).*B(y(t),:)';
  scale(t)   = sum(alpha(:,t));          % Scale factor for t.
  alpha(:,t) = alpha(:,t)/(scale(t) + eps);          % Scaled alpha for t.
end

loglike = sum(log10(scale + eps));
 %----------------------------------------------------------------------------------------------
% End of function hmmlogp
%----------------------------------------------------------------------------------------------
```

5. HMM RECOGNITION ALGORITHM

```
%   Filename             : hmmrecog.m
%   Written by           : Peter S.K. Hansen, IMM, Technical University of Denmark.
%   Date last revised  : 09/30/2000.
%   Modified by          : R.S. Kurcan, LTJG. TU NAVY.
%   Date last modified  : 02/10/2006.
%   Purpose               : This routine implements HMM recognition for a given
%   observation sequence based on the trained input model parameters.
%
function [logp,guess] = hmmrecog(A_m,B_m,pi_m,testidx);
%  hmmrecog --> HMM based word classifier.
%
%  <Description>
%    The function implements HMM based recognition, where the cell-array
%    data contains filenames for the spoken words to be recognized.
%    For each word, a frame based analysis is performed using the
%    function y = melcepst(s,fs,w,nc,p,n,inc) to give observation
%    vectors, which are then vector quantized into the possible
%    codebook vectors stored in the file cb.txt. The resulting observation
%   sequence and the probability measures for all the HMMs, given by the
%   cell-arrays A_m, B_m, and pi_m, are then used to calculate the
%   log-likelihoods for the HMMs (column of logp). The input “testidx”
%   contains the randomly chosen trial index for the testing set. The word
%   associated with the HMM of highest log-likelihood is declared
%   to be the recognized word, and the index is returned in guess.
%   This procedure is repeated for all the words in data.
%
%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, “Discrete-Time
%      Processing of Speech Signals”, IEEE Press, chapter 12, (2000).
%-----------------------------------------------------------------------------------------------

format long;

datadir='C:\Program Files\MATLAB71\work\Data';
cb = load('C:\Program Files\MATLAB71\work\Training\cb.txt','-ascii');
%index = load('C:\Program Files\MATLAB71\work\Training\trialindex.txt','-ascii');

% The structure array words.name includes the seven words to be recognized.
words(1).name='up'; words(2).name='down'; words(3).name='left';
words(4).name='right'; words(5).name='kill'; words(6).name='pan';
words(7).name='move';

% Before feature extraction MFCC parameters are set below.
fs = 8000;      % Sampling freq.
w ='Mtd';       % Hamming window in time domain, triangular shaped filters.
% in mel domain. All filters act in the absolute magnitude domain. Delta
% parameters are specified for computation as well.
nc = 12;        % 12 MFCC plus 12 delta-MFCC computed excluding 0'th coef.
p = 24;         % number of filters in critical-band filterbank (default 26).
n = 256;        % length of frame.
inc = 156;      % increments in the speech frames.
coef = [];      % feature matrix with each column representing a frame.

[dim,K] = size(cb);             % # of observation symbols.
R = length(A_m);                % # of trained words (7 words).

O = length(testidx);            % # of test words from each speaker per word.

cw = cd;
ind_folder = dir(datadir);
L = (length(ind_folder)-2)*O;  % Total # of ith words to be recognized/tested.
logp = zeros(R,L);
v = 1; cntr = 1;
```

```
for i = 1:R                         % Loop words.

   for a = 3:length(ind_folder)
      parentdir = strcat(datadir,'\',ind_folder(a).name,'\Cropped\', words(i).name);
      cd(parentdir);
      wfolder = dir;

      for m = 1:O
         % Load test word signal from file.
         s = load(wfolder(testidx(m)+2).name,'-ascii');

         cd(cw);
         y = melcepst(s,fs,w,nc,p,n,inc); % Extract 12 feature vectors.

