IMPLEMENTATION OF A CONNECTED DIGIT RECOGNIZER USING CONTINUOUS HIDDEN MARKOV MODELING

by

Panaithep Albert Srichai

Thesis submitted to the Faculty of the
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Electrical Engineering

APPROVED

_____
Dr. A. A. (Louis) Beex, Chairman

_____ _____
Dr. John S. Bay Dr. Ioannis M. Besieris

September 8, 1998
Blacksburg, Virginia

Keywords: Speech Recognition, Connected-Digit Recognition, Hidden Markov Models, HMM

IMPLEMENTATION OF A CONNECTED DIGIT RECOGNIZER USING CONTINUOUS HIDDEN MARKOV MODELING

by

Panaithep Albert Srichai

Dr. A. A. (Louis) Beex, Chairman

Bradley Department of Electrical Engineering

(ABSTRACT)

This thesis describes the implementation of a speaker dependent connected-digit recognizer using continuous Hidden Markov Modeling (HMM). The speech recognition system was implemented using MATLAB and on the ADSP-2181, a digital signal processor manufactured by Analog Devices.

Linear predictive coding (LPC) analysis was first performed on a speech signal to model the characteristics of the vocal tract filter. A 7 state continuous HMM with 4 mixture density components was used to model each digit. The Viterbi reestimation method was primarily used in the training phase to obtain the parameters of the HMM. Viterbi decoding was used for the recognition phase. The system was first implemented as an isolated word recognizer. Recognition rates exceeding 99% were obtained on both the MATLAB and the ADSP-2181 implementations. For continuous word recognition, several algorithms were implemented and compared. Using MATLAB, recognition rates exceeding 90% were obtained. In addition, the algorithms were implemented on the ADSP-2181 yielding recognition rates comparable to the MATLAB implementation.

Acknowledgements

I would like to thank my parents, Dr. and Mrs. Srichai, for showing the importance of an education and providing the opportunity to pursue my Masters in Electrical Engineering. Without their continuous support, I would not be where I am today.

Another person who has aided in my research is my fiancé, Holly Jennings, for providing the extra motivation to complete this thesis. She has stood by my side during my entire graduate work. In addition, she has continuously reminded me of the importance of balancing my school work and my personal time. With her continuous encouragement, I have been able to complete this thesis in a timely manner.

I would like to thank Dr. A. A. (Louis) Beex for his enormous guidance throughout my research. He has shown me what areas to focus my research on to make it as good as possible. In addition, he has allowed time in his busy schedule to check and proofread my thesis for soundness and completeness and providing me with valuable feedback.

Furthermore I would like to thank my committee members, Dr. John Bay and Dr. Ioannis Besieris, for taking their time and effort in reviewing this work and providing me with more feedback.

Of course, I would also like to thank those who work in the Digital Signal Processing Research Laboratory, especially Sundar Sankaran and John Tilki. I feel that Sundar has given me much assistance in my research work, especially by helping me setting up and programming the ADSP-2181. Without his assistance, the amount of work

spent on this research could have been twice as long. I would also especially like to thank John for helping me to utilize many of the features found in Microsoft Word to write this thesis. Without this, I might still be editing all my cross-references. In addition, my colleagues' company through long nights in the lab was informative as well as enjoyable.

Table of Contents

List of Figures

List of Tables

1. Introduction

Communication is vital in our lives. The primary method of communication among humans is by speech. In our daily lives, we use speech to engage in one-on-one conversations and group discussions, to communicate to others via the telephone, to broadcast information via radio or television, and more. A speech signal is basically a sequence of sounds that is strung together to represent a thought that a speaker wishes to convey to a listener. While other forms of communication exist in our lives, such as writing or typing, these methods of communication are usually slower than communication by speech. In addition, these methods require a certain amount of literacy to comprehend the message.

1.1 Background of Speech Recognition

Since speech is an important part of our lives, we strive to have machines to recognize and understand our speech and act upon it. There have been machines that were successfully implemented since the early 1970's [1]. These early systems recognized discrete utterances in relatively noise-free environments. These systems were mainly speaker dependent, in which the person using the system trained the system with his or her utterances. Furthermore, these systems were limited to a small vocabulary. As advances in the field of speech recognition progressed, speech recognition systems became more complex and sophisticated. Systems have recently been developed which are intended to recognize multiple users with an expanded vocabulary.

With any speech recognition system, there are several defining characteristics. First, the system is either a speaker dependent or a speaker independent system. Next is the size of the vocabulary that the system will recognize. Finally, a distinction is made on whether the system is used to recognize speech utterances spoken in isolation or as continuous speech. The type of application intended for the system will usually determine these characteristics. These three characteristics are discussed below.

In a speaker dependent system, usually only one person uses the system. The system uses the speech utterances of the user to obtain the parameters or models to characterize the words in the vocabulary. A disadvantage of this type of system is the necessity to retrain the system for different users. A speaker independent system is designed to recognize multiple users. Usually the people using the system do not originally train the system. A speaker independent system is generally trained with speech utterances from various people. Typically more models or templates are required for the same word in a speaker independent system. This is to fully take into consideration the various ways people speak the same words. Since the users do not usually train the system, these systems typically do not perform as well as a speaker dependent system.

Vocabulary sizes range from small, via medium, to large. A small vocabulary size system can typically recognize approximately 1 to 99 words. While this may seem limited, there are many applications that use a small vocabulary size. Such a system includes a simple phone system that can recognize digits. A medium vocabulary size system can typically recognize approximately 100 to 999 words, and a large vocabulary size system can recognize 1000 or more words. With the increased vocabulary size, more and more

applications are possible. Small vocabulary systems can usually represent entire words with models or templates. However, with larger vocabulary sizes, these models or templates typically represent subunits of words, which are strung together creating the word. In addition, as the vocabulary size increases, the memory requirements and processing time also increase making the system more complex.

In an isolated-word recognition system, the words are spoken in isolation with a distinct pause between words. This is the simplest form of recognition strategy since an endpoint detector can be easily used to recognize the boundaries of the words. The pauses between the words typically are on the order of 200 ms so that they are not confused with weak fricatives and gaps within words. Furthermore the user must be cooperative to make the isolated word recognition system successful. In a connected-speech recognition system, a connected string of words is recognized by matching individual words within the string and concatenating all the words. The difficulty in a connected-speech recognition system is determining the boundaries of the individual words making up the string. In addition, coarticulation effects can degrade the performance.

The earliest experiments with speech recognition attempted to recognize the speech by detecting the information in the acoustic signal. These attempts used filter banks to effectively divide the speech signal into frequency bands [2]. The energy concentration in each band was commonly computed along with information about the number of times the zero axis was crossed. Graphs of the parameters against time were

then compared with similar profiles that had been previously stored, and the recognized word would be the one that had the closest template.

Research soon suggested that a simple matching of the acoustic patterns was not sufficient to achieve high recognition rates. There is a lot of variability in the way the same word is spoken. The speech recognition problem was soon considered a pattern-recognition problem. In the late 1970s, further research had suggested that separating the pitch in the speech signal and modeling the resonances of vocal chamber, instead of relying just on acoustical contents, improve the recognition rate for words. This was achieved using linear predictive coding (LPC). A cepstral representation of the LPC coefficients soon proved to perform better than the LPC coefficients by itself.

With the advent of faster computers, dynamic programming techniques have been developed to solve the pattern-recognition problem. Dynamic Time Warping (DTW) was first studied. This is basically a template-matching scheme in which feature vectors, which are features extracted from the speech signal, are matched to templates representing the words in the vocabulary. Since the duration of the test utterance may differ from the duration of the template, the time axis of the test utterance needs to be "warped" (stretched or compressed in time) to match the time axis of the template. The template that matches the test utterance best is deemed to be the recognized word. More recently, stochastic modeling schemes have been incorporated to solve the speech recognition problem, such as the Hidden Markov Modeling (HMM) approach. Since the feature vectors are stochastic in nature, using a stochastic model for processing is appropriate and justifiable. Systems based on DTW and HMM have given encouraging results on isolated

and connected word recognition. More recently, Artificial Neural Network (ANN) techniques have been studied, in which the parallel computing found in biological neural systems is mimicked. This field of research is gaining in popularity as it is being applied to a large vocabulary.

Much of the recent researches on speech recognition involve recognizing continuous speech from a large vocabulary using HMMs, ANNs, or a hybrid form of the two [3]. Additional algorithms have been developed to tackle the problems of continuous speech [4][5][6]. All of these systems use some type of language constraints to aid in the recognition. These constraints include lexical constraints as well as syntactic constraints.

This thesis focuses on the implementation of a connected-digit recognizer using HMMs. Such an application would be useful in a hands-free telephone. The implementation of the connected-digit recognizer is performed on MATLAB and on a digital signal processor. Traditional algorithms used for connected word recognition using a small vocabulary size are examined. In addition, a family of related algorithms has been developed and successfully implemented to improve recognition rates from the traditional algorithms. These algorithms relied on a search pattern not seen in existing algorithms to find the boundaries between consecutive digits in the spoken string. In addition, the word duration statistics for a word is incorporated which greatly improved the performance of these algorithms.

1.2 Speech Production and Modeling

As a starting point, it is wise to first look at how speech is modeled. The techniques used to model speech are used to develop a speech recognition system. In speech processing terms, basic human speech can be broken down into two parts: *excitation* and *filtering* [7].

1.2.1 Excitation

Excitation is the process in which the sounds of speech are produced. For this reason, it is also commonly called the sound production. There are two elemental types of excitation: voiced sounds and unvoiced sounds.

Voiced sounds (as its name implies) are the result of air being forced through the glottis (an opening between the vocal folds) causing the vocal cords to vibrate. The glottis and the vocal cords are housed together within the larynx. The acoustical sound produced by the larynx is called *voice*. The tension of the vocal cords is controlled and adjusted by the speaker such that the pitch of the voice can be changed. In simplistic terms, voiced sounds can be modeled as an impulse train at a fixed pitch period. Examples of voiced sounds are the sounds of all the vowels in the English language.

Unvoiced sounds are the result of a constriction at a point along the vocal tract caused by the tongue, lips, teeth, or mouth. Forcing air through this constriction causes air turbulence leading to the unvoiced sounds. Since unvoiced sounds are the result of an air turbulence, it can be modeled by random noise. An example of an unvoiced sound is the sound of the letter 's'.

However, human speech sounds are not restricted to voiced or unvoiced sounds alone. In fact, speech is composed of a combination of voiced and unvoiced sounds blended together to form a vocabulary. A *mixed* sound is a combination of voiced and unvoiced sounds. One example of this is the sound of the letter 'z'. In addition, there are *plosive* sounds which are formed by a region of silence followed by a voiced sound, unvoiced sound, or both. An example of a plosive sound is the sound of the letter 'b'.

1.2.2 Filtering

Filtering involves the combination of the glottis and the vocal tract along with the placement of the tongue, lips, teeth, and the nasal passages. These elements can be thought of as "shaping" the sound leading to different sounds for the same excitation. For this reason, the filtering is sometimes called sound shaping. The filters involved are the glottal shaping filter G(z), the vocal tract filter H(z), and the filter radiating from the lips, R(z). An extensive explanation of these filter models is beyond the scope of this thesis [1]. Figure 1.1 shows a discrete-time model for speech production. The entire sound shaping filter is then

$$S(z) = E_1(z)G(z)H(z)R(z); \text{ for voiced speech}$$
$$S(z) = E_2(z)H(z)R(z) \quad ; \text{for unvoiced speech}$$

where $E_1(z)$ represents an impulse train and $E_2(z)$ represents noise.

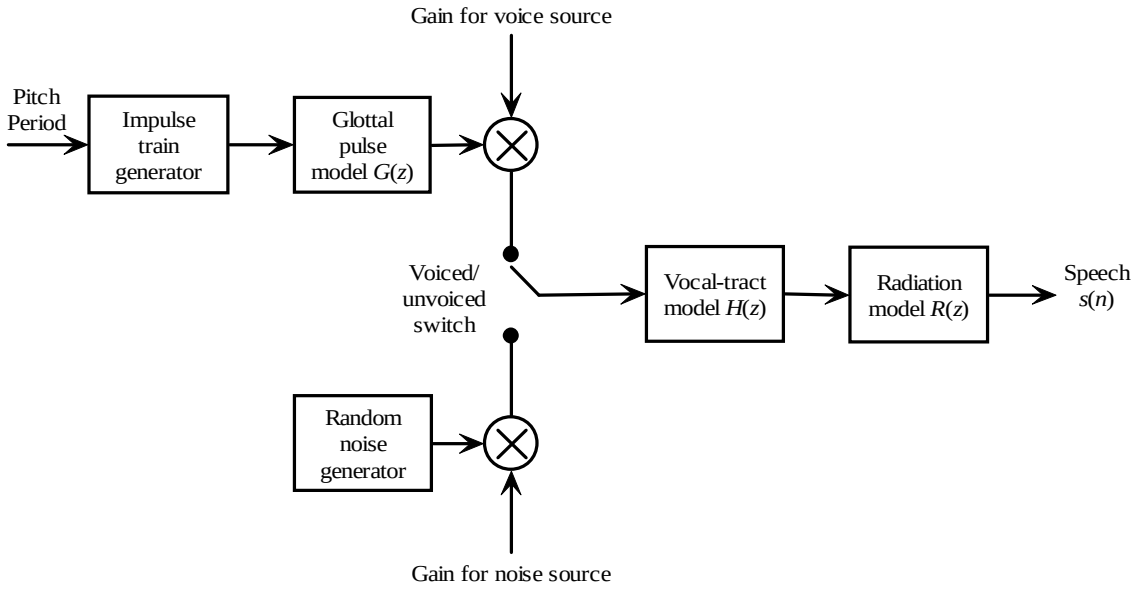


Figure 1.1 Discrete-time model of speech production

The sound shaping filter can be effectively modeled using an all-pole autoregressive (AR) model [8]. By using an all-pole model simple computational techniques, such as linear predictive coding (LPC), can be used to obtain the coefficients of the model. LPC estimates the spectral envelope of the sound-shaping filter.

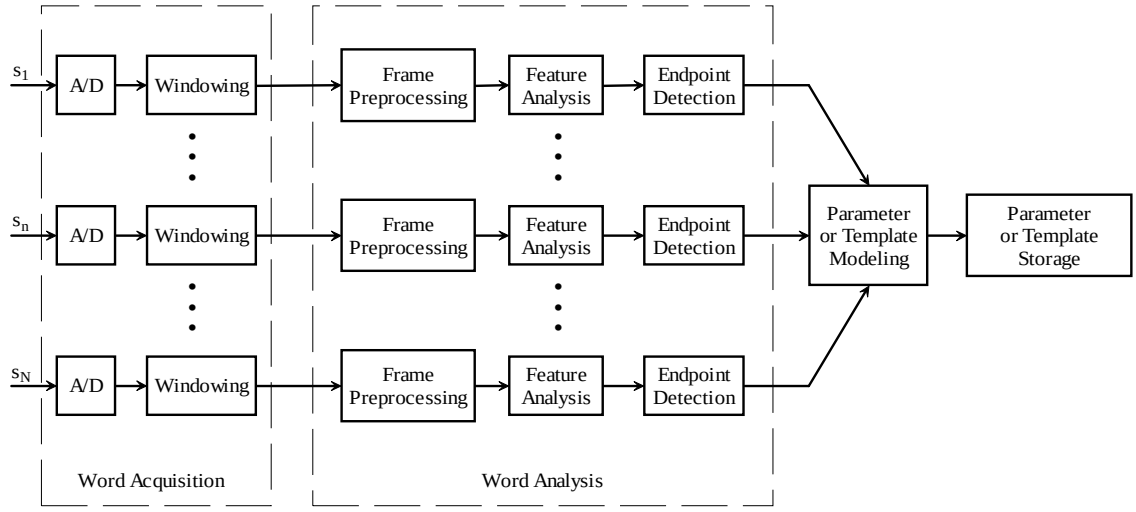
Chapters 2 and 3 describe the speech recognition system in detail and some of the strategies employed. Chapter 4 presents the results of the speech recognition system on isolated words. Chapters 5 and 6 deal with connected word recognition.

2. The Speech Recognition System

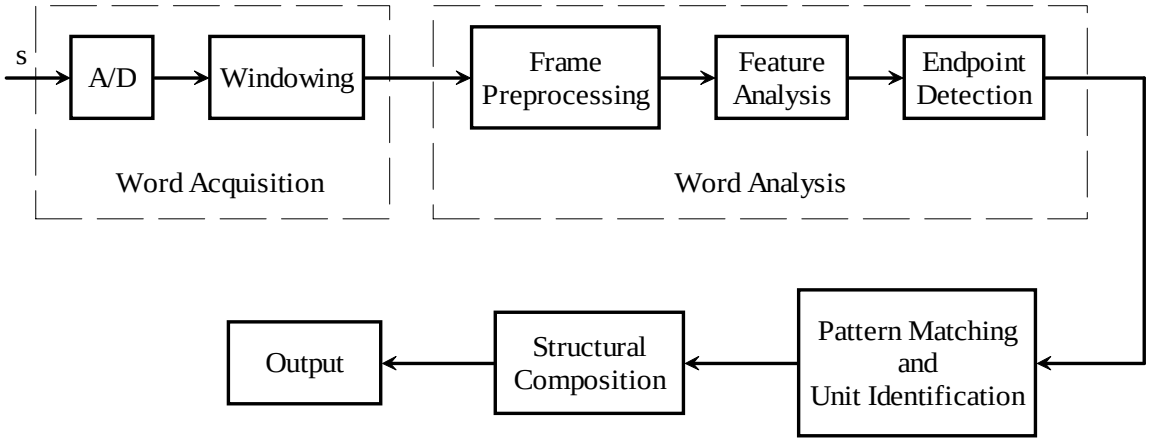
Chapter 1 explains speech production and the techniques used to model speech. This chapter describes the speech recognition system and some of the elements within this system. While the excitation will vary greatly from person to person and even between different physical and emotional states of the same person, the filtering is less sensitive to these factors. For this reason, the sound shaping part will provide all the necessary information for an effective speech recognition system.

2.1 Elements of a Speech Recognition System

In any speech recognition system, there are two distinct phases: *the training phase* and *the recognition phase*. For both phases, information about the sound-shaping filter (vocal tract) must be extracted first from the speech signal. In the training phase, information from multiple utterances of the same linguistic unit is used to develop a set of models or templates about the sound-shaping filter. In the recognition phase, the information about the sound-shaping filter is used along with the different models or templates available from the training phase to make a decision to recognize the speech. The basic block diagram for the training and recognition phases of a speech recognition system is shown in Figure 2.1.



(a)



(b)

Figure 2.1 Block diagram for a speech recognition system
(a) Training phase
(b) Recognition phase

2.1.1 Analog-to-Digital Conversion

Before we can do any analysis on the speech, we must be able to acquire the speech signal into the computer. This is performed by sampling the analog speech signal

via an analog-to-digital converter. Since speech is typically bandlimited to about 4,000 Hz, the speech signal needs to be sampled fast enough such that aliasing is avoided.

2.1.2 Windowing

Once the sampled speech record has been acquired, it needs to be segmented and sliced into overlapping time frames. This is the process of windowing the speech signal. Each frame is then analyzed separately and independently from the other frames. In choosing the window frame, there are two considerations: the type of window and the length of the window. In choosing the type of window, we want to minimize the distortion of the waveform while we want to smooth the abrupt discontinuities at the boundaries of the window. In choosing the length of the window, there are two competing factors. A larger window length will improve the spectral resolution yielding more information in a time frame. However a smaller window length will yield better time resolution [1].

2.1.3 Frame Preprocessing

The next block in the speech recognition system is the frame preprocessing. As this block implies, it performs any type of processing on a frame of speech before the actual analysis. The frame preprocessor might extract some information about the frame or perform some filtering before the actual analysis. In simplistic terminology, the preprocessing block makes the speech frame suitable for the next processing step.

2.1.4 Feature Analysis

The next block is the feature analysis block. The feature analysis block analyzes the speech frame to extract necessary information to aid in recognition. Some of the common parameters that are typically analyzed for speech recognition purposes include energies in different frequency bands, linear predictive coefficients, and cepstral coefficients. The output of the feature analysis is usually a vector called the *feature vector*. The sequence of feature vectors, from one time frame to the next, is used for the training or recognition phase in the speech recognition system.

2.1.5 Endpoint Detection

The endpoint detection block crops out any "nonspeech" regions before and after the speech signal. This effectively isolates the speech signal from the silent regions. Endpoint detection is very important in the successful implementation of a speech recognition system. Since all the information is in the speech waveform, it is necessary that only the speech signal is analyzed and not the silent regions before the beginning, or after the ending, of the speech utterance.