         % Cepstral mean-substraction applied to the 6 MFCC parameters.
         y1 = y(:,1:12); y1 = y1 - mean(y1(:));
         y = [y1 6*y(:,13:24)];
         y = y';

         [dim,T] = size(y);            % Number of vectors T.

         fprintf(1,'... Reading file:%4d, %s, T %4d\n',cntr,wfolder(testidx(m)+2).name,T);
         cntr = cntr + 1;
         yk = zeros(T,1);

         % Vector quantization.
         for (t = 1:T)
           [dist,k] = min(sum((cb-repmat(y(:,t),1,K)).^2));
           yk(t) = k;
         end

         % Scoring
         for (j = 1:R)       % Estimate log-likelihood for each model.
         logp(j,v) = hmmlogp(yk,A_m{j},B_m{j},pi_m{j});
         end
         v = v+1;
         clear('y','y1','s','yk','k');
         cd(parentdir);
      end
   end
end
cd(cw);
[loglike,guess] = max(logp);
%------------------------------------------------------------------------------------------------
% End of function hmmrecog.m
%------------------------------------------------------------------------------------------------
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. HMM CLASSIFICATION RESULTS FOR MULTIPLE EXPERIMENTS

```
%   Filename              : hmmresult.m
%   Written by            : R.S. Kurcan, LTJG. TU NAVY.
%   Date last revised  : 02/15/2005.
%   Purpose               : This subroutine estimates confusion matrices and
%   computes 95 % confidence intervals for N experiments
%
%   hmmresult --> HMM based IWR classification results and confusion matrices
%
%  <Description>
%    The function implements estimates confusion matrices and computes
%    95 % confidence intervals for N experiments where N is the number
%    of multiple experiments. N is chosen such that (N*0.05/2) should be
%    integer for 95 % confidence interval computation.
%-------------------------------------------------------------------------------------------------
clear all;
N = 80;                        % # of experiments
conf = cell(N,1);
rate = cell(N,1);
R = 7;

for i=1:N
   [cb,K,T,dist,index] = hmmcodebook;
   [A_m,B_m,pi_m,loglike_m,testidx] = hmmtrain;
   [logp,guess] = hmmrecog(A_m,B_m,pi_m,testidx);
   O =length(testidx);
   classrate = zeros(R,R);
      for h = 1:R;
         for k = 1:R;
            classnum = 0;
            n = (O*(k-1)*20)+1;
            m = O*k*20;
               for l = n:m
                  if guess(l) == h; classnum = classnum + 1; end
               end
            classrate(k,h) = classnum/(O*20);
         end
      end
    classrate = classrate*100;
    overallrate = sum(diag(classrate))/R;
    format short;
    conf(i) = {classrate};
    rate(i) = {overallrate};
    clear('cb','K','T','dist','index','A_m','B_m','pi_m','loglike_m',...
        'testidx','logp','guess','guess','classnum','classrate','overallrate');
    rmdir('C:\Program Files\MATLAB71\work\Training','s');
end

mkdir('H:\Docs\Thesis_Work\Confusion');
savepath1 = 'H:\Docs\Thesis_Work\Confusion\conf.mat';
save(savepath1,'conf','-mat');  % Confusion matrices for N exp. are stored in cell array.
savepath2 = 'H:\Docs\Thesis_Work\Confusion\rate.mat';
save(savepath2,'rate','-mat');   % Class. rates for each of N exp. are stored in cell array.

%   95% Confidence intervals computation and average of N exps. classification results.

confint = zeros(8,2);
b = []; sum_conf = zeros(7,7); d = zeros(80,1);
for i = 1:N
   a = diag(conf{i})';              % Diag. elements of confusion matrices set as rows.
   b = [b ; a];
   sum_conf = sum_conf + conf{i};
   d(i) = rate{i};
end

c = sort(b);                        % Columns of the matrix b is sorted in ascending order.
e = sort(d);
confint(1:7,1) = c(1+(N*0.05/2),:);    % min in 95 % confidence interval.
confint(1:7,2) = c(N-(N*0.05/2),:);    % max in 95 % confidence interval.
confint(8,1) = e(1+(N*0.05/2),1);      % min in 95 % conf.  interval for avg classification.
confint(8,2) = e(N-(N*0.05/2),1);      % max in 95 % conf. interval for avg classification.

classification = mean(c);              % Mean of N exp. classification result.
mean_conf = sum_conf / N;              % Mean of N exp. confusion matrix.

savepath3 = 'H:\Docs\Thesis_Work\Confusion\confint.mat';
save(savepath3,'confint','-mat');      % 95% confidence intervals are stored in array.

savepath4 = 'H:\Docs\Thesis_Work\Confusion\classification.mat';
save(savepath4,'classification','-mat');         % Average classification rates for all words
% out of  N exp. are stored in array.

savepath5 = 'H:\Docs\Thesis_Work\Confusion\mean_conf.mat';
save(savepath5,'mean_conf','-mat');              % Average confusion matrix for N exp. is
% stored in array.
%
%-----------------------------------------------------------------------------------------------
% End of function hmmresult.m
%-----------------------------------------------------------------------------------------------
```