Most endpoint detectors rely on certain parameters in the speech signal. Usually, the parameters are described by the energy in the signal and the zero-crossing rate. However, the issue of endpoint detection becomes increasingly problematic using these parameters. To begin with, the beginning or the end of a speech signal could possibly be cropped out if it starts or ends in low-energy phonemes, such as weak fricatives or nasals. In addition, some speakers allow their speech to trail off in energy, while others tend to

produce bursts of air at the beginning and/or end of the speech. The problems of endpoint detection are not solely caused by speakers using the system. Background noise may vary also, creating interference with the speech utterance. The endpoint detector should adapt to these situations such that the system gets the best representation of the speech signal.

2.1.6 Parameter Modeling and Storage

The parameter modeling step is only for the training phase and attempts to obtain parameters from the linguistic units present in the speech. These linguistic units can be entire words or parts of words, like phonemes or syllables. Many different utterances of the same speech unit are needed for this block. Depending on the type of system being implemented this step can vary. It can be as simple as just storing the feature vectors in a library to use as a template, as is the case for a system using DTW, or it can be as complicated as modeling some type of stochastic behavior from the feature vectors and storing these parameters, as is the case with HMMs. Since this research uses HMMs, this stage consists of obtaining the parameters which make up an HMM. This is discussed in more detail in Chapter 3.

2.1.7 Pattern Matching and Unit Identification

The pattern matching and unit identification block is found in the recognition phase and attempts to match up the linguistic units extracted from the speech in the feature vector with the library of models present. This library is built up from the parameter modeling block from the training phase. The pattern matching system determines how the

library represents the features of the speech unit. For a system using DTW, the templates of the speech units are matched up with the features. For an HMM system, the likelihood of generating the features is computed. Whichever type of system is incorporated, the unit identification system chooses the best template or model for representing the features.

2.1.8 Structural Composition

The structural composition block in the recognition phase pieces the individual linguistic units into something meaningful. If the linguistic units represent entire words, then this block could attempt to piece all the words together to recognize a complete string or sentence. For linguistic units that represent phonemes or syllables, the structural composition block could attempt to recognize the word being spoken.

2.2 Word Acquisition and Analysis

The word acquisition and analysis section of the speech recognition system is now examined in more detail. The word acquisition includes the A/D and windowing blocks, while the analysis deals with the preprocessing, the feature analysis, and endpoint detection blocks. Some specifics on the implementation of these blocks are also discussed. Word acquisition and analysis are performed in real time. The processor must be able to take in the speech sample and perform all the analysis before the next frame of speech is ready. After the feature vectors have been obtained and stored, the pattern matching and unit identification block and the structural composition block are executed.

2.2.1 Sampling and Windowing

The speech signal is acquired by sampling it at a rate of 11,025 Hz. Since speech is bandlimited to about 4,000 Hz, this sampling rate is sufficient in the sense that it satisfies Nyquist criteria. A window length of 45 ms (498 samples) with a 67% overlap (30 ms or 332 samples) between consecutive frames was used. This window length was chosen primarily based on past research [9].

To implement this on the DSP processor, the speech signal is sampled using an interrupt-driven method and stored in a circular buffer of length 664 (498 samples for the window plus (498 − 332 =) 166 new samples for an overlap of (2/3 × 498 =) 332 samples). A counter is used to count the number of samples and an index pointer is used to keep track of the first sample for the next frame. When a frame is ready to be processed the corresponding consecutive 498 samples, starting at the position pointed to by the index pointer, are copied into a separate buffer. The index pointer is then incremented by 166 modulo 664. This results in 332 samples of overlap between consecutive frames. When the system is ready to sample initially, it will loop until a full frame of 498 samples is taken in. Afterwards, the system loops until the next 166 samples are taken in.

2.2.2 Frame Preprocessing

The frame preprocessing stage is next and involves computing some parameters used for the endpoint detector and performing some filtering on the frame of speech samples. A block diagram of the frame preprocessing steps is shown below in Figure 2.2.

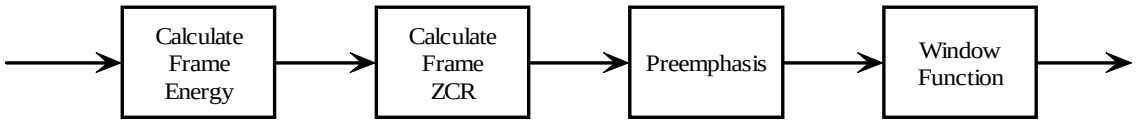


Figure 2.2 Block diagram of Frame Preprocessing

The first step is calculating the short-term energy and the zero crossing rate (ZCR). These quantities are used for endpoint detection. The short-term energy for a frame ending at time *m* is formally given by the following equation:

$$E(m) = \sum_{n=m-N+1}^{m} [s(n)]^2 \tag{2.1}$$

where *N* is the number of samples within each frame. Equation (2.1) involves summing the squares of each sample within a frame. The approximate short-term energy calculation actually used in the implementation is a variation of (2.1). In fact, it is actually a magnitude measure given by:

$$M_s(m) = \sum_{n=m-N+1}^{m} |s(n)| \tag{2.2}$$

This measurement will yield similar information as the energy measure of (2.1) since the summation of the magnitude of the signal is computed.

The zero crossing rate is then computed. A zero crossing is defined as the signal changing sign from one sample to the next. This is a relatively simple procedure in which the number of zero crossings over the frame interval is counted and recorded.

After the short-time frame energy and the zero crossing rate are computed, the speech frame is preemphasized by passing it through a high pass filter of the form:

$$P(z) = 1 - \alpha z^{-1} \tag{2.3}$$

For our implementation, $\alpha = 0.95$ is used. There are two main reasons for preemphasizing the speech frame [1]. First of all, the glottal signal can be modeled as a two-pole filter with both poles near $z = 1$. Introducing a zero near $z = 1$ will tend to cancel one of the glottal poles, while the lip radiation characteristics will cancel the other pole. In addition, preemphasis will allow the higher formants to exert influence on the outcome of the LPC analysis. Formants are the set of resonant frequencies in the speech signal which "form" the overall spectrum. The third main reason for preemphasizing the speech frame is to prevent numerical instability. Speech signals dominated by low frequencies are highly predictable. A large LP model order will result when the autocorrelation matrix is badly conditioned. Thus preemphasis will tend to whiten the spectrum, yielding better numerical stability in calculating the LP parameters.

After preemphasizing, the frame of data is multiplied by the windowing function. A Hamming window is used in the implementation of the speech recognition system [11]. The Hamming window is given as

$$w(n) = 0.54 - 0.46 \cos \frac{2\pi n}{N-1}, \qquad n = 0,\ 1,\ \ldots,\ N-1 \tag{2.4}$$

This window has the effect of having a smooth transition at the ends resulting in lower sidelobes in the spectral domain as opposed to the rectangular window.

2.2.3 Feature Analysis

The feature analysis section is the next processing block. Recall from Section 1.2.2 that the vocal tract can be modeled using an AR model by performing linear predictive analysis. The LP parameters of interest for this implementation are the weighted cepstral coefficients. In addition, a set of delta cepstral coefficients, which are the time-derivative of the cepstral coefficients, are desirable. Other parameters must first be calculated to ultimately yield the cepstral coefficients which include the reflection coefficients and the predictor coefficients of the AR model. There are several methods for finding these predictor coefficients including the Levinson-Durbin algorithm [12][13]. The strategy used in the implementation for obtaining the LP parameters is different and is shown in the functional block diagram given below in Figure 2.3.

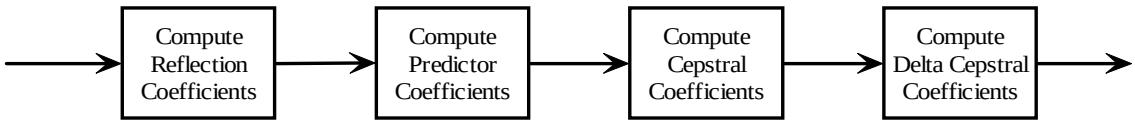


Figure 2.3 Block diagram of Feature Analysis

2.2.3.1 Reflection Coefficients

From the speech frame, the reflection coefficients of the LP lattice filter that is being modeled are calculated first, using the Schur recursion. Two arrays, the *P* array and the *Q* array, need to be initialized first. The *P* array is initialized with the first *p+1* values

from the autocorrelation sequence, and the Q array is also initialized with the first p values from the autocorrelation sequence, but in reverse order. The initialization is as follows:

$$\begin{aligned} P_0 &= Q_p = r_0 \\ P_1 &= Q_{p-1} = r_1 \\ &\vdots \\ P_{p-1} &= Q_1 = r_{p-1} \\ P_p &= r_p \end{aligned} \tag{2.5}$$

where r_k is the kth autocorrelation value associated with the time frame defined as

$$r_k = \sum_{m=0}^{N-1-k} s(m)s(m+k) \tag{2.6}$$

The Schur recursion is as follows [7]:

$$\begin{aligned} K_n &= \frac{ABS[P_1]}{P_0 \times SIGN[P_1]} \\ P_0 &= P_0 + P_1 K_n \\ \left.\begin{aligned} P_m &= P_{m+1} + K_n Q_{p-m+1} \\ Q_{p-m+1} &= Q_{p-m+1} + K_n P_{m+1} \end{aligned}\right\} & \; m = 1, 2, ..., p-n \end{aligned} \tag{2.7}$$

The above is solved recursively for n = 1, 2, . . . , p, where p is the LP filter order and K_n is the nth reflection coefficient. For our system, a tenth order system is used. The lattice structure associated with this AR model is shown in Figure 2.4.

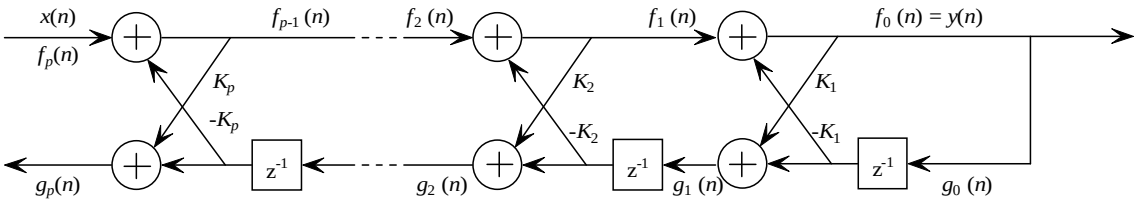


Figure 2.4 Lattice structure for the LP speech model

2.2.3.2 Predictor Coefficients

The reflection coefficients are next converted to the predictor coefficients. The predictor coefficients are the parameters for the all-pole filter that is being modeled, which has the form of

$$H(z) = \frac{1}{1 + \sum_{k=1}^{p} a_k z^{-k}} \tag{2.8}$$

where the a_k are the predictor coefficients and p is the filter order. The conversion from the reflection coefficients to the predictor coefficients is given by the following recursion [7]:

$$\begin{aligned} a_i^{(i)} &= K_i \\ a_j^{(i)} &= a_j^{(i-1)} + K_i a_{i-j}^{(i-1)} \quad 1 \le j \le i-1 \end{aligned} \tag{2.9}$$

The above is solved recursively for i = 1, 2, . . . , p. The final predictor coefficients are found from

$$a_j = a_j^{(p)} \qquad 1 \le j \le p \tag{2.10}$$

2.2.3.3 Cepstral Coefficients

While it is possible to use these predictor coefficients as the feature vectors for the Hidden Markov Model, the system in this research uses a cepstral representation. This cepstral representation includes the cepstral coefficients and the delta cepstral coefficients. Past research has shown that using cepstral features, rather than LPC coefficients,

improves speech recognition [14][15]. One of the reasons for this is that the LPC coefficients are strongly affected by the glottal dynamics. In addition, the cepstral coefficients allow the Euclidean metric distance measure as a natural distortion measure. Weighting of the cepstral coefficients, which are derived from the LPC coefficients, provides slight improvements in recognition [1].

To convert the predictor coefficients to the cepstral coefficients, the following recursion is used [7].

$$\begin{aligned} c_1 &= -a_1 \\ c_k &= -a_k - \sum_{i=1}^{k-1} a_i c_{k-i}\left(\frac{k-i}{k}\right) \qquad 1 \le k \le p \\ c_k &= -\sum_{i=1}^{p} a_i c_{k-i}\left(\frac{k-i}{k}\right) \qquad p > k \end{aligned} \tag{2.11}$$

In the above recursion, p represents the model order of the LP filter and c_k represents the kth cepstral coefficient. For the system used, twelve cepstral coefficients are calculated from the ten predictor coefficients. The cepstral coefficients are then weighted by the weighting function to improve the performance of the system by reducing some of the glottal pulse characteristics [9][10].

$$c_k^w = \left[1 + \frac{Q}{2}\sin\left(\frac{\pi k}{Q}\right)\right], \quad 1 \le k \le Q \tag{2.12}$$

where Q is the number of cepstral coefficients used in the implementation.

2.2.3.4 Delta Cepstral Coefficients

The delta cepstral coefficients are the final parameters determined in the frame analysis block. These coefficients are the time derivative of the cepstral coefficients and give information about the spectral changes from frame to frame. These coefficients are computed by first considering a window of several frames. The delta cepstral coefficients represents the difference in the cepstral coefficients from one frame to the next.

The delta cepstral coefficients at time frame *t* are computed using the following formula [9]:

$$\Delta c_k(t) = \left[\sum_{l=-L}^{L} l\, c_k(t-l) \right] G, \quad 1 \le k \le Q \tag{2.13}$$

Here, Q is the number of cepstral coefficients, $(2L + 1)$ is the length of the window (in frames), and G is a gain factor. For the system used in this research, a window length of 5 frames is used so that $L = 2$. In addition, the gain factor is chosen such that the variances of both the cepstral coefficients and the delta cepstral coefficients are approximately equal. The purpose of this is to ensure that one set of coefficients does not dominate over the other set while computing a Euclidean metric distance measure on the final feature vector.

2.2.4 Endpoint Detection

Recall that in the frame preprocessing, the short-time frame energy and the zero crossing rate were calculated, to determine the endpoints of the speech signal and thus isolate the speech utterance. In the endpoint detection, these values are used to determine

the boundaries of the spoken speech signal. The endpoint detection algorithm used in this research is a modification of the one proposed by Rabiner and Sambur [16].

There are two types of thresholds that are set up, for each the frame energy and the ZCR [7]: *possible thresholds* and *word start thresholds*. The possible thresholds are set just above the background noise and may occasionally be exceeded by spurious background noise. The word start thresholds are set relatively high, such that it is exceeded only when the system is sure that speech is being spoken.

There are two other thresholds used in this research. The first is the *minimum word length threshold*. This is set to the minimum number of frames for a spoken string. This threshold allows the rejection of isolated background noise spikes. The second threshold is the *silence threshold*. This threshold is the length of silence the system will look for to detect that a spoken string has ended. Once the system detects a speech signal, it must continue to store the cepstral features even if there are some silences within the connected string. The system will stop storing cepstral features when a sufficiently long silence is encountered, taken to indicate the end of the string. This silence threshold is set for half a second of real time.

To search for the start of a word, the algorithm compares the short-time frame energy and the ZCR with their respective word start thresholds. If one of these parameters exceeds its word start threshold, then the word start flag is asserted and the cepstral features are stored in memory for use by the pattern-matching block. If neither the frame energy nor the ZCR exceed their word start thresholds, then they are compared with their possible start thresholds. If one of these exceeds its threshold, then the possible

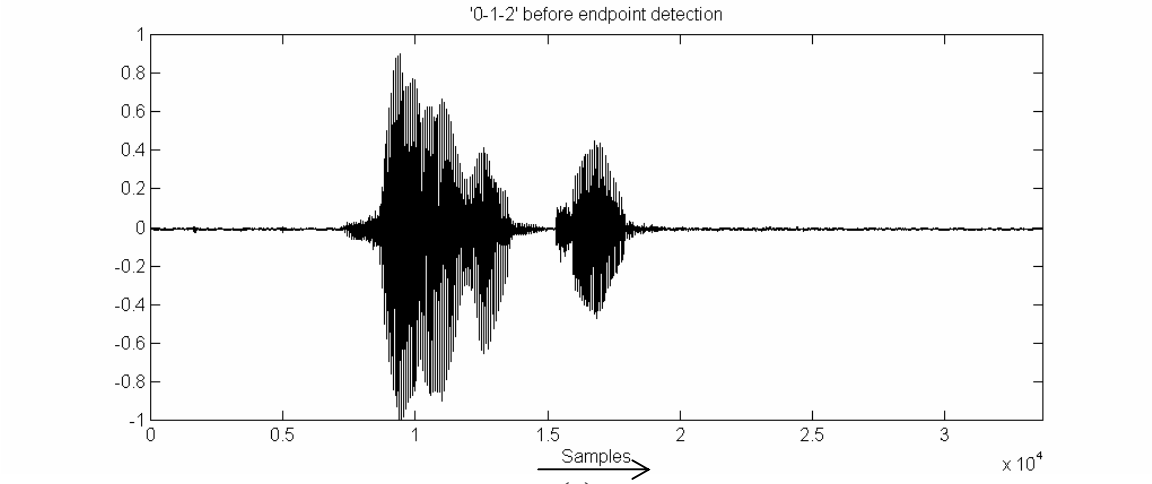
start flag is asserted and the cepstral features are stored in memory. For these features to actually become part of a word, the corresponding word start thresholds must be exceeded before both the frame energy and the ZCR fall below their possible start thresholds.

Once the word start flag is asserted, the algorithm proceeds to search for the end of the string. The end of the string is detected when the number of consecutive silent frames (frame energy and ZCR are both below their possible thresholds) exceeds the silence threshold, which is set to half a second. When this occurs, the system stops storing the cepstral features, and the last half-second of frames are discarded from pattern matching block consideration.

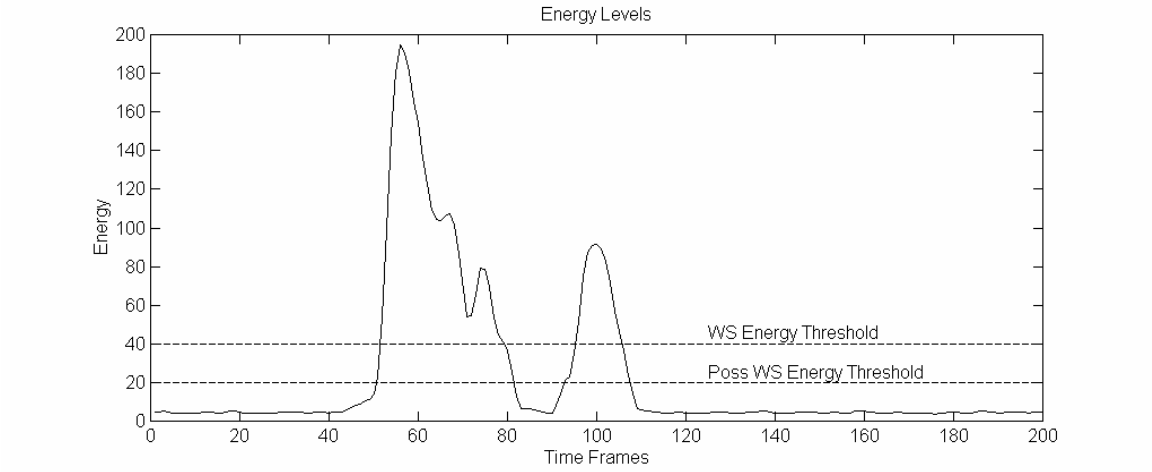
Figure 2.5 illustrates the effect of performing endpoint detection. The waveform of the connected string '0-1-2' before the endpoint detector is shown in Figure 2.5a. The speech signal was analyzed on MATLAB and normalized first before performing the detection. On the DSP implementation, this normalization does not take place. The waveform is first sliced into 45 ms time frames with 30 ms of overlap. Figure 2.5b and Figure 2.5c plot the energy levels and ZCR respectively. The dashed lines represent the word start and possible word start thresholds. All thresholds where chosen experimentally. Before the start of the speech, the energy level and the ZCR are both below their respective possible word start thresholds. Only when the start of the speech occurs do the energy levels and ZCR exceed their possible word start and word start thresholds. Finally, at the end of the speech, the energy levels and ZCR remain below their possible word start thresholds for over half a second. Figure 2.5d shows the

waveform after the endpoint detector. Notice that the silent regions before and after the speech have been cropped out, leaving just the speech waveform itself.

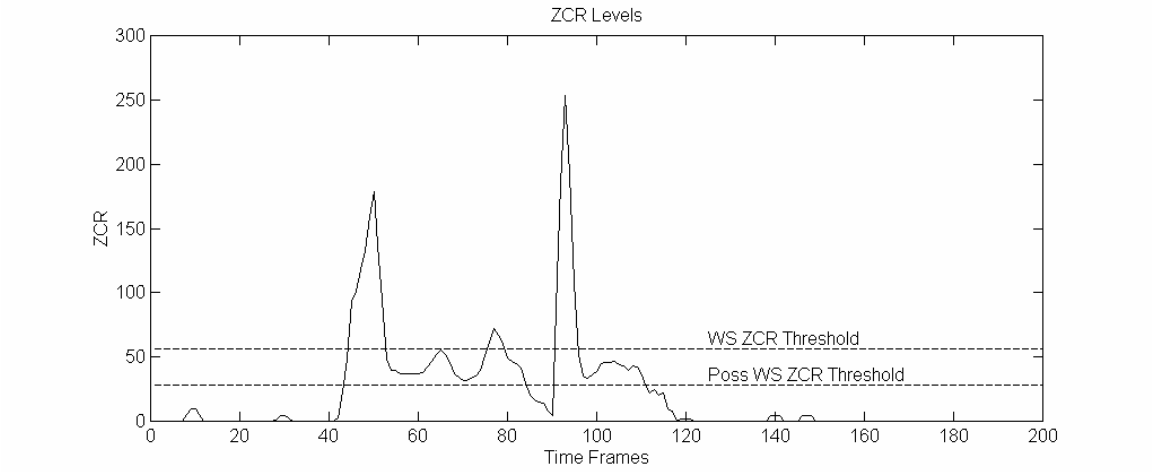
The next chapter describes the parameter modeling and pattern-matching blocks of the speech recognition system.



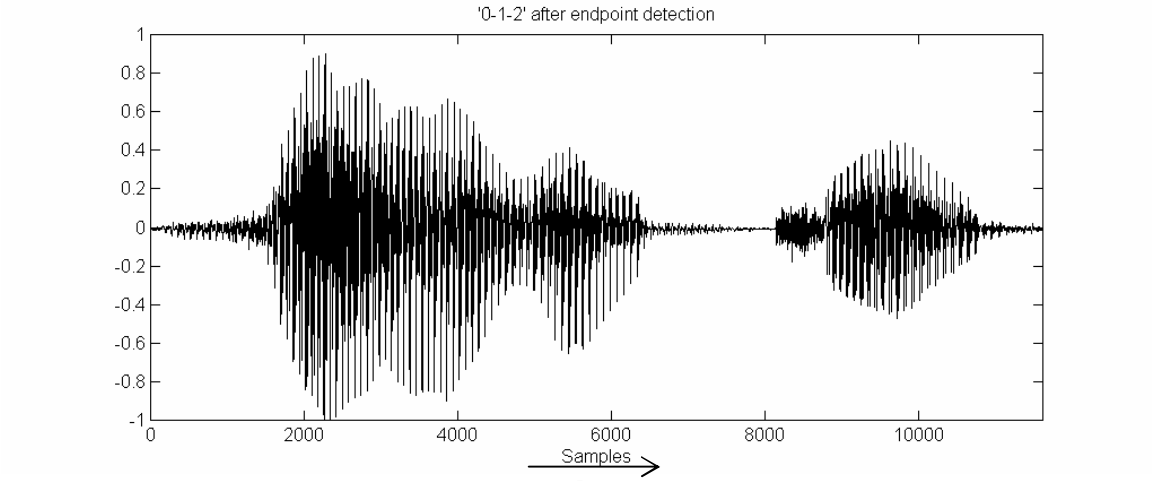
(a)



(b)



(c)



(d)

Figure 2.5 Plot of waveform '0-1-2' before and after endpoint detection
(a) Waveform of '0-1-2' before endpoint detection
(b) Energy levels
(c) ZCR levels
(d) Waveform of '0-1-2' after endpoint detection

3. Hidden Markov Modeling

The speech recognition system developed in this research uses a continuous Hidden Markov Model. From the block diagrams of Figure 2.1, HMMs play an important part in the parameter modeling block of the training phase and the pattern matching block of the recognition phase. This chapter first defines what an HMM is and then describes how it applies to speech recognition.

3.1 Definition of a Hidden Markov Model

An HMM models a doubly stochastic process. One of the stochastic processes, called the observation sequence, is readily observable to the outside world. An example of this would be the outcomes of a coin tossing experiment, or for our case of speech recognition, this would be the cepstral and delta cepstral features from one time frame to the next. The other stochastic process, called the state sequence, is not readily observable and thus hidden. For the coin tossing experiment, this may be which coin was used to determine the outcomes or how the outcomes were determined. Since this state sequence is hidden, it can only be observed and determined through the observation sequence.

For a small vocabulary speech recognition system like the ten digits of the English language, an HMM will usually model an entire word. In a larger vocabulary system, the HMM may model some speech utterances corresponding to subunits of words. For recognition purposes, we want to find the HMM which most likely produced the given

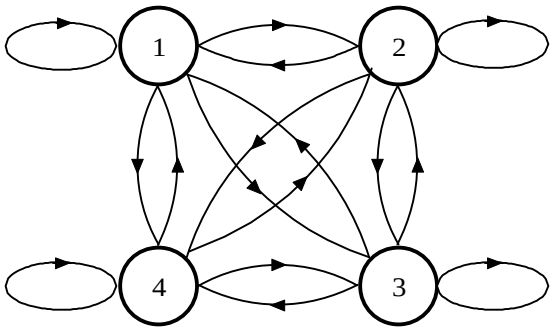
observation sequence. The word corresponding to the latter HMM is the recognized word.

Let us first examine how an HMM works and then how this can be applied to speech recognition [1]. To begin with let us say there are S states in the model, and we have an observation sequence

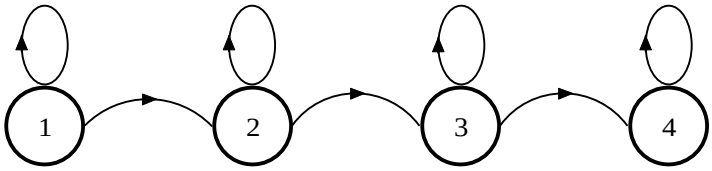
$$\underline{y}(1), \underline{y}(2), \underline{y}(3), \ldots, \underline{y}(t), \ldots, \underline{y}(T)$$

where $\underline{y}(t)$ represents the D-dimensional observation vector generated at time t, and the total number of observations is T. Each observation vector is the outcome from one of the S states. An HMM can be thought of as a "finite state machine." An observation is generated at each time from one of the S states. Then a transition, or jump, to another state (or maybe the same state) occurs and another observation is generated. This process is repeated generating the final observation sequence.

There are usually certain constraints associated with HMMs that make determining the state sequence easier. To begin with, only certain states can be valid starting points for the HMM. In addition, some transitions from one state to the next may not be valid (see Figure 3.1b). Only valid state transitions can occur from one time, t, to the next time, $t + 1$. This can include making a transition to the same state. Finally, only certain states are valid for ending the sequence. Figure 3.1 gives an illustration of different types of HMMs for four states.



(a)



(b)

Figure 3.1 Examples of 4-state HMMs
(a) A fully connected HMM
(b) A left-to-right HMM

Figure 3.1a is an example of a fully connected HMM, or an ergodic model. In this type of model, any state can be a valid starting point and ending point. Also, any particular state can be reached from any of the other states in one transitional jump. In most cases, there will be some constraints placed on the HMM. Figure 3.1b is an example of a left-to-right model where the state sequence must proceed starting from the left and ultimately ending on the right. In this type of model, there is only one valid starting point (State 1) and one valid ending point (State 4). From the diagram, we see that the only valid transitions are to itself or to the next higher state until the final state is reached. This type of model imposes some type of order in the observation sequence since observations

associated with lower-numbered states will occur before observations associated with the higher-numbered states. The fact that this occurs means that a left-to-right model can be applied to our speech recognition system since the same type of order exists in how a word is spoken.

The state sequence is the hidden random process and is denoted as $\underline{x}$ with associated random variables $\underline{x}(t)$. The HMM assumes that a transition will occur at each observation time. The likelihood of these transitions occurring can be quantified by the *state transition probability*. A state transition probability of making a transition from state j to state i is denoted as $a(i|j)$. Stated mathematically, for any arbitrary time, t,

$$a(i|j) = P(\underline{x}(t) = i|\underline{x}(t-1) = j) \tag{3.1}$$

With the assumption that the state transition probability at time t does not depend on the history of the state sequence, the random sequence becomes a first-order Markov process. Since the random sequence takes on discrete integer values which represent the states, this becomes a Markov chain.

Placing all the state transition probabilities into a single matrix yields the *state transition matrix A* which is given by

$$A = \begin{bmatrix} a(1|1) & a(1|2) & \cdots & a(1|S-1) & a(1|S) \\ & \ddots & & & \\ & & a(i|j) & & \\ & & & \ddots & \\ a(S|1) & a(S|2) & \cdots & a(S|S-1) & a(S|S) \end{bmatrix}, \tag{3.2}$$

where S is the total number of states in the model. The state transition probabilities are assumed to be stationary with respect to time. This means that $a(i|j)$ does not depend on

when the transition occurs. Because a transition will occur at every observation time, every column of A will sum to unity.

The *state probability vector* at time t is defined as

$$\pi(t) = \begin{bmatrix} P(x(t)=1) \\ P(x(t)=2) \\ \vdots \\ P(x(t)=S) \end{bmatrix} \tag{3.3}$$

Using this definition, the above expression can be written as

$$\begin{aligned} \pi(t) &= A\pi(t-1) \\ &= A^{t-1}\pi(1) \end{aligned} \tag{3.4}$$

where $\pi(1)$ is the *initial state probability vector*.

The observation sequence is the observed random process and is denoted as $\underline{y}$ with associated random variables $\underline{y}(t)$. Recall that each observation is generated from the state the system is in. Therefore, each state has an associated *observation pdf* which is the means for generating an observation. For state i, the observation pdf is denoted as $f_{\underline{y}(t)|\underline{x}(t)}(\xi|i)$. The random variables $\underline{y}(t)$ are assumed to be independent and identically distributed and therefore do not depend on time t. Thus,

$$f_{\underline{y}|\underline{x}}(\xi|i) = f_{\underline{y}(t)|\underline{x}(t)}(\xi|i) \quad \text{for arbitrary } t. \tag{3.5}$$

All of the parameters of an HMM have now been defined, and the HMM is now formally represented as

$$\mathcal{M} = \left\{ S, \pi(1), A, \left\{ f_{\underline{y}|\underline{x}}(\xi|i), 1 \le i \le S \right\} \right\} \tag{3.6}$$

3.2 Discrete and Continuous Hidden Markov Models

There are basically two types of HMM: the *discrete HMM* and the *continuous HMM*. For a discrete HMM, the observation pdf's for all the states are discrete. This means that the observation sequence can only take on a finite set of values. Vector quantization (VQ) is used to quantize the naturally occurring observations into one of the number of permissible, discrete sets. This is similar to quantizing an analog signal into a discrete signal by passing it through an analog-to-digital converter. The discrete signal will then take one of a finite number of values.

The continuous HMM, which is used in this research, represents the general case where the observation pdf's are continuous and "unquantized". The observation pdf is approximated using a *Gaussian mixture density*. This assumes that the pdf can be approximated by summing *M* weighted multivariate Gaussian pdf's. The observation pdf is of the form

$$f_{\underline{y}|\underline{x}}(\xi|i) = \sum_{m=1}^{M} c_{im}\,\mathcal{N}(\xi;\mu_{im},\Sigma_{im}) \tag{3.7}$$

where c_{im} is the *mixture coefficient* for the *m*th component of state *i*, and $\mathcal{N}(\cdot)$ denotes a multivariate Gaussian pdf with mean μ_{im} and covariance matrix Σ_{im} where the associated probability is given by

$$\mathcal{N}(\xi;\mu_{im},\Sigma_{im}) = \frac{1}{(2\pi)^{D/2}(\det\Sigma_{im})^{1/2}}\, e^{-(\xi-\mu_i)^T\Sigma^{-1}(\xi-\mu_i)/2} \tag{3.8}$$

where *D* is the length of the feature vector. The mixture coefficients must be nonnegative and satisfy the constraint

$$\sum_{m=1}^{M} c_{im} = 1, \qquad 1 \le i \le S \tag{3.9}$$

The *likelihood* for generating observation $\underline{y}(t)$ from state i is defined as

$$b\left(\underline{y}(t)|i\right) = f_{\underline{y}|\underline{x}}\left(\underline{y}(t)|i\right) \tag{3.10}$$

Now that we have formally defined what an HMM is and some of its properties, It is time to consider the two problems at hand for applying an HMM in our speech recognition system: the *training* problem and the *recognition* problem. The training problem involves estimating the parameters which make up the HMM. Once these parameters have been estimated, they can be applied to determine the likelihood if the trained HMM produced an incoming observation sequence. Between the two problems, the recognition problem is easier and is discussed first.

3.3 Recognition Problem

The recognition problem deals with determining the likelihood that a given HMM model produced an incoming observation sequence. There are several techniques which can be applied to compute this likelihood including a forward-backward approach or a state-space approach. Both of the above approaches calculate the likelihood assuming any state sequence is possible. These techniques are discussed in more detail in the literature [1][9][17]. However, the most popular approach (and the approach used in this research) is the Viterbi decoding method [1][9][17]. This method is popular mainly due to its ease of implementation and its efficiency. The Viterbi decoding method is discussed next.

The Viterbi decoding method for computing the likelihood involves finding the optimal state sequence using dynamic programming (DP) techniques utilizing the Bellman Optimality Principle (BOP). DP and BOP will be discussed in the next few pages. The Viterbi method finds the number $P(y, I^* \mid \mathcal{M})$ where

$$I^* = \underset{I}{\operatorname{argmax}}\, P\left(y, I \middle| \mathcal{M}\right) \tag{3.11}$$

represents the optimal state sequence and $I = \{i_1, i_2, ..., i_T\}$ represents any state sequence of length T.

DP techniques are used to solve the Viterbi problem. The problem is tackled by considering a grid as shown in Figure 3.2 where the observation times are laid out on the abscissa while the states are along the ordinate [1]. Each point on the grid can be indexed by the time, state pair (t, i). While searching this grid, two restrictions are imposed.

1. Every path must advance in time by one, and only one, time step. Thus sequential grid points will be of the form (t,i) and (t+1,j) where $1 \le i,j \le S$.
2. The final grid points for any path must be of the form (T,i_f), where i_f constitutes a legal final state in the model.

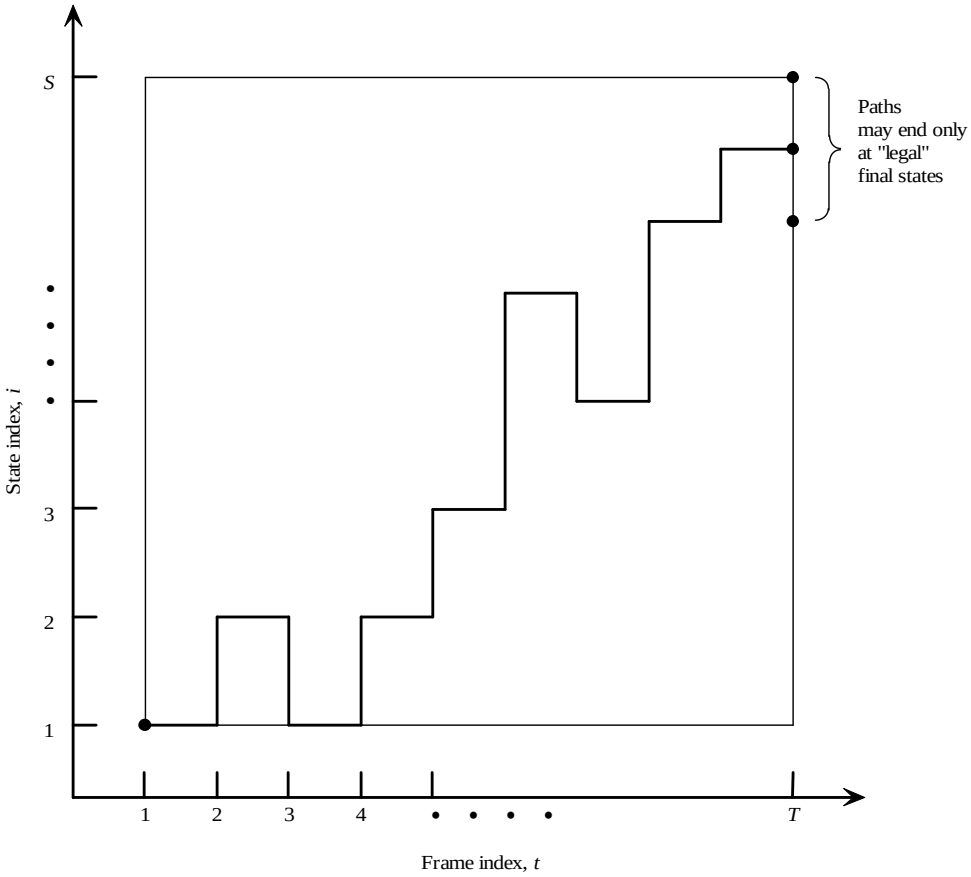


Figure 3.2 Search grid for Viterbi decoding

As the grid is searched, several different types of costs will be defined. In general, these costs are various likelihoods that are being examined. The higher the likelihood is, the better the cost. The first of these is a Type *N* cost which is associated with any node in the grid. A Type *N* cost is defined as follows (the primed notation will be obvious later):

$$\begin{aligned} d'_N(t,i) &= b\big(y(t)\big|i\big) \\ &= P\big(\underline{y} = y(t)\big|\underline{x}(t) = i\big) \end{aligned} \tag{3.12}$$

In addition, a Type *T* cost denoting the cost of making a transition in the grid is defined as

$$d'_T\left[(t,i)|(t-1,j)\right] = a(i|j) = P(\underline{x}(t)=i|\underline{x}(t-1)=j) \tag{3.13}$$

for any arbitrary i and j and for arbitrary $t > 1$. For the initial state probability, we assume that all paths start from a fictitious node (0,0) and making a Type T cost of

$$d'_T\left[(1,i)|(0,0)\right] = P(\underline{x}(1)=i) = \pi_i(1) \tag{3.14}$$

to any state i. The accumulated cost (Type B) associated with any transition from (t-1,j) to (t,i) is then defined as

$$d'_B\left[(t,i)|(t-1,j)\right] = d'_T\left[(t,i)|(t-1,j)\right]d'_N(t,i) = a(i|j)b(y(t)|i) \tag{3.15}$$

for $t > 1$, and

$$d'_B\left[(1,i)|(0,0)\right] = d'_T\left[(1,i)|(0,0)\right]d'_N(1,i) = \pi_i(1)b(y(1)|i) \tag{3.16}$$

for $t = 1$.

Considering a complete path through the grid, the total accumulated cost is the product of the Type B costs and is given as

$$\begin{aligned} D' &= \prod_{t=1}^{T} d_B\left[(t,i_t)|(t-1,i_{t-1})\right] \\ &= \prod_{t=1}^{T} a(i_t|i_{t-1})b(y(t)|i_t) \\ &= L(y,I|\mathcal{M}) \end{aligned} \tag{3.17}$$

where we define

$$a(i_1|i_0) = \pi_i(1) \tag{3.18}$$

The best state sequence, I^*, yields the *maximum* cost and the total optimal cost is

$$[D']^* = L\left(y, I^* \middle| \mathcal{M}\right) \tag{3.19}$$

Note that equations (3.15), (3.16), and (3.17) involve forming the products of likelihoods, which will often become very small and cause numerical problems in the solution. To circumvent this problem, negative logarithms are taken. This transforms the multiplications into additions and changes the problem of determining the optimal state sequence into finding the *minimum* cost path.

By taking negative logarithms, the Type *N* cost becomes

$$\begin{aligned} d_N(t, i_t) &= -\log\left[d'_N(t, i_t)\right] \\ &= -\log\left[b\left(y(t) \middle| i_t\right)\right] \end{aligned} . \tag{3.20}$$

The Type *T* cost then becomes

$$\begin{aligned} d_T\left[(t,i) \middle| (t-1, j)\right] &= -\log d'_T\left[(t,i) \middle| (t-1, j)\right] \\ &= -\log\left[a\left(i \middle| j\right)\right] \end{aligned} \tag{3.21}$$

for $t > 1$, and

$$\begin{aligned} d_T\left[(t,i) \middle| (t-1, j)\right] &= -\log\left[d'_T\left[(1,i) \middle| (0,0)\right]\right] \\ &= -\log\left[\pi_i(1)\right] \end{aligned} \tag{3.22}$$

for $t = 1$. The Type *B* cost is then

$$\begin{aligned} d_B\left[(t,i) \middle| (t-1, j)\right] &= -\log\left[d'_B\left[(t,i) \middle| (t-1, j)\right]\right] \\ &= d_T\left[(t,i) \middle| (t-1, j)\right] + d_N(t,i) \\ &= -\log\left[a\left(i \middle| j\right)\right] - \log\left[b\left(y(t) \middle| i\right)\right] \end{aligned} \tag{3.23}$$

for $t > 1$, and

$$
\begin{aligned}
d_B[(1,i)|(0,0)] &= -\log[d'_B[(1,i)|(0,0)]] \\
&= d_T[(1,i)|(0,0)] + d_N(1,i) \\
&= -\log[\pi_i(1)] - \log[b(y(1)|i)]
\end{aligned}
\tag{3.24}
$$

for $t = 1$. Finally, the total accumulated cost for a given path is then

$$
\begin{aligned}
D &= \sum_{t=1}^{T} d_B[(t,i_t)|(t-1,i_{t-1})] \\
&= \sum_{t=1}^{T} \{-\log[a(i_t|i_{t-1})] - \log[b(y(t)|i_t)]\}
\end{aligned}
\tag{3.25}
$$

where

$$
D = -\log D' \tag{3.26}
$$

The task at hand then becomes determining the optimal state sequence given the observation sequence, in order to determine the optimal likelihood. The optimal state sequence is solved using DP techniques. The DP algorithm presented below incorporates the BOP [1]. The BOP states that to find the minimum cost path segment from node $(0,0)$ to node (t_k, i_k), denoted $(0,0) \underset{*}{\rightarrow} (t_k, i_k)$, that passes through a predecessor node (t_{k-1}, i_{k-1}), it is not necessary to reexamine all the paths leading from $(0,0)$ to (t_{k-1}, i_{k-1}) enroute to (t_k, i_k). It is sufficient to simply extend $(0,0) \underset{*}{\rightarrow} (t_{k-1}, i_{k-1})$ over the minimal cost path segment possible to reach (t_k, i_k).