LIST OF REFERENCES

[Atal, 1971] B. S. Atal and L. S. Hanauer, “Speech analysis and synthesis by linear prediction of the speech wave,” *Journal of the Acoustic Society of America*, Vol. 50, pp. 637-655, 1971.

[Becchetti, 1999] C. Becchetti and L. P. Ricotti, *Speech Recognition Theory and C++ Implementation,* John Wiley & Sons, West Sussex, England, 1999.

[Baum, 1966] L. E. Baum and T. Petrie, “Statistical inference for probabilistic functions of finite state Markov chains,” *Annals of Mathematical Statistics,* Vol. 37, pp. 1554-1563, 1966.

[Baum, 1970] L. E. Baum, T. Petrie and G. Soules, “A maximization technique in the statistical analysis of probabilistic functions of Markov chains,” *Annals of Mathematical Statistics,* Vol. 41, pp. 164-171, 1970.

[Brookes, 1997] Mike Brookes, VOICEBOX: Speech Processing Toolbox for MATLAB (software), Avail. www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html, 1997, last accessed 02.15.2006.

[Cooley, 1965] J. W. Cooley and J. W. Tukey, “An algorithm for the machine computation of complex Fourier series,” *Mathematical Computations,* Vol. 19, pp. 297-301, 1965.

[Davis, 1980] S. Davis and P. Mermelstein, “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 28, No. 4, pp. 357-366, 1980.

[Deller, 2000] J. R. Deller, J. H. L. Hansen and J. G. Proakis, *Discrete-Time Processing of Speech Signals*, IEEE Press, New York, 2000.

[Deng, 2003] L. Deng and D. O’Shaughnessy, *Speech Processing A Dynamic and Optimization-Oriented Approach*, Marcel Dekker, New York, 2003.

[Dugad, 1996] R. Dugad and U.B. Desai, “A tutorial on hidden Markov models,” Technical Report No. SPANN-96.1, Indian Institute of Technology, Bombay, India, May 1996.

[Fargues, 2005] M. P. Fargues, Class Notes for EC4440 (Statistical Digital Signal Processing), Naval Postgraduate School, Monterey, California, 2005 (unpublished).

[Furui, 1986] S. Furui, “Speaker-independent isolated word recognition using dynamic features of speech spectrum,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 34, No. 1, pp. 52-59, 1986.

[Furui, 1992] S. Furui, M. M. Sondhi, *Advances in Speech Signal Processing,* Marcel Dekker, New York, 1992.

[Gold, 2000] B. Gold and N. Morgan, *Speech and Audio Signal Processing,* John Wiley & Sons, 2000.

[Graciarena, 2003] M. Graciarena, H. Franco, K. Sonmez and H. Bratt, “Combining standard and throat microphones for robust speech recognition,” *IEEE Signal Processing Letters,* Vol. 10, No. 3, pp. 72-74, March 2003.

[Jayant, 1984] N. S. Jayant and P. Noll, *Digital coding of waveforms: principles and applications to speech and video,* Prentice-Hall, 1984.

[Jelinek, 1997] F. Jelinek, *Statistical Methods for Speech Recognition,* The MIT Press, Massachusetts, 1997.

[Karnjanadecha, 2001] M. Karnjanadecha and S. A. Zahorian, “Signal modeling for high-performance robust isolated word recognition,” *IEEE Transactions on Speech and Audio Processing,* Vol. 9, No. 6, pp. 647-654, September 2001.

[Kinnunen, 2000] T. Kinnunen, T. Kilpeläinen, P. Fränti “Comparison of Clustering Algorithms in Speaker Identification,” *Proceedings of International Conference on Signal Processing and Communications (SPC 2000),* pp. 222-227, Marbella, Spain, September 2000.

[Kinnunen, 2001] T. Kinnunen, T. Kilpeläinen, P. Fränti “Is Speech Data Clustered? – Statistical Analysis of Cepstral Features,” *Proceedings of 7th European Conference on Speech Communication and Technology (Eurospeech 2001),* vol. 4, pp. 2627-2630, Aalborg, Denmark, September 2001.

[Lippmann, 1997] R. P. Lippmann, Lincoln Laboratory, “Speech recognition by machines and humans,” *Speech Communication*, Vol.22, pp.1-15, 1997.

[Linde, 1980] Y. Linde, A. Buzo and R. M. Gray, “An algorithm for vector quantizer design,” *IEEE Transactions on Communications,* Vol. 28, pp. 84-95, 1980.

[Makhoul, 1985] J. Makhoul, S. Roucos and H. Gish, “Vector quantization in speech coding,” *Proceedings of the IEEE*, Vol. 73, No. 11, pp. 1551-1588, November, 1985.

[Marsh, 1999] A. Marsh, “Human ear and hearing,” The University of Western Australia, Online Information and Course Notes, 1999, Avail. [http://www.kemt.fei.tuke.sk/Predmety/KEMT320_EA/_web/Online_Course_on_Ac oustics/hearing.html], last accessed 02.15.2006. [Moon, 1997] S. Moon and J. N. Hwang, “Robust speech recognition based on joint model and feature space optimization of hidden Markov models,” *IEEE Transactions on Neural Networks*, Vol .8, No .2, pp. 194-204, March 1997.

[Newton, 2006] M. Newton, Master’s thesis, Naval Postgraduate School, Monterey, California, 2006 (To be completed.)

[Oppenheim, 1968] A. V. Oppenheim, R. W. Schafer and T. G. Jr. Stockham, “Nonlinear filtering of multiplied and convolved signals,” *Proceedings of the IEEE*, Vol. 56, No. 8, pp. 1264-1291, 1968.

[Owens, 1993] F. J. Owens, *Signal Processing of Speech*, McGraw-Hill, 1993.

[O’Shaughnessy, 1987] D. O’Shaughnessy, *Speech Communications,* Addison Wesley, 1990.

[Picone, 1993] J. W. Picone, “Signal modeling techniques in speech recognition,” *Proceedings of the IEEE,* Vol. 81, No. 9, pp. 1215-1247, September 1993.

[Qiang, 1998] H Qiang and Z. Youwei, “On prefiltering and endpoint detection of speech signal,” *Proceedings of 1998 Fourth International Conference on Signal Processing, (ICSP '98),* Vol. 1, pp. 749-752, October 1998.

[Rabiner, 1975] L. R. Rabiner and M. R. Sambur, “An algorithm for determining the endpoints of isolated utterances,” *The Bell System Technical Journal,* February 1975.

[Rabiner, 1978] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals,* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[Rabiner, 1989] L. R. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” *Proceedings of the IEEE,* Vol.77, No.2, pp.257-286, February 1989.[Rabiner, 1993] L. R. Rabiner and B-H. Juang, *Fundamentals of Speech Recognition,* Prentice Hall, 1993.

[Sayood, 2000] K. Sayood, *Introduction to Data Compression,* Second Edition, Morgan Kaufmann Publishers, San Francisco, California, 2000.