Using the BOP and the DP techniques, the Viterbi algorithm can now be presented. Before the algorithm is presented, several variables are explained first. The partial joint likelihood of occurrence at sequence time t of the optimal partial state sequence $\{i_1^*, i_2^*, \cdots, i_t^*\}$ and the partial observation sequence $\{y(1), y(2), \cdots, y(t)\}$ is given by

$D_{\min}(t,i_t)$. In other words, $D_{\min}(t,i_t)$ represents the distance from (0,0) to (t,i_t) over the best path. Grouping all of these joint likelihoods together gives the optimal partial joint likelihood matrix $D_{\min}$. In addition, knowledge of the state sequence associated with the best path through the HMM might be useful. $\Psi(t,i_t)$ is defined to be the best last state on the optimal partial path ending at (t,i_t). In other words, this is an index of the predecessor node $(t-1,i_{t-1})$ to (t,i_t). Clearly, the optimal path from (0,0) to (t,i_t) can be found by backtracking using $\Psi(t,i_t)$. Using these variables, the Viterbi algorithm is now presented below [1][17]:

Step 1: Initialization

- $D_{\min}(1,i_1) = -\log[\pi_{i_1}(1)] - \log[b(y(1)|i_1)] \quad 1 \le i_1 \le S$ (3.27)
- $\Psi(1,i_1) = 0$ (3.28)

Step 2: Recursion

For $2 \le t \le T, \quad 1 \le i_t \le S$

- $D_{\min}(t,i_t) = \min_{1 \le i_{t-1} \le S} \{D_{\min}(t-1,i_{t-1}) - \log[a(i_t|i_{t-1})]\} - \log[b(y(t)|i_t)]$ (3.29)
- $\Psi(t,i_t) = \underset{1 \le i_{t-1} \le S}{\operatorname{argmin}} \{D_{\min}(t-1,i_{t-1}) - \log[a(i_t|i_{t-1})]\}$ (3.30)

Step 3: Termination

- $D^* = \min_{\text{legal } i_T} [D_{\min}(T,i_T)]$ (3.31)

- $i_T^* = \underset{\text{legal } i_T}{\operatorname{argmin}}\left[D_{\min}(T, i_T)\right]$ (3.32)

Step 4: Path (state sequence) backtracking

For $t = T-1, T-2, \ldots, 1$

- $i_t^* = \Psi\left(t+1, i_{t+1}^*\right)$ (3.33)

The optimal likelihood is given by the quantity D^*, and the optimum state sequence is given by $I^* = \left\{i_1^*, i_2^*, \cdots, i_T^*\right\}$.

3.4 Training Problem

Now we turn our attention to the problem of training. The purpose of training the system is to estimate the parameters that make up each HMM model. Multiple observation sequences from the same model are needed. There should be a sufficient number of training sequences such that all the statistical variations can be represented across the multiple utterances. The training aims at estimating the state probability matrix, the initial state probability distribution, and the mean vectors, covariance matrices, and mixture weights of the observation pdf's from an arbitrary initial model. Like the recognition problem, there are different approaches for the training. All the approaches involve an iterative procedure to estimate these parameters. Among the most popular for discrete HMMs is included the forward-backward (Baum-Welch) reestimation procedure. This training algorithm is comparable to the forward-backward algorithm of recognition.

The F-B reestimation procedure is discussed in the literature [1][9][17]. The training algorithm used in this research is the Viterbi reestimation, which uses the Viterbi decoding and backtracking for recognition [1].

The Viterbi reestimation algorithm provides a simple and very efficient procedure for estimating the parameters of a HMM. First, some notation is defined. We define $\underline{u}$ to be a random process with random variables $\underline{u}(t)$ that model transitions at time t. The following notations are defined:

$u_{j|i}$ = label for a transition from state i to state j

$u_{\bullet|i}$ = set of transitions exiting state i

To begin the reestimation, multiple utterances, say L, are needed. In addition, we start with an initial model $\mathcal{M}$. Viterbi decoding and backtracking is used to determine the path (optimal state sequence) for each of the L utterances. While performing the Viterbi decoding for all the utterances, the following numbers are noted:

$n(u_{j|i})$ = number of transitions $u_{j|i}$

$n(u_{\bullet|i})$ = number of transitions from the set $u_{\bullet|i}$

The parameters for the new model $\overline{\mathcal{M}}$ are estimated next. The state transition probabilities for the new model are estimated by

$$a(j|i) = \frac{n(u_{j|i})}{n(u_{\bullet|i})} \tag{3.34}$$

In addition, for each of the S states, we have a set of observations (from the L training sequences) that occur within the state. Recall that M Gaussian mixture densities are used to characterize the observation pdf, $f_{\underline{y}|\underline{x}}\left(\xi|i\right)$. Estimates of the mean vector for each mixture component, covariance matrix of each mixture component, and the mixture coefficients for each state need to be found using the data in each set. These can be estimated using several types of algorithms which are discussed in the next section.

The new model, $\overline{\mathcal{M}}$, is then compared with the previous model, $\mathcal{M}$, to determine how statistically similar the two models are. One such distance measure is

$$D'\left(\mathcal{M},\overline{\mathcal{M}}\right)=\frac{1}{\overline{T}}\left[\log\left[\mathcal{L}\left(\overline{y}|\mathcal{M}\right)\right]-\log\left[\mathcal{L}\left(\overline{y}|\overline{\mathcal{M}}\right)\right]\right] \tag{3.35}$$

where $\overline{y}$ is a length $\overline{T}$ sequence generated by $\overline{\mathcal{M}}$. In general, this distance measure is not symmetric in the sense that

$$D'\left(\mathcal{M},\overline{\mathcal{M}}\right)\neq D'\left(\overline{\mathcal{M}},\mathcal{M}\right) \tag{3.36}$$

To obtain a distance measure that is symmetric with respect to the models, we use

$$D\left(\mathcal{M},\overline{\mathcal{M}}\right)=\frac{D'\left(\mathcal{M},\overline{\mathcal{M}}\right)+D'\left(\overline{\mathcal{M}},\mathcal{M}\right)}{2} \tag{3.37}$$

This distance is compared with a threshold and, if it is exceeded, the initial model $\mathcal{M}$ is replaced with the new model $\overline{\mathcal{M}}$. The entire process is repeated in an iterative fashion until the model distance falls below the threshold, where convergence is assumed.

3.5 Observation Probability Density Function

Recall that the observation pdf's are assumed to be Gaussian mixture densities of the following form:

$$f_{\underline{y}}(\xi) = \sum_{m=1}^{M} c_m \mathcal{N}(\xi; \mu_m, \Sigma_m) \tag{3.38}$$

where M is the number of mixture components, and c_m, μ_m, and Σ_m are the mixture coefficient, mean vector, and covariance matrix for the mth Gaussian pdf, which is given as

$$\mathcal{N}(\xi; \mu_m, \Sigma_m) = \frac{1}{(2\pi)^{D/2}(\det \Sigma_m)^{1/2}} e^{-(\xi-\mu_m)^T \Sigma^{-1}(\xi-\mu_m)/2} \tag{3.39}$$

In general, the individual entities in each of the observation vectors can be assumed to be independent from each other. Thus, the covariance matrix for the mth mixture component will be a diagonal matrix with the variance of each entity along the diagonal. To simplify matters, we will consider a variance vector σ_m^2 to be the variance for the mth Gaussian pdf, which contains the diagonal entries of the covariance matrix. Equation (3.39) then reduces to the product of the individual Gaussian pdf's for each entity given by the expression

$$\mathcal{N}(\xi; \mu_m, \sigma_m^2) = \prod_{d=1}^{D} \mathcal{N}(\xi_d; \mu_{m_d}, \sigma_{m_d}^2) \tag{3.40}$$

where $\mathcal{N}(\xi_d; \mu_{m_d}, \sigma_{m_d}^2)$ is the Gausian pdf for the dth entity with mean μ_{m_d} and variance $\sigma_{m_d}^2$.

There exist several algorithms that attempt to solve the mixture density problem for the observation pdf by estimating the mixture coefficients, the mean vectors, and the covariance matrices. Fast and efficient methods for obtaining these estimates include the clustering algorithms. Clustering algorithms group the vectors into different clusters and the means and variances are calculated from each cluster group. These algorithms include the many different types of *k*-means algorithms. An iterative solution to this problem can be found using the expectation maximization (EM) algorithm. For this research, a segmental *k*-means algorithm and the EM algorithm were examined and contrasted. Both methods are discussed below.

3.5.1 The Segmental k-means Algorithm

The segmental *k*-means algorithm is an example of a clustering algorithm that quickly estimates a mixture density representation. This algorithm is a recursive algorithm that starts with two clusters and finishes with *M* clusters. The algorithm is given below.

1. Initialization.
    - Compute mean vector from all the vectors belonging to the observation pdf, μ.
    - Group the vectors into two clusters depending on whether a vector is closer to $\mu - \varepsilon$ or $\mu + \varepsilon$, where ε is a small positive vector. A Euclidean distance measure is used to determine which cluster each vector belongs to.

2. Recursion
    - For each cluster, compute the mean vector of that cluster.
    - Divide each mean vector into two: one slightly less than the mean and one slightly greater than the mean.
    - Regroup all the vectors by determining which perturbed mean vector is closer, using a Euclidean distance measure.
    - Repeat until the vectors have been grouped into *M* clusters.
3. Termination
    - For each of the *M* clusters, calculate the sampled mean and sampled variance vector. These will be used as estimates for the actual mean vector and variance vector in the observation pdf.
    - The mixture weight for each cluster is the number of vectors in the cluster divided by the total number of vectors modeling the pdf.

The segmental *k*-means algorithm is a fast and efficient method for estimating the parameters in a mixture density representation. However, a drawback for this method, or any clustering method, is that they must assign each vector to belong to any one of *M* clusters by using some type of distance measure criterion. Clearly this is not the best solution when the mixture densities overlap. The vectors can possibly be assigned to the wrong cluster, which can skew the means and decrease the variances relative to their true values.

3.5.2 The EM Algorithm

The EM (expectation-maximization) algorithm is an iterative algorithm for finding a mixture density representation [18][19]. There are two major steps in this algorithm: an expectation step followed by a maximization step. The expectations are with respect to the unknown underlying variables using current estimates of the parameters and conditioned upon the observations. The maximization then provides new estimates of the parameters. A full development of this algorithm is beyond the scope of this thesis.

By defining $c_i^c = \{c_{i1}^c, c_{i2}^c, ..., c_{iM}^c\}$ as the current estimates for the mixture coefficients, $\mu_i^c = \{\mu_{i1}^c, \mu_{i2}^c, ..., \mu_{iM}^c\}$ as the current estimates for the mean vectors, and $\sigma_i^{c2} = \{\sigma_{i1}^{c\,2}, \sigma_{i2}^{c\,2}, ..., \sigma_{iM}^{c\,2}\}$ as the current estimates for the variance vectors, the next estimates of the EM algorithm are given as

$$c_m^+ = \frac{1}{N}\sum_{k=1}^{N} \frac{c_m^c \mathcal{N}\left(\xi_k; \mu_m^c, \sigma_m^{c\,2}\right)}{f_{\underline{y}}(\xi_k)} \tag{3.41}$$

$$\mu_m^+ = \left\{\sum_{k=1}^{N} \xi_k \frac{c_m^c \mathcal{N}\left(\xi_k; \mu_m^c, \sigma_m^{c\,2}\right)}{f_{\underline{y}}(\xi_k)}\right\} \Bigg/ \left\{\sum_{k=1}^{N} \frac{c_m^c \mathcal{N}\left(\xi_k; \mu_m^c, \sigma_m^{c\,2}\right)}{f_{\underline{y}}(\xi_k)}\right\} \tag{3.42}$$

$$\sigma_m^{+\,2} = \left\{\sum_{k=1}^{N} \mathrm{diag}\left(\xi_k - \mu_m^+\right)\left(\xi_k - \mu_m^+\right) \frac{c_m^c \mathcal{N}\left(\xi_k; \mu_m^c, \sigma_m^{c\,2}\right)}{f_{\underline{y}}(\xi_k)}\right\} \Bigg/ \left\{\sum_{k=1}^{N} \frac{c_m^c \mathcal{N}\left(\xi_k; \mu_m^c, \sigma_m^{c\,2}\right)}{f_{\underline{y}}(\xi_k)}\right\} \tag{3.43}$$

where N is the number of feature vectors in the histogram and $c_i^+ = \{c_{i1}^+, c_{i2}^+, ..., c_{iM}^+\}$, $\mu_i^+ = \{\mu_{i1}^+, \mu_{i2}^+, ..., \mu_{iM}^+\}$, and $\sigma_i^{+2} = \{\sigma_{i1}^{+\,2}, \sigma_{i2}^{+\,2}, ..., \sigma_{iM}^{+\,2}\}$ are the new estimates for the mixture

coefficients, mean vectors, and variance vectors respectively. After each iteration, the change in the likelihood from the previous iteration is computed and compared to a threshold. If this change is greater than the threshold, the iteration is repeated with the new estimates as current estimates. Once this change is below the threshold, convergence is assumed.

The EM algorithm, in general, provides good global convergence. However, there may be some local maxima in the distribution function causing the EM algorithm to not necessarily converge to the global maximum. Thus the initial estimates need to be "good" to guarantee convergence to the global maximum. As discussed earlier, the segmental *k*-means algorithm can be used to obtain "good" initial estimates for the EM algorithm. This approach was observed to always converge correctly to the global maximum.

This concludes the discussion on HMMs. The next chapter describes the implementation of the speech recognition system on isolated words using continuous HMMs to model the individual words.

4. Implementation and Results

This chapter discusses specific details regarding the implementation of the speech recognition system and its performance results on isolated words. The system was mainly implemented and tested on a PC with a 16-bit sound card and on an ADSP-2181 processor which is manufactured by Analog Devices. The PC implementation is discussed first.

4.1 PC Implementation

The speech recognition system was first implemented and tested on a PC. A 16-bit sound card was used to take in all the speech utterances in the form of a .WAV file. MATLAB was mainly used for the signal processing, and a C program was written for the training procedure. We first analyze the training procedure and report some of its findings on known models.

4.1.1 The Training Analysis

Before we discuss the actual implementation of the speech recognition system, we first tested the validity of the training program by performing some controlled experiments and seeing how closely the parameters generated from the training program matched the actual parameters. Both the segmental *k*-means algorithm and the EM algorithm were analyzed for estimating the observation pdf.

To test the training procedure, an observation sequence was generated from a known model with "overlapping" Gaussian mixture density components. To begin with, a series of observation vectors containing three elements were generated using a 3-state left-to-right HMM with the following transition matrix:

$$A = \begin{bmatrix} 0.8 & 0 & 0 \\ 0.2 & 0.3 & 0 \\ 0 & 0.7 & 1 \end{bmatrix}$$

and with the following 4-mixture mean vectors, variance vectors, and mixture weights for each state:

State 1:

$$\mu_1 = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 7 & 3 & 5 & 8 \\ -3 & -1 & -4 & 0 \end{bmatrix}$$

$$\sigma_1^2 = \begin{bmatrix} 2 & 1 & 0.5 & 3 \\ 0.5 & 0.7 & 3 & 1 \\ 1.5 & 1.2 & 3 & 1 \end{bmatrix}$$

$$c_1 = \begin{bmatrix} 0.5 & 0.2 & 0.1 & 0.2 \end{bmatrix}$$

State 2:

$$\mu_2 = \begin{bmatrix} 7 & 9 & 11 & 6 \\ 0 & 3 & 2 & 5 \\ -4 & -6 & -8 & -5 \end{bmatrix}$$

$$\sigma_2^2 = \begin{bmatrix} 0.5 & 2 & 2.5 & 1 \\ 1.5 & 1 & 2.5 & 3 \\ 2 & 1 & 3 & 0.3 \end{bmatrix}$$

$$c_2 = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

State 3:

$$\mu_3 = \begin{bmatrix} 2 & 5 & 7 & 3 \\ -2 & -1 & -3 & -5 \\ -1 & 3 & 2 & 6 \end{bmatrix}$$

$$\sigma_3^2 = \begin{bmatrix} 2.5 & 3 & 3 & 0.2 \\ 3 & 2 & 1.5 & 1 \\ 0.5 & 1 & 2 & 3 \end{bmatrix}$$

$$c_3 = \begin{bmatrix} 0.1 & 0.4 & 0.3 & 0.2 \end{bmatrix}$$

To test the training program 200 observation sequences were generated. The Viterbi reestimation procedure was used to estimate the parameters for the HMM. As an initial guess, we assume that the different states occur an almost equal number of times for all the observation sequences. Based on this assumption, the observation sequences were segmented into state sequences. Both the segmental *k*-means and the EM algorithms were used to obtain initial estimates for the mean vectors, variance vectors, and the mixture weights (denoted $\hat{\mu}$, $\hat{\sigma}$, and $\hat{c}$ respectively). The training algorithm was run starting from 25 random initial models. The initial models were generated using

$$\bar{\mu} = \frac{\hat{\mu}}{2} + r\hat{\mu} \tag{4.1}$$

$$\bar{\sigma}^2 = \frac{\hat{\sigma}^2}{2} + r\hat{\sigma}^2 \tag{4.2}$$

$$\bar{c} = \hat{c} \tag{4.3}$$

where r is a random scalar variable, uniformly distributed between 0 and 1.

Using (3.37), the distances were computed between the models from two successive iterations. Convergence was assumed when this distance was below a threshold of 0.1. The model that produced the largest likelihood across the entire training set was used as the final model. Two training methods were run. The first one used the segmental *k*-means algorithm to estimate the observation pdf, while the second method used the EM algorithm. After running the training programs the following estimates were generated by the respective methods.

The estimates from the segmental *k*-means algorithm were:

$$A = \begin{bmatrix} 0.7886 & 0 & 0 \\ 0.2114 & 0.3651 & 0 \\ 0 & 0.6349 & 1 \end{bmatrix}$$

and

State 1

$$\mu_1 = \begin{bmatrix} 0.7308 & 2.8948 & 2.9844 & 7.5638 \\ 6.6129 & 2.8241 & 7.0093 & 8.0777 \\ -3.0018 & -1.1175 & -2.7686 & -0.5994 \end{bmatrix}$$

$$\sigma_1^2 = \begin{bmatrix} 2.0208 & 1.3753 & 1.8501 & 4.7655 \\ 0.9319 & 0.9636 & 0.7383 & 0.8026 \\ 1.6361 & 1.5159 & 2.6018 & 3.4260 \end{bmatrix}$$

$$c_1 = \begin{bmatrix} 0.4165 & 0.2072 & 0.1660 & 0.2104 \end{bmatrix}$$

State 2

$$\mu_2 = \begin{bmatrix} 5.7300 & 9.3901 & 11.5606 & 6.6676 \\ 1.3781 & 2.7128 & 2.6033 & 3.7581 \\ -4.2023 & -6.2182 & -8.2190 & -5.3367 \end{bmatrix}$$

$$\sigma_2^2 = \begin{bmatrix} 2.9219 & 0.8135 & 2.0035 & 1.2444 \\ 5.5697 & 2.5964 & 3.0839 & 5.4828 \\ 1.4915 & 1.4121 & 2.4015 & 0.8558 \end{bmatrix}$$

$$c_2 = \begin{bmatrix} 0.2952 & 0.1841 & 0.1873 & 0.3333 \end{bmatrix}$$

State 3

$$\mu_3 = \begin{bmatrix} 2.1437 & 4.8196 & 6.7477 & 4.9505 \\ -1.3211 & -1.0139 & -3.7206 & -3.1523 \\ 0.4982 & 2.3533 & 4.1401 & 3.5058 \end{bmatrix}$$

$$\sigma_3^2 = \begin{bmatrix} 1.8902 & 1.0827 & 6.6205 & 2.8444 \\ 2.9301 & 2.0203 & 2.5168 & 2.6547 \\ 3.6225 & 1.7408 & 7.0414 & 3.3078 \end{bmatrix}$$

$$c_3 = \begin{bmatrix} 0.1688 & 0.2281 & 0.2462 & 0.3568 \end{bmatrix}$$

The estimates from the EM algorithm were:

$$A = \begin{bmatrix} 0.7914 & 0 & 0 \\ 0.2086 & 0.3377 & 0 \\ 0 & 0.6623 & 1 \end{bmatrix}$$

and

State 1

$$\mu_1 = \begin{bmatrix} 0.7244 & 3.1906 & 1.6712 & 7.8958 \\ 7.0971 & 3.0081 & 6.7031 & 8.0191 \\ -2.7977 & -1.4909 & -3.4337 & -0.0061 \end{bmatrix}$$

$$\sigma_1^2 = \begin{bmatrix} 1.3675 & 0.9528 & 3.1760 & 3.2391 \\ 0.3445 & 1.0255 & 0.7613 & 0.9254 \\ 1.5680 & 2.3368 & 1.8390 & 0.9050 \end{bmatrix}$$

$$c_1 = \begin{bmatrix} 0.2862 & 0.2414 & 0.2749 & 0.1976 \end{bmatrix}$$

State 2

$$\mu_2 = \begin{bmatrix} 6.8526 & 8.0683 & 10.8004 & 6.2970 \\ 0.0498 & 3.5702 & 2.1328 & 4.8671 \\ -4.3478 & -5.9834 & -7.8167 & -4.9207 \end{bmatrix}$$

$$\sigma_2^2 = \begin{bmatrix} 0.4748 & 7.9965 & 2.3808 & 1.3608 \\ 0.9402 & 1.4479 & 2.8391 & 3.1326 \\ 1.6686 & 1.5294 & 2.9476 & 0.2457 \end{bmatrix}$$

$$c_2 = \begin{bmatrix} 0.2744 & 0.2519 & 0.2444 & 0.2293 \end{bmatrix}$$

State 3

$$\mu_3 = \begin{bmatrix} 3.0274 & 5.0959 & 7.4025 & 2.9863 \\ -1.8453 & -0.9844 & -2.8943 & -4.9725 \\ 0.7678 & 3.0938 & 2.1376 & 6.1694 \end{bmatrix}$$

$$\sigma_3^2 = \begin{bmatrix} 3.7906 & 1.5187 & 2.3980 & 0.1777 \\ 2.8609 & 1.8504 & 1.3345 & 1.0325 \\ 3.7617 & 0.8271 & 1.9956 & 2.8086 \end{bmatrix}$$

$$c_3 = \begin{bmatrix} 0.2238 & 0.2958 & 0.2875 & 0.1928 \end{bmatrix}$$

Since we know the true statistics of the observation data, the estimated parameters can be compared with the true parameters. Figures 4.1 through 4.3 show the true histogram of the first element for all the observations and the histogram of this observation based on Viterbi decoding for the three states. The histograms for the second and third elements can be found in Appendix A.

The histograms show that regardless of which method is used, the observations which are assigned to the states by Viterbi decoding are almost identical to those truly generated in the states, despite having "overlapping" Gaussian mixture components. The observation pdf's were then compared. Figures 4.4 through 4.6 show the true pdf and the reestimated pdf from both training procedures for the first element. The pdf's for the second and third elements can be found in Appendix A.

From the results shown above, it is evident that the EM algorithm for estimating the observation pdf performs somewhat better than the segmental *k*-means algorithm. Figures 4.4 through 4.6 demonstrate that the estimated observation pdf generated using the EM algorithm matches the true pdf better than the estimates generated by the

segmental *k*-means algorithm. This difference in performance can, likely, be attributed to the fact that using the segmental *k*-means algorithm, some of the vectors could have been assigned to the wrong cluster. Consequently, the EM algorithm is incorporated into the training algorithm for the actual speech recognition despite the longer computational time for convergence of the algorithm.