[Sibbald, 2001] A. Sibbald, “An introduction to sound and hearing,” White paper, Sensaura, 2001, Avail. [http://www.sensaura.co.uk/whitepapers/pdfs/dev005.pdf], last accessed February 15, 2006.

[Shozakai, 1998] M. Shozakai, S. Nakamura and K. Shikano, “Robust speech recognition in car environments,” *Proceedings of the 1998 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP `98), Vol. 1, pp. 269-272, May 1998.

[Sorensen, 2005] A. Sorensen and M. Swanholm, Speech coding and recognition course notes, Avail. [http://www.itu.dk./courses/TKG/E2002], last accessed February 15, 2006.

[Srydal, 1995] A. Srydal, R. Bennett and S. Greenspan, *Applied Speech Technology,* CRC Press, Florida, 1995.

[Theodoridis, 2003] S. Theodoridis and K. Koutroumbas, *Pattern Recognition,* Second Edition, Academic Press, San Diego, California, 2003.

[Vaidyanathan, 2004a] R. Vaidyanathan, K. Hyunseok, L. Gupta and J. West, "Parametric and non-parametric signal analysis for mapping air flow in the ear-canal to tongue movements: a new strategy for hands-free human-machine interfaces," *Proceedings of the 2004 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP `04),* Vol. 5, pp. 613-16, May 2004.

[Vaidyanathan, 2004b] R. Vaidyanathan, L. Gupta, B. Chung, T. J .Allen, R. D. Quinn, M. Tabib-Azar, J. Zarycki, J. Levin, "Human-machine interface for tele-robotic operation: mapping of tongue movements based on aural flow monitoring," *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004),* Vol. 1, pp. 859-865, September-October 2004.

[Vergin, 1999] R. Vergin, "An algorithm for robust signal modeling in speech recognition," *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98),* Vol. 2, pp. 969-972, May, 1998.

[Waibel, 1990] A. Waibel and K. F. Lee, *Readings in Speech Recognition,* Morgan-Kauffmann, Palo Alto, California, 1990.

[Wang, 2005] H. Wang, Class Notes on Hidden Markov Models, Avail. [http://tigpbp.iis.sinica.edu.tw/05_Spring/access%20to%20all/20050408part1.ppt#282,1,Hidden Markov Models], last accessed February 17, 2006.

[Westerlund, 2003] N. Westerlund, "Applied Speech Enhancement for Personal Communication," Licentiate thesis, Blekinge Institute of Technology, Ronneby, Sweden, May 2003.

[Witten, 1982] I. H. Witten, *Principles of Computer Speech,* Academic Press, 1982.

[Ying, 1993] G. S. Ying, C. D. Mitchell, L. H. Jamieson, "Endpoint detection of isolated utterances based on a modified Teager energy measurement," *Proceedings of*

1993 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-93), Vol. 2, pp. 732-735, April 1993.

[Zhang, 2004] Z. Zhang, L. Zicheng, M. Sinclair, A. Acero, L. Deng, J. Droppo, X. Huang and Y. Zheng, “Multi-sensory microphones for robust speech detection, enhancement and recognition,” *Proceedings of the 2004 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP `04),* Vol. 3, pp. iii-781-4, May 2004.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Jeffrey B. Knorr
   Chairman, Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

4. Dan Boger
   Chairman, Information Sciences Department
   Naval Postgraduate School
   Monterey, California

5. Monique P. Fargues
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

6. David Jenn
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California

7. Ravi Vaidyanathan
   Systems Engineering Department
   Naval Postgraduate School
   Monterey, California

8. Deniz Kuvvetleri Komutanligi (Turkish Navy Headquarters)
   Ankara, Turkey

9. Deniz Harp Okulu Komutanligi (Turkish Naval Academy)
   Istanbul, Turkey

10. Dz.K.K. Arastirma Merkezi Komutanligi
    Elektronik Grup Baskanligi
    (Turkish Navy Research Development Center)
    Istanbul, Turkey

11. Dz.K.K. Deniz Bilimleri Enstitusu
(Turkish Naval Science Institution and Engineering School)
Istanbul, Turkey