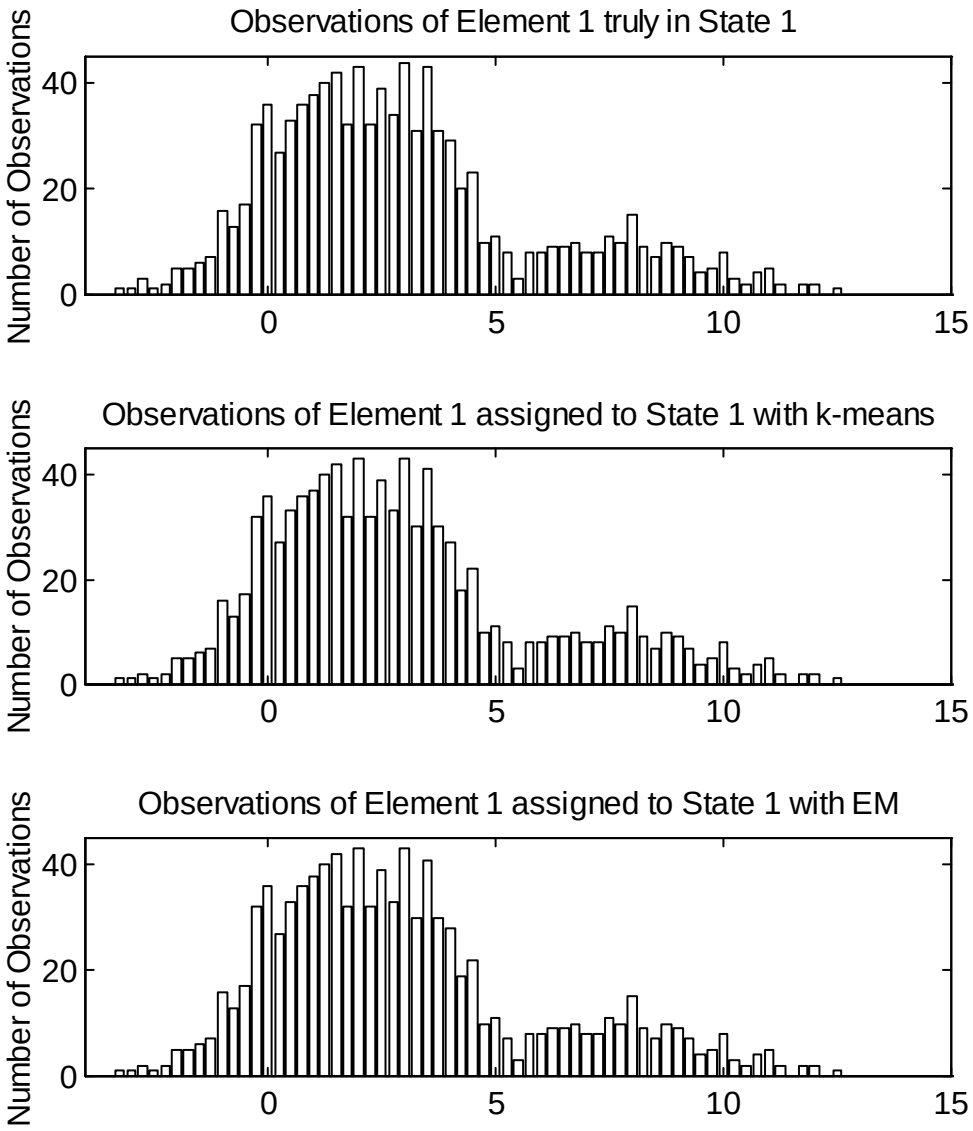


Figure 4.1 Histograms for Element 1, State 1

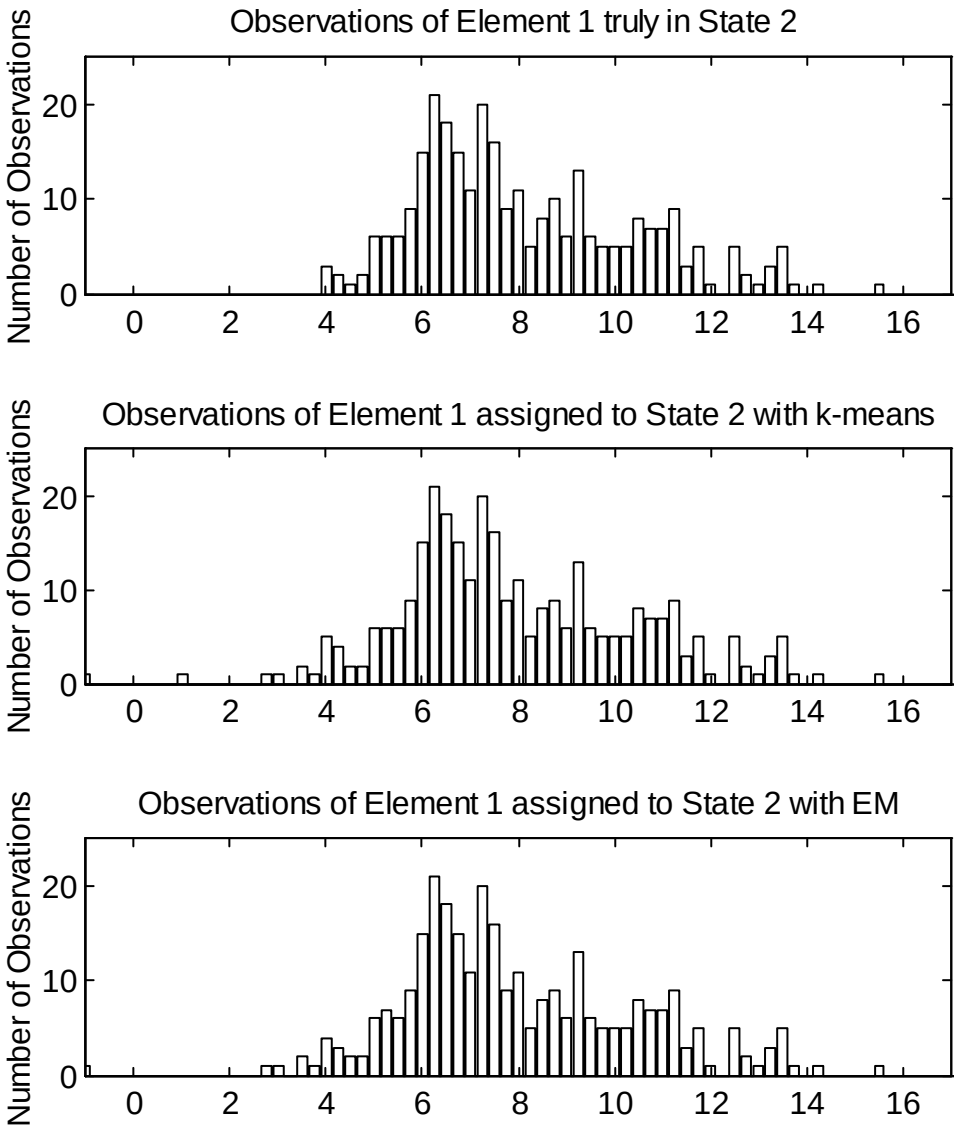


Figure 4.2 Histograms for Element 1, State 2

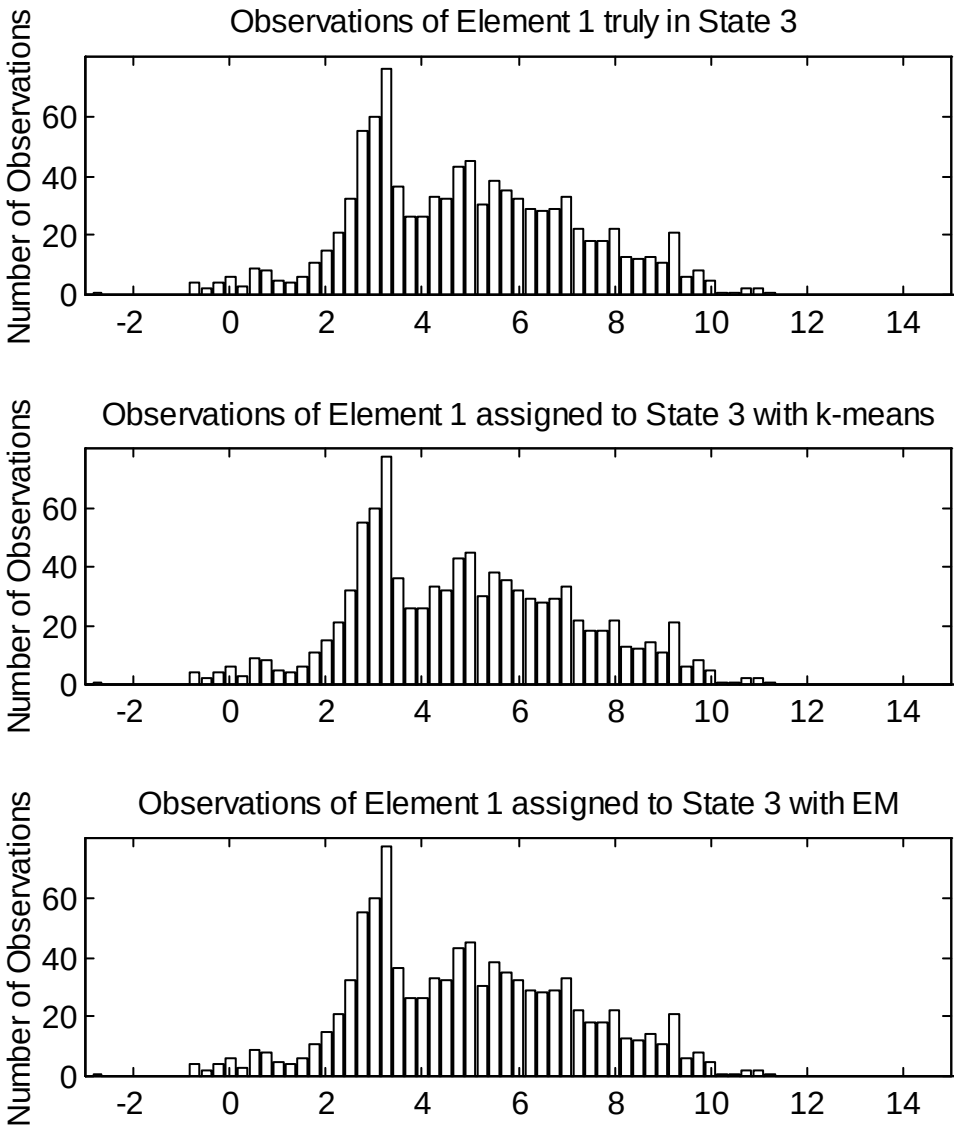


Figure 4.3 Histograms for Element 1, State 3

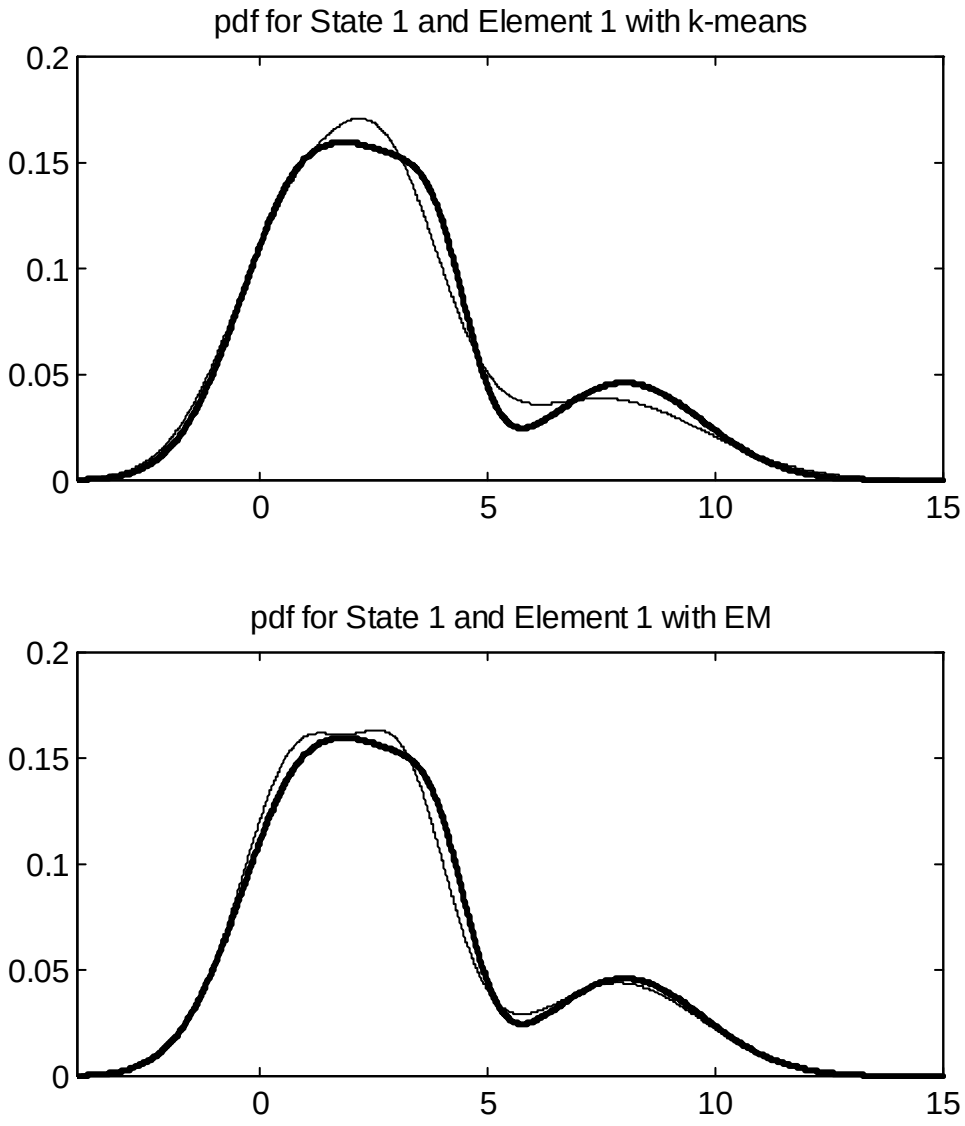


Figure 4.4 True (▬) and Estimated (—) observation pdf's for Element 1, State 1

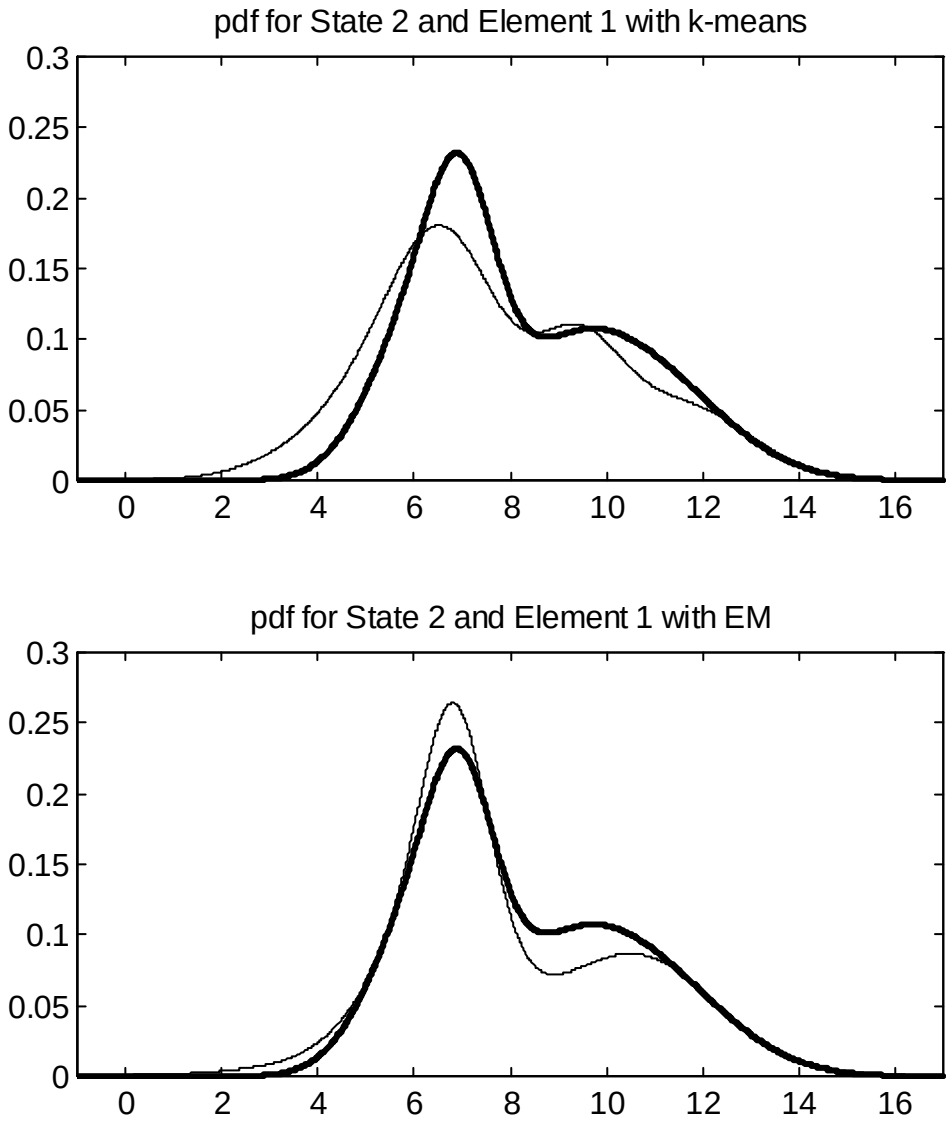


Figure 4.5 True (▬) and Estimated (—) observation pdf's for Element 1, State 2

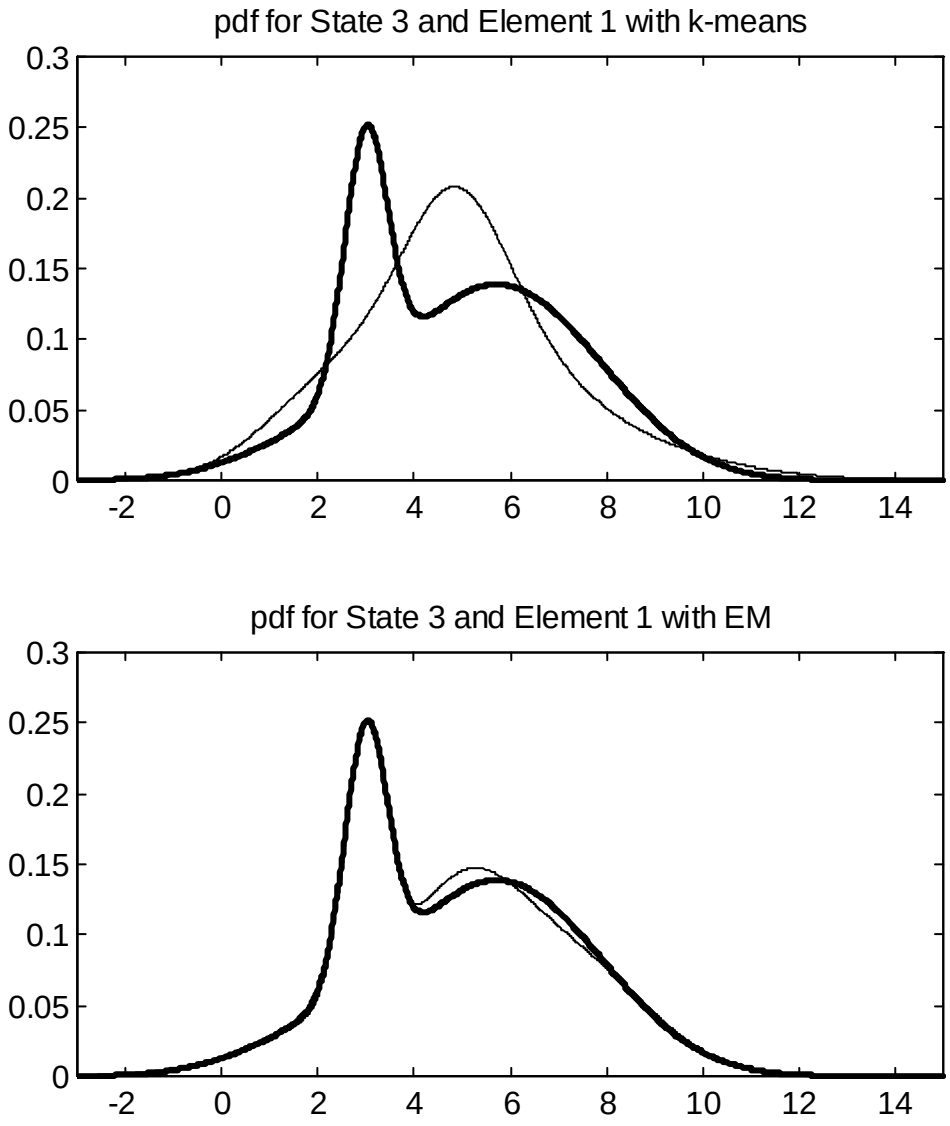


Figure 4.6 True (▬) and Estimated (—) observation pdf's for Element 1, State 3

4.1.2 Results on Words

Now we discuss the speech recognition part. A left-to-right HMM is used to model an entire word. The vocabulary consists of the ten digits (0 through 9). In addition, 7 states and 4 mixture densities were used based on past research [20].

For the training phase, multiple utterances of a linguistic unit need to be present. For the ten digits of the English alphabet, we let one HMM model an entire word. Of each digit, 50 speech utterances were taken in using a 16 bit sound card, using a sampling rate of 11,025 Hz. To include some variability in how each digit is spoken, the first 25 utterances were recorded at one sitting, and the last 25 utterances were recorded at a different sitting. In addition, each speech utterance for the training phase was either taken in isolation or out of context from a string of digits. The word tends to sound different when spoken in isolation or as part of a string. By doing this, we try to incorporate all the different ways the word can be spoken. MATLAB was then used to calculate the cepstral and delta cepstral coefficients and to determine the endpoints of the word using the methods described in Section 2.2. A C program was written to estimate the parameters for the HMMs using 7 states and 4 mixture densities. The training phase used Viterbi reestimation with the EM algorithm to estimate all the parameters for the HMMs as described in Section 3.4.

Once these models have been estimated, the speech recognition system was tested on isolated words. For the recognition phase, the test utterances were taken in the same manner as the training utterances using a 16 bit sound card sampled at 11,025 Hz. The recognition phase was not performed in real time. The feature vectors were obtained as

before using the method described in Section 2.2. The Viterbi method was used for the pattern matching and unit identification as described in Section 3.3. Since each HMM models an entire word, the structural composition block is not needed for isolated words.

Recognition tables were generated to evaluate the performance of the system. An associated figure of merit is included to give an idea of how well the system works. The figure of merit is simply the sum of all the elements along the diagonal in the recognition table which yields the total number of correctly recognized utterances.

Table 4.1 shows the results from the training set. Here we see that the figure of merit for this set of utterances is 500, which represents a 100% recognition rate. This high recognition rate is not surprising since these same utterances were used to generate the model parameters. Table 4.2 shows the results from the testing set. The testing data produced a figure of merit of 498, which represents a 99.6% recognition rate. While not as high as the training set, we see that the models produced by the training set are a good representation for the vocabulary at hand. We see that the recognition system on the PC performs very well on isolated words.

Table 4.1 Recognition rates for the training set on MATLAB

	Input words for recognition									
Output	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	50	0	0	0	0	0	0	0
3	0	0	0	50	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	0	0	50	0	0	0	0
6	0	0	0	0	0	0	50	0	0	0
7	0	0	0	0	0	0	0	50	0	0
8	0	0	0	0	0	0	0	0	50	0
9	0	0	0	0	0	0	0	0	0	50

Figure of merit = 500

Table 4.2 Recognition rates for the testing set on MATLAB

	Input words for recognition									
Output	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	48	0	0	0	0	0	0	0
3	0	0	0	50	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	0	0	50	0	0	0	0
6	0	0	2	0	0	0	50	0	0	0
7	0	0	0	0	0	0	0	50	0	0
8	0	0	0	0	0	0	0	0	50	0
9	0	0	0	0	0	0	0	0	0	50

Figure of merit = 498

4.2 DSP Implementation

This speech recognition system was next implemented on a development/demo board based on the ADSP-2181 processor which is manufactured by Analog Devices. This is a 33 MIPS processor with 32 k of 16 bit words of memory and two serial ports (0 and 1) [21]. The speech utterances are taken in through a microphone, which is connected to a stereo programmable codec. The codec samples the speech at 11,025 Hz and transfers the samples to the ADSP-2181 via serial port 0 [22]. For the DSP implementation, only the recognition phase was implemented. The training phase was performed on a PC, and the generated parameters were transferred to the DSP.

4.2.1 DSP Issues

One of the checks which needs to be made is if there is enough memory to hold all the parameters for a 7 state 4 mixture density HMM. For each model, we need to store the state transition matrix. Since we are using a left-to-right model, the state transition matrix is a lower triangular matrix of the following form

$$A = \begin{bmatrix} a(1|1) & 0 & 0 & 0 & 0 & 0 & 0 \\ a(2|1) & a(2|2) & 0 & 0 & 0 & 0 & 0 \\ 0 & a(3|2) & a(3|3) & 0 & 0 & 0 & 0 \\ 0 & 0 & a(4|3) & a(4|4) & 0 & 0 & 0 \\ 0 & 0 & 0 & a(5|4) & a(5|5) & 0 & 0 \\ 0 & 0 & 0 & 0 & a(6|5) & a(6|6) & 0 \\ 0 & 0 & 0 & 0 & 0 & a(7|6) & 1 \end{bmatrix} \tag{4.4}$$

Here we see that only 12 state transition probabilities need to be stored to represent the entire state transition matrix. In addition, the parameters that make up the observation pdf

for each state need to be stored. Since each feature vector consists of 24 elements (12 cepstral coefficients and 12 delta cepstral coefficients), the mean vectors and the standard deviation vectors need to have 24 elements for each mixture component. Thus for each state, there need to be 96 (24 elements $\times$ 4 mixture densities) elements to represent all the mean vectors and 96 elements to represent all the standard deviation vectors. In addition, four mixture weights need to be stored for each state. Thus, for each model, there need to be 672 (96 $\times$ 7) elements for all the mean vectors, 672 elements for all the standard deviation vectors, and 28 elements for all the mixture weights to represent all the observation pdf's.

Each model would need 12 elements for the state transition matrix, 672 elements for the mean vectors, 672 elements for the standard deviation vectors, and 28 elements for the mixture weights. This gives a grand total of 1384 storage elements per HMM model. To represent each of the ten digits in the vocabulary, 13,840 storage elements are needed. Since the ADSP-2181 has 32 k of 16-bit words of memory, there is plenty of room to store all ten models. To avoid complexity, the training parameters that were found in the previous section for all the words were also used for the DSP implementation.

Recall from Section 2.2 that all the preprocessing, LPC analysis, and cepstral calculations are performed in real time. These parameters are continuously calculated while the samples are being collected. However, since the delta cepstral coefficients require knowledge of future cepstral coefficients, these parameters can not be calculated in real time. The delta cepstral coefficients are calculated only after the end of the string has been detected and all the cepstral coefficients have been calculated.

The maximum time for the speech signal was set at five seconds. This should allow ample time to speak a word in isolation or a string of at least 7 digits. Recall from Section 2.2.1 that each time frame is 45 ms in duration and successive time frames overlap each other by 30 ms in duration. Thus for each additional time frame, 15 ms would have passed. For five seconds, there would be a total of 333 time frames. Since each feature vector contains 24 elements, there would have to be 7,992 storage elements set aside in memory to store at most five seconds of speech.

After the cepstral and delta cepstral coefficients, Viterbi decoding was used to compute the likelihood of each model generating the observation sequence. The model producing the best likelihood is chosen as the recognized word.

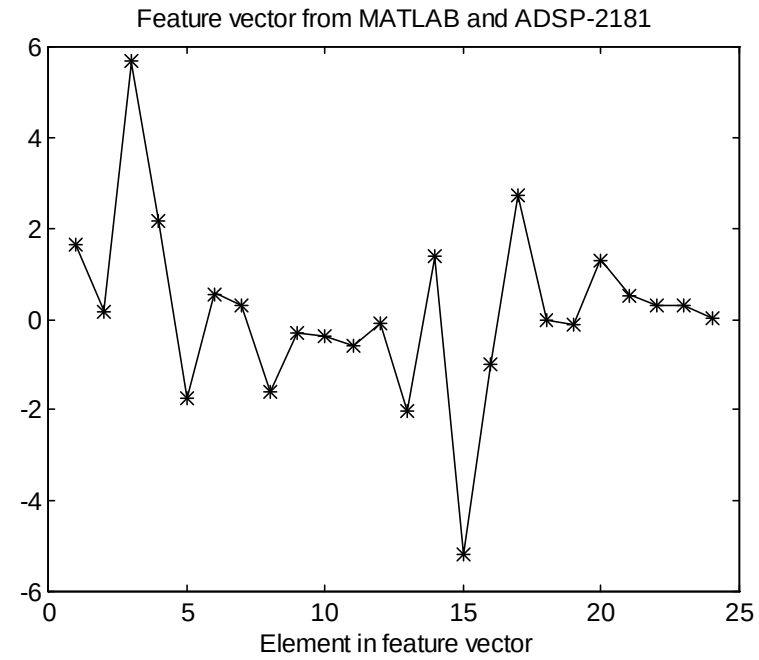
4.2.2 Finite Word Length Effects

The effects of finite word length were examined to see how the quantizations and roundoffs in the algorithm affect the results. Since the ADSP-2181 is a 16-bit fixed point processor, there will inherently be errors introduced because of the finite word length.

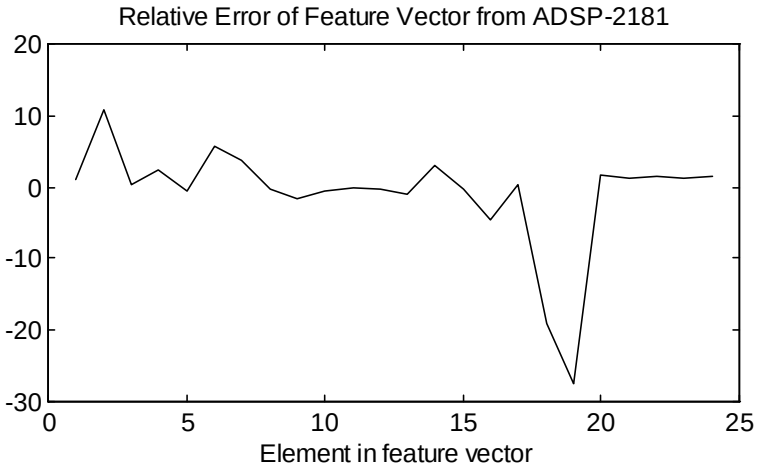
We begin by examining the feature analysis stage in which the cepstral and delta cepstral coefficients are calculated. Samples from a speech utterance of the word '1' were taken in through a PC. The calculations of the feature vectors (cepstral and delta cepstral coefficients) were performed both in MATLAB and on the ADSP-2181 and the results were compared. To ensure that the exact same samples were used, the samples were downloaded ahead of time into the processor. The coefficients generated through MATLAB were assumed to be the true values since there will be a minimal loss of

precision in the calculations. MATLAB performs all the calculations using a 64-bit floating point representation. In the ADSP-2181, a signed fixed-point representation of 5.11 was used. The results for the first time frame are plotted together and shown in Figure 4.7.

Figure 4.7a shows that the computed cepstral and delta cepstral coefficients on the ADSP-2181 (indicated by the continuous curve) do not deviate much from the true cepstral and delta cepstral coefficients (indicated by the ✱). Figure 4.7b gives a plot of the relative error in the calculation. We see that the relative errors of some of these cepstral coefficients appear to be substantial. However, these occur when the actual cepstral coefficient is close to zero. When these errors are compared to the maximum allowed value of 16, the errors generated are not very significant.



(a)

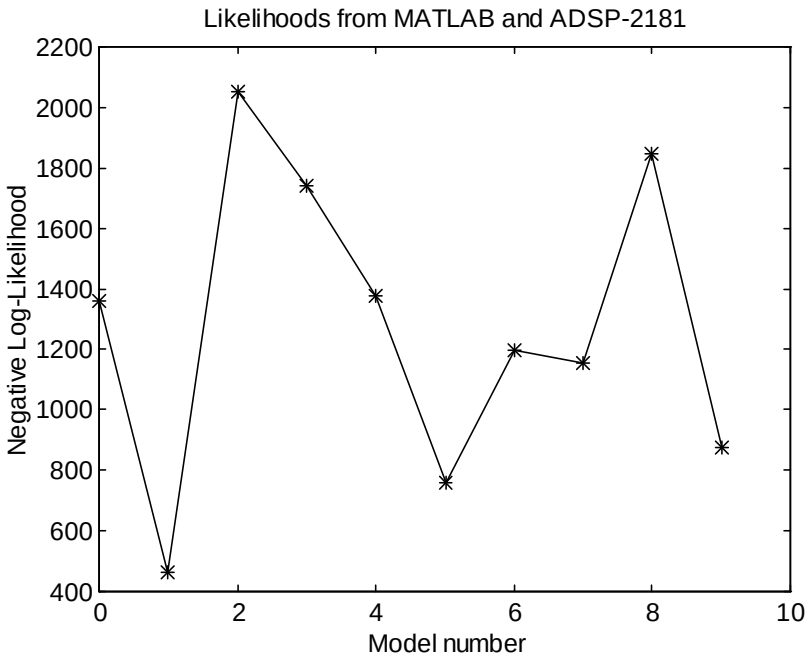


(b)

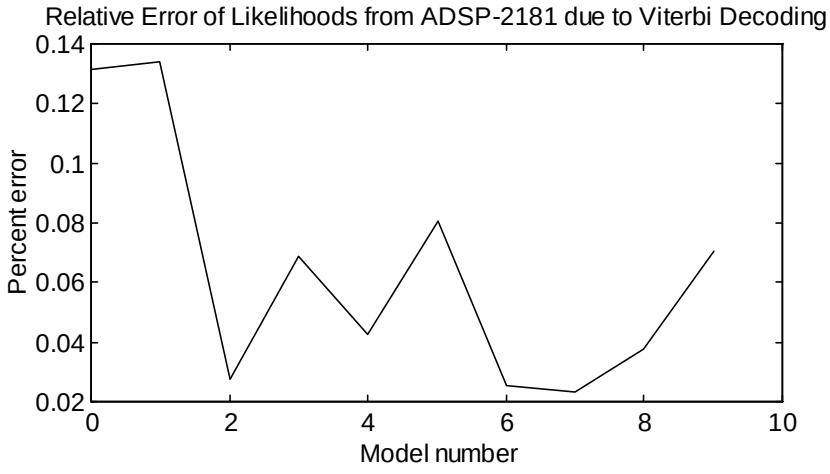
Figure 4.7 Finite word length effect on feature vector
(a) Actual feature vector: (✳) MATLAB, (—) ADSP-2181
(b) Percent error of feature vector

We next examine only the pattern matching and unit identification block to see how the finite word length calculations affect the likelihoods after Viterbi decoding. The true feature vectors from the previous experiment were used in the calculations of the likelihood from each model in both MATLAB and on the ADSP-2181. Figure 4.8 shows the results from this test. Figure 4.8a shows that the negative-log likelihoods produced from the ADSP-2181 after Viterbi decoding do not deviate much from the true likelihoods. The relative error is shown in Figure 4.8b. From this graph, we see that the relative error in the negative-log likelihood calculations deviates by no more than 0.13%.

The entire speech recognition system was next looked at to give an idea of the total effects of the finite word length calculations. This includes both the effects from the feature analysis and from Viterbi decoding. The same samples of the word '1' were used in this experiment. The true likelihoods and the likelihoods produced from the ADSP-2181 are then compared. The results are shown in Figure 4.9.

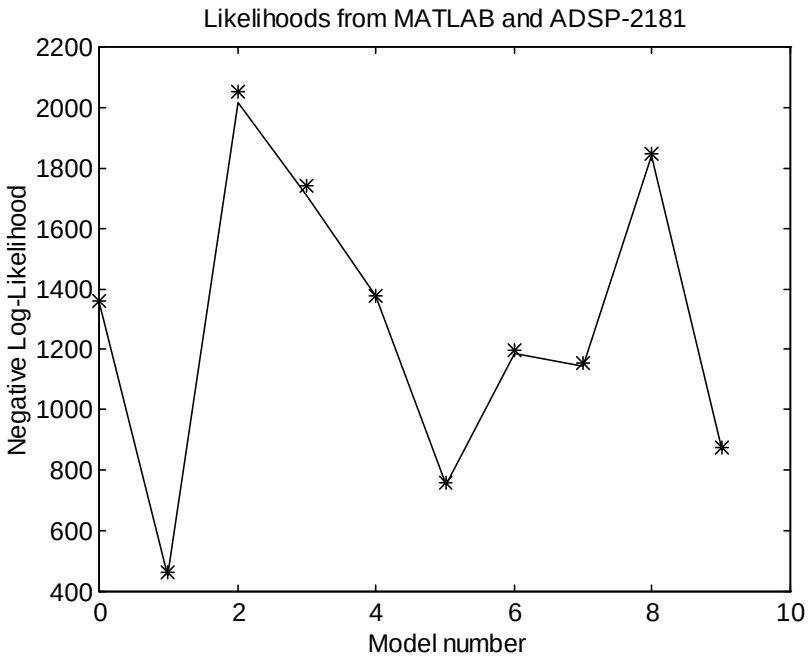


(a)

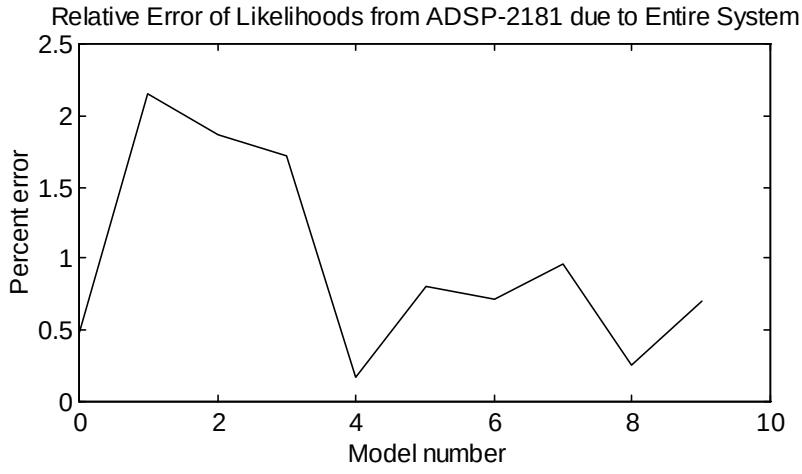


(b)

Figure 4.8 Finite word length effect of Viterbi decoding
(a) Likelihood: (✳) MATLAB, (—) ADSP-2181
(b) Relative error of likelihood



(a)



(b)

Figure 4.9 Finite word length effect of Speech Recognition System
(a) Likelihood: (✳) MATLAB, (—) ADSP-2181
(b) Relative error of likelihood

Figure 4.9a shows that there is a slight noticeable difference in the negative-log likelihoods produced by the ADSP-2181 from the true likelihoods. Again, the relative error of the likelihood is shown in Figure 4.9b. From this graph, we see that the relative error in the negative-log likelihood calculations increases to a maximum of about 2.2%.

For this particular case, the best HMM is the one representing the word '1'. The effects due to finite word length did not hinder making the correct decision. However, it is possible for the two best models to yield very similar likelihoods. The correct model may truly yield the better likelihood. However, the effects due to finite word length can cause the incorrect model to appear to be the better model. In this case, an error would occur due to the finite word length.

4.2.3 Results

To test the recognition system, we first used the training utterances and the testing utterances generated for the PC tests. The audio output of the sound card was connected directly to the codec and the speech samples were played.

Table 4.3 gives the results for the training set. Like before, we see that the figure of merit is 500, giving a recognition rate of 100%. Table 4.4 gives the results for the testing set. Here, the figure of merit is 499 yielding a recognition rate of 99.8%. It is interesting to note that the recognition rate actually improved slightly. However, this does not appear to be statistically significant. In addition, the test words that failed previously on the PC actually passed on the ADSP-2181 and vice versa.

Table 4.3 Recognition rates for the training set on ADSP-2181

	Input words for recognition									
Output	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	50	0	0	0	0	0	0	0
3	0	0	0	50	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	0	0	50	0	0	0	0
6	0	0	0	0	0	0	50	0	0	0
7	0	0	0	0	0	0	0	50	0	0
8	0	0	0	0	0	0	0	0	50	0
9	0	0	0	0	0	0	0	0	0	50

Figure of merit = 500

Table 4.4 Recognition rates for the testing set on ADSP-2181

	Input words for recognition									
Output	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	50	0	0	0	0	0	0	0
3	0	0	0	50	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	0	0	50	0	0	0	1
6	0	0	0	0	0	0	50	0	0	0
7	0	0	0	0	0	0	0	50	0	0
8	0	0	0	0	0	0	0	0	50	0
9	0	0	0	0	0	0	0	0	0	49

Figure of merit = 499

To conclude the testing on the ADSP-2181, we want to see how the recognition system performs on a different source. To do this, a microphone was connected directly to the codec and the individual digits were spoken through the microphone. Of each digit, 50 utterances were spoken into the microphone and the recognition results tabulated as before. The results are given in Table 4.5. The figure of merit is 495, yielding a recognition rate of 99.0%. We see that the recognition rate decreases slightly, but the system still performs very well. For all practical purposes, we conclude that there is no difference in the performance of the speech recognition system on isolated words, whether implemented on the PC or the ADSP-2181.

Table 4.5 Recognition rates for the testing set on ADSP-2181 using microphone

	Input words for recognition									
Output	0	1	2	3	4	5	6	7	8	9
0	50	0	0	0	0	0	0	0	0	0
1	0	50	0	0	0	0	0	0	0	0
2	0	0	50	0	0	0	0	0	0	0
3	0	0	0	50	0	0	0	0	0	0
4	0	0	0	0	50	0	0	0	0	0
5	0	0	0	0	0	48	0	0	0	1
6	0	0	0	0	0	0	50	0	2	0
7	0	0	0	0	0	0	0	50	0	0
8	0	0	0	0	0	0	0	0	48	0
9	0	0	0	0	0	2	0	0	0	49

Figure of merit = 495

This concludes the implementation on isolated words. Next, we will focus on connected word recognition, which is the main topic of this thesis.

5. Connected Digit Recognition

We now turn our attention to the connected digit recognition algorithm used in the implementation of our system to recognize connected strings of digits from a continuous speech signal.

The main purpose of this research is to be able to recognize the individual words in a small vocabulary, from a continuous speech utterance. For our case, the words are simply the ten digits of the English language. Since continuous speech has no distinct boundaries between words, the main difficulty associated with connected word recognition is locating the boundaries of the individual words within the string. In addition, coarticulation between adjacent words can degrade performance and make the task of determining the boundaries more difficult. Once the boundaries of each word are determined, the problem at hand becomes just an isolated word recognizer in which each word is chosen according to the model that yields the maximum likelihood. The problem of recognizing strings of digits becomes even more difficult when the number of words within the string is unknown. Thus, determining the boundaries of each word is very important for the successful recognition of the connected string.

Another issue regarding continuous speech is the number of words present. Obviously, the algorithm for CSR will be easier and less complex if the number of words in the string is known in advance. However this is usually not the case in practice.

This chapter first describes two algorithms for continuous speech recognition. The chapter then concludes with an analysis of the connected string waveform to determine word boundaries within the string.

5.1 Level Building Algorithm

The first approach in solving the connected-speech problem is the level building (LB) algorithm [1][9][23]. The LB algorithm used with Viterbi decoding attempts to find the sequence of words $\{W_1, W_2, \ldots, W_P\}$ which maximizes the joint probability of the observation sequence and the state sequence. Using the assumption that there are approximately L words in the test utterance, the entire speech utterance is partitioned into L "levels" where each level represents a word in the test utterance. An attempt is made to "fit" isolated word reference strings in each of these levels by using the Viterbi method to determine the word sequence. For this case, we want L to represent the maximum number of words in the test utterance. This approach can be viewed as a large search grid as illustrated in Figure 5.1. Dynamic programming techniques are used to find the best path through this search grid.

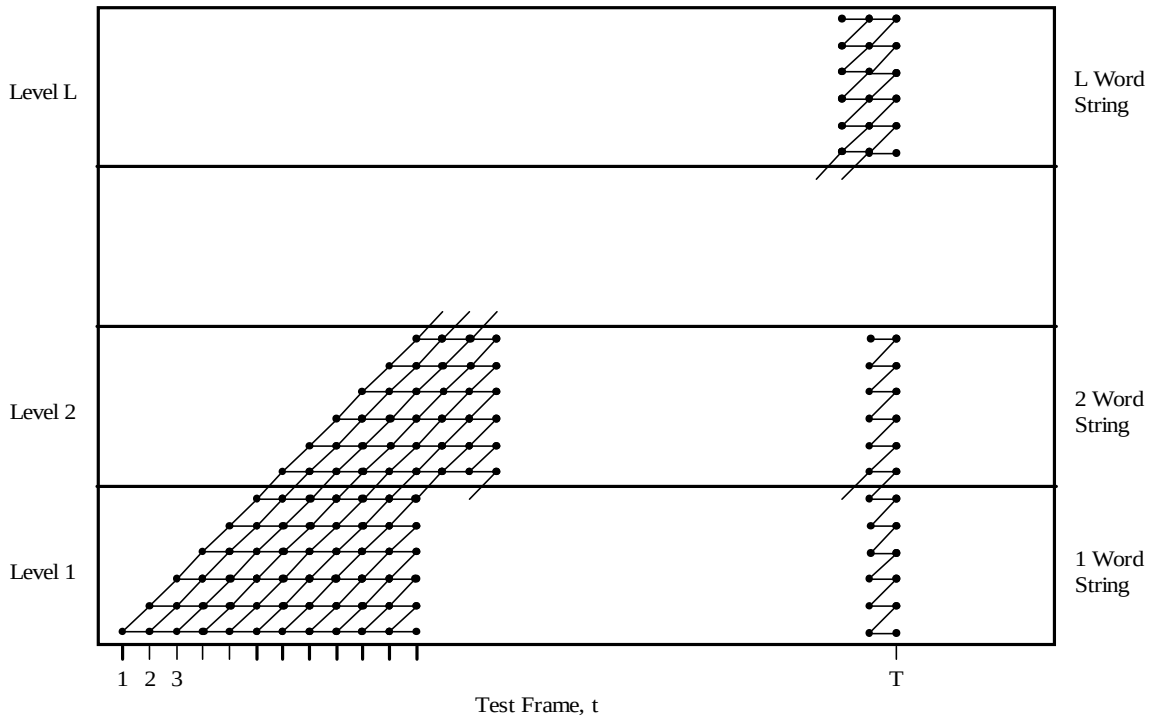


Figure 5.1 Search grid for Level-Building algorithm

Before we begin the search, the partial joint likelihood of occurrence at sequence time t of the optimal partial state sequence $\{i_1^*, i_2^*, \cdots, i_t^*\}$ and the partial observation sequence $\{y(1), y(2), \cdots, y(t)\}$ is given by $D_{\min}(t, i_t)$. At each level, the algorithm attempts to find the best model, $\mathcal{M}_q$, corresponding to the best word q in the vocabulary. At level 1, the best model is found over all the frames of the test utterance using Viterbi decoding. This is given by the following algorithm [23].

1. Initialization

$$D_{\min}(1,1) = -\log\left[b^q\left(y(1)|1\right)\right] \tag{5.1a}$$

For $2 \le i \le S$

$$D_{\min}(1,i) = 0 \tag{5.1b}$$

where $b^q\left(y(t)|i\right)$ is the likelihood using $\mathcal{M}_q$ (see (3.10)).

2. Recursion

For $2 \le t \le T$, $\quad 1 \le i_t \le S$

$$D_{\min}(t,i_t) = \min_{1 \le i_{t-1} \le S} \left\{D_{\min}(t-1,i_{t-1}) - \log\left[a^q\left(i_t|i_{t-1}\right)\right]\right\} - \log\left[b^q\left(y(t)|i_t\right)\right] \tag{5.2}$$

where $a^q\left(i_t|i_{t-1}\right)$ is the state transition probability given in (3.1) for $\mathcal{M}_q$.

3. Termination

For $1 \le t \le T$

$$P(l,t,q) = D_{\min}(t,S) \tag{5.3}$$

$$B(l,t,q) = 0 \tag{5.4}$$

P is an array that holds the output of the level (the joint negative-log likelihood) at each time frame. *B* is an array that contains the time frame of the most likely place where the previous level ended. Since we are analyzing the first time frame, this array is initialized to zero. The above algorithm is repeated for all the models in the system. After cycling through all the models, the algorithm uses the *P* array to find the best model at each time frame. The following arrays are then formed which are the pertinent information for the first level.

$$\hat{P}(l,t) = \min_{q}[P(l,t,q)] \tag{5.5a}$$

$$\hat{B}(l,t) = B\left[l,t,\operatorname{argmin}_{q}(P(l,t,q))\right] \tag{5.5b}$$

$$\hat{W}(l,t) = \operatorname{argmin}_{q}(P(l,t,q)) \tag{5.5c}$$

$\hat{P}(l,t)$ is the level output best likelihood at (l,t). $\hat{B}(l,t)$ contains the time frame of the most likely place where the previous level ended. $\hat{W}(l,t)$ indicates the best word at (l,t).

The transition to level 2 and all higher levels is straightforward since these levels pick up from the previous outputs. The computations are similar to level 1, and the only major difference is in the initialization procedure. The algorithm for the higher levels is shown below

1. Initialization

$$D_{\min}(1,1) = \infty \tag{5.6a}$$

For $2 \le t \le T$

$$\begin{aligned} D_{\min}(t,1) = \min\left\{\hat{P}(l-1.t-1), D_{\min}(t-1,1) - \log\left[a^{q}(1|1)\right]\right\} \\ - \log\left[b^{q}(y(t)|1)\right] \end{aligned} \tag{5.6b}$$

$$\alpha(t,1) = \begin{cases} t-1, \text{ if } \hat{P}(l-1,t-1) < D_{\min}(t-1,1) - \log\left[a^{q}(1|1)\right] \\ \alpha(t-1,1), \text{ otherwise} \end{cases} \tag{5.7}$$

2. Recursion

For $2 \le t \le T$, $\quad 1 \le i_t \le S$

$$D_{\min}(t,i_t) = \min_{1 \le i_{t-1} \le S}\left\{D_{\min}(t-1,i_{t-1}) - \log\left[a^{q}(i_t|i_{t-1})\right]\right\} - \log\left[b^{q}(y(t)|i_t)\right] \tag{5.8}$$

$$\alpha(t, j) = \alpha\left(t-1, \underset{1 \le i \le S}{\operatorname{argmin}} \left(D_{\min}(t-1, i) - \log\left[\alpha^{q}(i|j)\right] \right) \right) \tag{5.9}$$

3. Termination

   For $1 \le t \le T$

$$P(l, t, q) = D_{\min}(t, S) \tag{5.10}$$

$$B(l, t, q) = \alpha(t, S) \tag{5.11}$$

Equation (5.6a) sets the joint likelihood to zero (infinity for negative-log likelihood) for the first time frame of State 1 while (5.6b) chooses the most appropriate place for the level to start. Equation (5.7) is the initial backpointer array that records the frame where the previous model in the previous level ended. This backpointer array is updated in the recursion step from (5.9).

For each level, the above algorithm is repeated for all the models in the system. After cycling through all the models, we use (5.5) to build up the $\hat{P}$, $\hat{B}$, and $\hat{W}$ matrices for each successive level. The algorithm repeats until the final level, *L*, is reached. The final column in $\hat{P}$ contains the best string likelihood of length *l*. For example, the "best string" of length *L* has the joint likelihood $\hat{P}(L, T)$. The word string can be found by backtracking using the $\hat{B}$ array to yield the words in the string from the $\hat{W}$ array. The "best string" has length L_{best} and is the level that yielded the best joint likelihood. This is computed by

$$L_{best} = \underset{1 \le l \le L}{\operatorname{argmin}} \left\{ \hat{P}(l, T) \right\} \tag{5.12}$$

The sequence of words $\{W_1, W_2, \cdots, W_{L_{best}}\}$ is then found by backtracking. This is found by the algorithm below.

$$t = T \tag{5.13}$$

For $l = L_{best}, L_{best} - 1, \ldots, 1$

$$W_l = \hat{W}(l,t) \tag{5.14}$$

$$t = \hat{B}(l,t) \tag{5.15}$$

The variable t contains the last time frame for the current level. This variable is updated from the $\hat{B}$ array in (5.15).

5.2 One-Stage Algorithm

The one-stage (OS) algorithm for connected-word recognition, along with Viterbi decoding, finds the optimal path in "one stage" of computation instead of building a series of "levels" as the LB algorithm describes [1]. Knowing this, a “stage” is then one of the words making up the sequence of words. This is similar to a “level” in the LB algorithm. The OS algorithm is best visualized with the three-dimensional grid shown in Figure 5.2. The observations are laid out along the abscissa and the states along the ordinate (similar to the search grid of Figure 3.2). The third axis (q axis) corresponds to the word index.

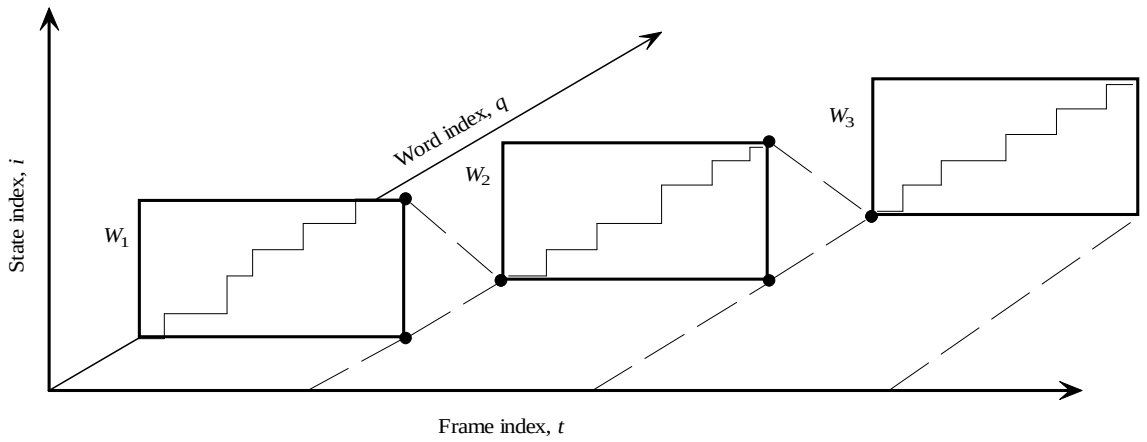


Figure 5.2 Search grid illustrating the OS algorithm

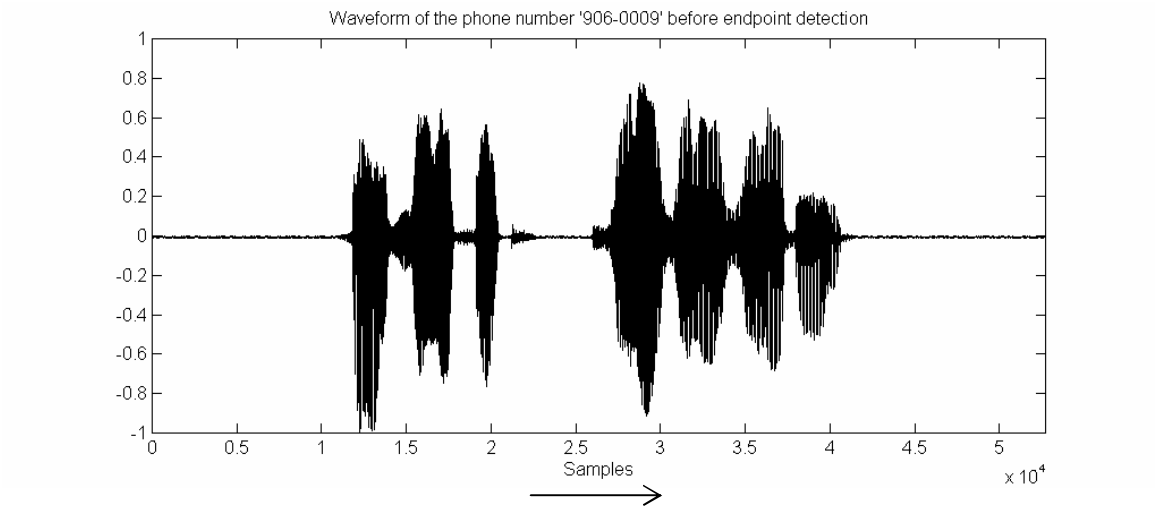
Each individual stage represents one of the words in the sequence of words. Therefore, each stage is represented by a Viterbi search grids (Figure 3.2). Viterbi decoding as, described in Section 3.3, is performed for all the reference models in the system. Transitions along the q dimensions are only made under very strict constraints. For the current reference model, say q, it is only possible to exit the search grid when the path reaches the upper boundary of that particular grid. Only after the path exits the current search grid is it possible to continue at the bottom of another grid (or possibly the same search grid to allow the same words to occur consecutively). With these constraints, there are obviously two sets of rules for the OS algorithm: *within-model* transition rules, which govern the path search while the evaluation is internal to one of the word search grids, and *between-model* transition rules, which govern the path search at the top of the boundaries. These transition rules need to be defined. Before we can define them, we

need to do a full analysis on the entire speech utterance to try to locate the boundaries between consecutive words. The boundary analysis is discussed next.

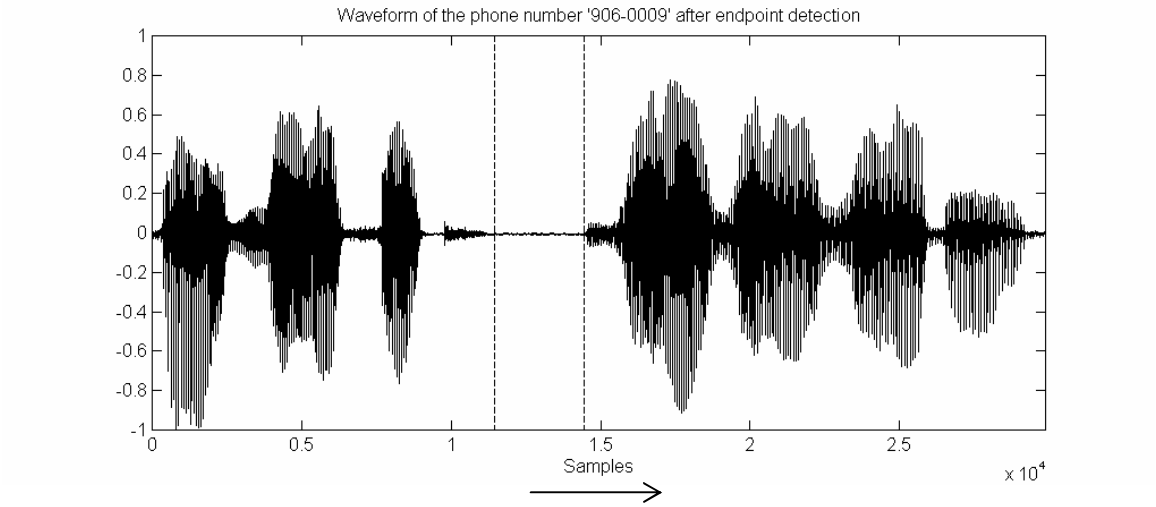
5.2.1 Boundary Analysis

To determine the boundaries of a word, we will incorporate two methods to find the boundaries of all the words. The first of the two is quite simple and is a measure of the silent times within the connected string of digits. Recall from the block diagram of the speech recognition system that the energy in the frame and the number of zero crossings are used as an indication to determine the endpoints of an isolated word. When both of these are below a preset threshold for a length of time, we consider this to be the end of the word. The idea here is to incorporate a slight modification to account for slight pauses within the continuous speech. When these pauses occur, they will almost always occur between words and thus give a good indication of where some of the boundaries between the words occur.

Figure 5.3 illustrates the use of detecting pauses within a connected string of digits. For this example, if a pause of 150 ms or more occurs, the algorithm detects this and notes the start and the end of this pause. Figure 5.3a shows the waveform of the phone number '906-0009'. Here we see that there is a pause between the digits '6' and second '0'. After passing this waveform through the modified endpoint detector, we see that the algorithm detected two separate strings within the entire speech waveform (Figure 5.3b). The dashed lines indicate where the beginning and the ending of each string of digits occurs.



(a)



(b)

Figure 5.3 Waveform of the phone number '906-0009'
(a) Before endpoint detection
(b) After endpoint detection

The second method is a little more complicated. Recall that we use a left-to-right HMM sequence for all of the models. Thus for an *S* state HMM, each word will progress through the different models starting in State 1 and ultimately ending in State *S*. To find the boundary of the word, only the partial joint likelihood of the final state needs to be examined. For the model that matches the word spoken, this likelihood will not change by too much as the word is analyzed from one time frame to the next time frame until the end of the word. However, for a model that does not match the word, the change in the partial joint likelihood of the final state from one time frame to the next will be much greater. Since the word must progress from State 1 to State *S*, the last state can be analyzed while performing the Viterbi decoding to determine the endpoint of the current word.

To illustrate this analysis, we again examine the phone number '906-0009'. From Figure 5.3b, the waveform of this signal, after the modified endpoint detector, shows that there are no definite boundaries among the first three digits and among the last four digits. Clearly more analysis must be done to recognize this string correctly.

An analysis on the first three digits ('906') was performed. A 7 state HMM is implemented to model each digit. The partial joint likelihood of State 7 at each time was examined for each model while performing Viterbi decoding. Figure 5.4 shows a plot of this likelihood for the HMMs modeling the words '9', '0', and '3'. By inspection, we see that for the HMM representing the word '9', the likelihood begins fairly flat when the word '9' is being spoken. After the end of the word, the likelihood tends to rise sharply. For the HMM representing the word '0', the likelihood increases almost linearly before beginning

to flatten out when the word '0' is being spoken. Again, after the end of the word, the likelihood tends to rise sharply. For the HMM representing the word '3', the partial joint likelihood is plotted as a comparison. The plot shows that there are some time frames where the change from one to the next is not large. However, this likelihood is worse than the likelihoods representing models of words that are in the string. Figure 5.5 shows the change in likelihood for the three models discussed above. From this graph, we can see when the change in likelihood decreases. By placing thresholds on the change in likelihood, the boundaries of the words in the sequence can be found.

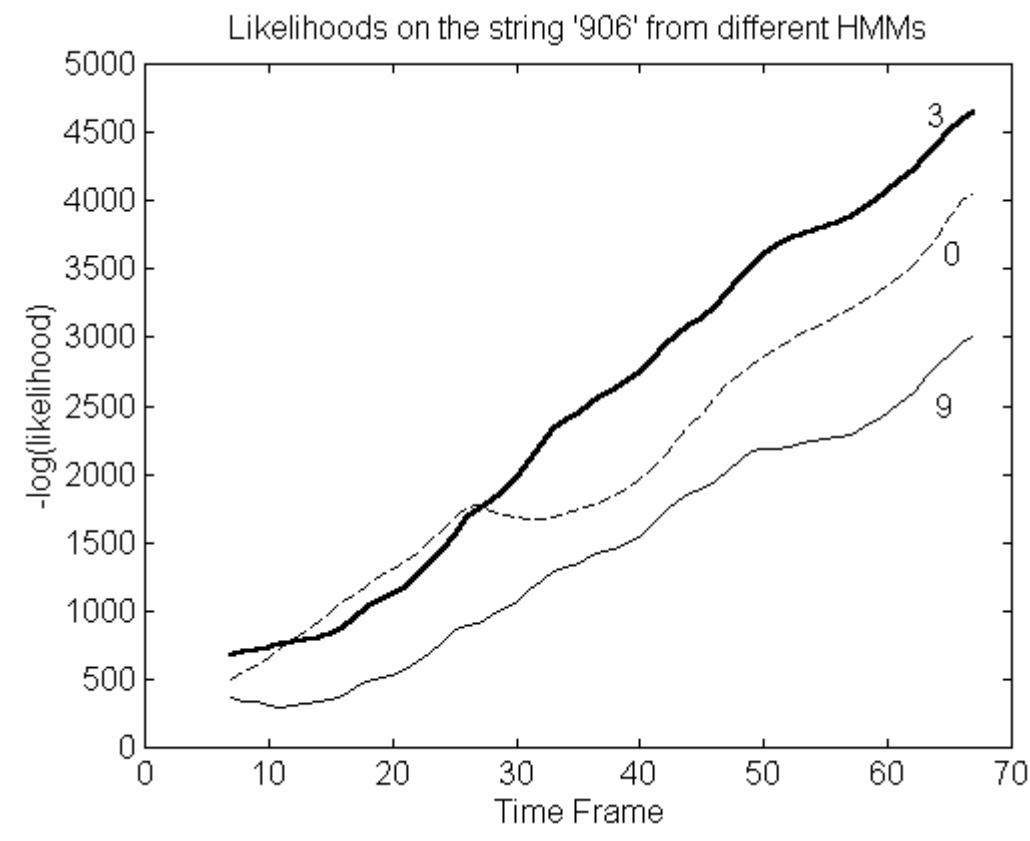


Figure 5.4 Analysis on the connected string '906'

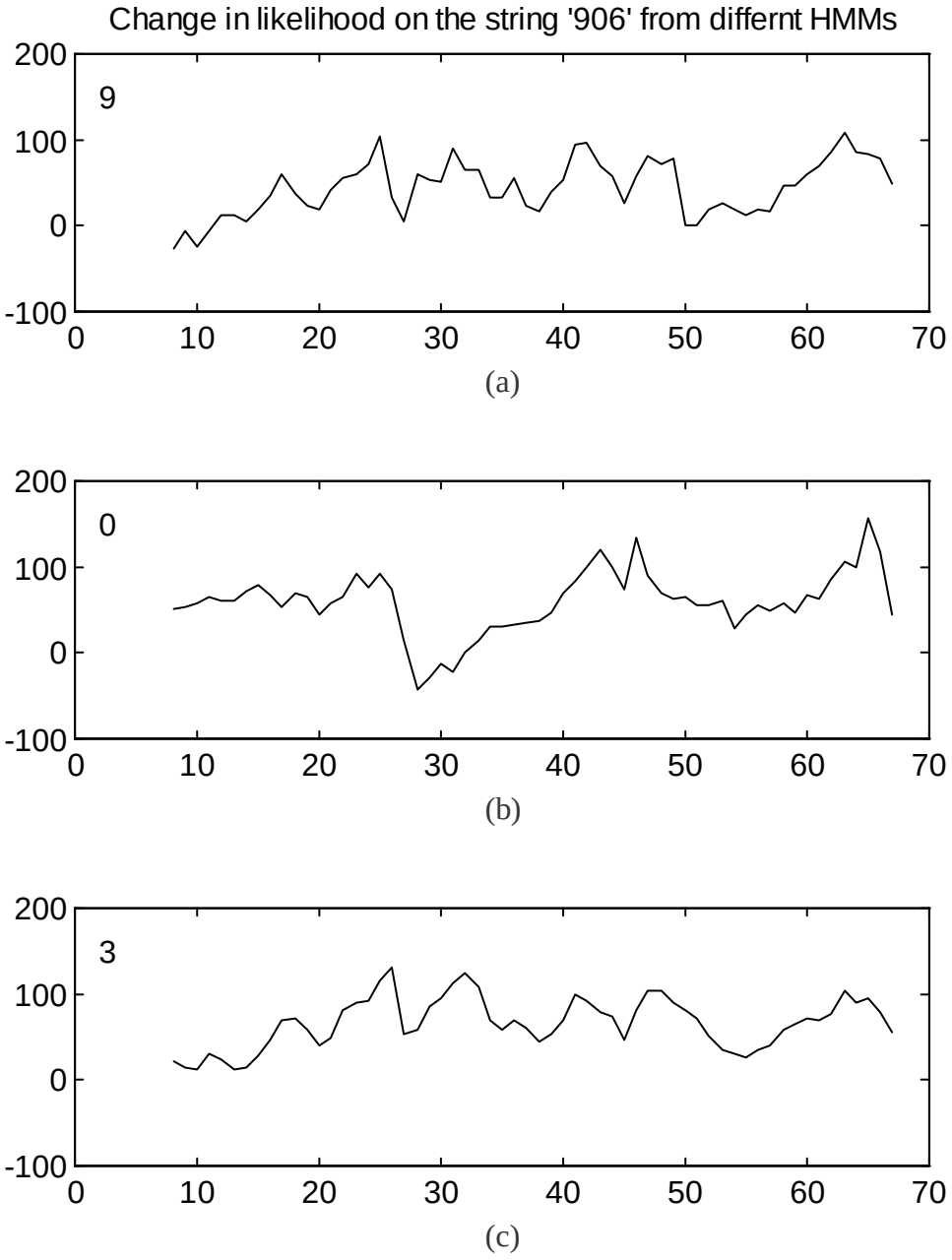


Figure 5.5 Change in likelihood for the string ‘906’
(a) with the HMM for the digit ‘9’
(b) with the HMM for the digit ‘0’
(c) with the HMM for the digit ‘3’

To analyze the final state to determine the end of the word, we need to determine the change in likelihood from one time frame to the next time frame of each model while performing the Viterbi decoding. To improve recognition, we first delay our analysis for a preset time, t_d, determined from training. We next note when the change in likelihood drops below a threshold, T_p. After the latter occurs, we next detect when the upward change in likelihood is greater than this threshold T_p for two consecutive time frames. Upon this occurance, we determine that the end of the word has occurred for the current model. The entire algorithm is summarized below.

1. Allow t_d time frames to pass before comparing the change in likelihood in the final state.
2. Note when the change in likelihood drops below a preset threshold T_p.
3. Note when the change in likelihood rises above T_p for two consecutive frames.

To determine t_d, the training data is inspected with the trained model for each word. To choose the minimum delay time, we choose the minimum time it takes for any of the analyzed state sequence estimates derived from the training data to reach the final state. This delay increases recognition since we do not want to prematurely end a word when it might still be in an early state. To determine T_p, we examine the maximum change in likelihood from all the training data in the last state. We then increase this change for T_p slightly to give a kind of safety margin.

After performing Viterbi decoding along with this boundary check for each model, the likelihood along with the number of time frames for that model is the output. Since the time duration may differ for the different models, it is important to weight the likelihood accordingly since the longer a word is spoken, the less likely a model would have produced that word. The weighting is necessary so that a model associated with a short word in question is not so easily mistaken for the actually correct model associated with a longer time. To weight the likelihood, we divide the likelihood by the number of time frames for that model. All of these weighted likelihoods are compared, and the one that yields the smallest weighted likelihood is deemed the correct word.

Once the word is decided, the length of that word is noted. The next word picks up where the first word has ended and the entire process is repeated. This is continued until the end of the string.

To relate this to the one-stage algorithm, the *within-model* transition rules are such that the change in likelihood for the final state does not satisfy the boundary check mentioned above. The *between-model* transition rules are those that satisfy the boundary check.

Figure 5.6 illustrates how this boundary analysis is applied to the same phone number of '906-0009'. Again, the dashed lines represent the starting points and the ending points of the individual strings within this speech utterance. The double-dashed lines represent where the algorithm detected the beginning of the next word after performing all the boundary checks. It is clear that the algorithm did a fairly good job of detecting these boundaries and the string was correctly recognized as '906-0009'.

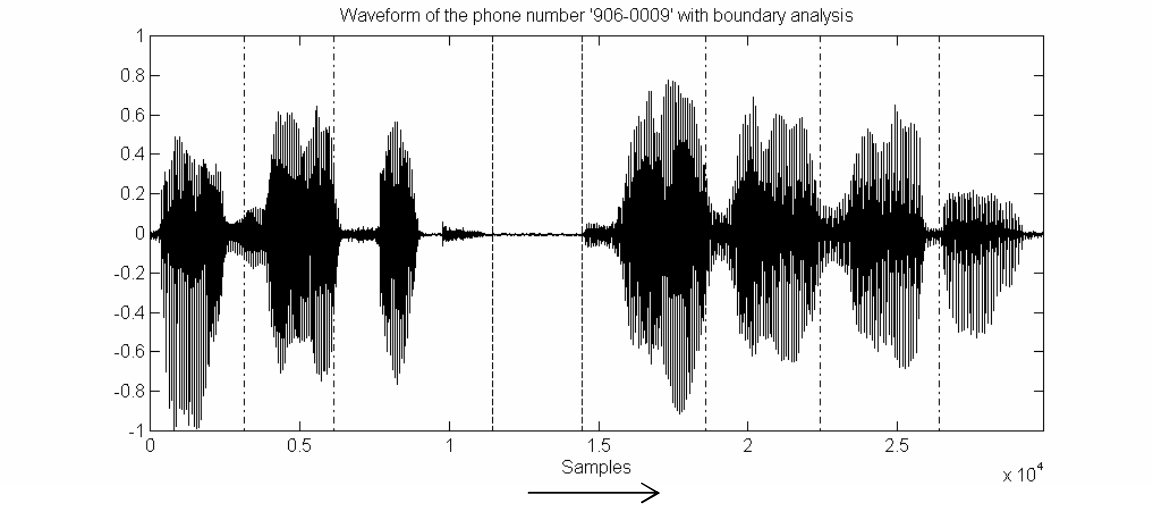


Figure 5.6 Waveform of the phone number '906-0009' with boundary analysis

5.3 Left-Right-Left One Stage

The algorithm described in the previous section is in reality a one way approach for determining the sequence of words making up the connected string. Since this approach starts at the first time frame and ultimately ends up at the last time frame, we will call this the left-right OS approach (LR-OS). While the LR-OS algorithm works fairly well, the performance of the procedure leaves room for improvement. Since we aim at obtaining very high recognition rates, a second procedure is proposed called the left-right-left one stage approach (LRL-OS).

In the LRL-OS approach, two passes are actually performed on the connected string. The first pass is the same as the LR-OS approach. In addition, we also perform a right-left pass. The right-left pass is identical to the left-right pass, except that the analysis

is started on the last frame and ultimately ends on the first frame. However, to perform this reverse analysis, every word in the vocabulary must also be modeled in a reverse sense, i.e. using a right-to-left HMM.

After both passes have been analyzed, there are two possible candidate strings for the correct string. The chosen string is the one that produces the largest overall likelihood over all the digits. The overall likelihood is the product of all the individual word likelihoods. The idea here is that if one pass should give an incorrect string while the other one finds the correct string, the one producing the correct string will have the higher likelihood and will be chosen over the other.

While the LRL-OS algorithm should increase recognition over the LR-OS algorithm, there is a price to pay. The computational requirements effectively double and the overall complexity increases since there are many more things to manage.

5.4 Other Considerations

5.4.1 Durational Constraints

In general, the HMM matches to the portions of the connected test string are not constrained in time [23]. Thus it is possible to match the HMM to a large or small part of the test string, even if it is unlikely that the word that the model represents would be as long or as short in duration as the match. An example of this would be matching the HMM representing the word “technical” against the actual test word of “the.” For a connected word recognizer, this can turn out to be a problem that significantly degrades the performance of the recognizer. To alleviate some of the problems caused by the time

duration phenomenon, durational constraints are often taken into account for HMM modeling.

A simple Gaussian durational model can be used for each word in the vocabulary. The Gaussian durational model is characterized by the probability density function for the duration D for the qth word given by

$$P_q(D) = \frac{1}{\sqrt{2\pi}\cdot\sigma_q} e^{-\frac{(D-\overline{D}_q)^2}{2\sigma_q^2}} \tag{5.16}$$

where $\overline{D}_q$ is the mean and σ_q is the standard deviation of the duration for the qth word. Both the mean and standard deviation are estimated by the sample mean and sample standard deviation taken from the training data used to model the word. We next discuss how the duration constraints can be used in both the LB algorithm and the OS algorithm.

The incorporation of the durational constraints in the LB algorithm is as follows. First, it is necessary to find the word duration, d_t at the end of each level. This duration information is recovered from the backtracking procedure and is given by

$$d_t = t - B(l,t,q) \;\; \forall t \tag{5.17}$$

The cumulative probability at the end of each level is modified to produce a weighted probability array of

$$\tilde{P}(l,t,q) = P(l,t,q)\cdot\left[P_q(d_t)\right]^{\gamma} \tag{5.18}$$

where γ is a weighting parameter. This weighted probability array, $\tilde{P}(l,t,q)$, is then used in place of the unweighted probability array, $P(l,t,q)$, in (5.5), to construct the $\hat{P}$, $\hat{B}$, and $\hat{W}$ arrays.

The incorporation of the duration constraints was performed on the LRL-OS algorithm. After both strings have been found, the individual likelihoods are weighted with the durational constraints. Afterwards, the overall weighted likelihood is found by multiplying the individual weighted likelihoods. The best string is found by choosing the one yielding the best overall weighted likelihood.

5.4.2 Syntactic Considerations

Up until now, we have made the assumption that any word in the vocabulary can follow any other word in the vocabulary with equal likelihood. While this may be a valid assumption for a connected digit recognizer, this is rarely the case for a general purpose connected word recognizer.

The use of syntactic constraints actually helps the continuous speech recognition problem in two ways. First, it reduces the number of HMMs to perform Viterbi decoding on since highly unlikely models are ignored. This in turn reduces the number of computations the system must make. For example, consider a system whose vocabulary consists of nouns, verbs, and articles ('a', 'an', and 'the'), and sentences are to be recognized. A word following an article must always be a noun. Thus the HMMs representing the articles and the verbs can be ignored if the previous word is an article

since it is very unlikely that these words (articles or verbs) will follow the previous word. In addition, the use of syntactic constraints helps improve recognition rates. By just concentrating on those models that are likely to represent the next word, the decision on the next word does not have to come from all the HMMs in the system. Thus the recognition rates increase since an error is more likely to occur if the decision is based on a larger number of models.

So far we have mentioned how syntactic information can be used to aid in a connected word recognition system. By using syntactic constraints in the system, there are certain paths taken in performing Viterbi decoding on the HMMs. However, it is possible that a word can be recognized incorrectly at the beginning and the path taken (through the syntactic rules) is entirely incorrect. In this case, the entire recognized string will be completely incorrect. From past research, this type of error can be made very small [1][23].

The use of syntactic information to recognize a string of digits in a phone number can be used since there are usually some patterns in the digits, such as the three digits of an area code or the first three digits in a seven-digit number. However, as with all systems, there are many exceptions and care must be taken such that some words are not completely eliminated.

6. Implementation on Connected Digits

The connected word recognition algorithms were implemented for the ten digits and their performances were analyzed. Both the level building algorithm and the one-stage algorithm were implemented using MATLAB. The one-stage algorithm was also implemented on the ADSP-2181.

6.1 Implementation Results

6.1.1 PC Implementation

First, we look at the level-building algorithm and present the results. We then look at the one-stage algorithm with various modifications made to it. Before we get into any of the actual results, we first discuss the different types of errors that can occur.

There are basically two types of errors to quantify: string errors and word errors. A string error occurs when the recognized string does not match up exactly with the intended connected speech. Causes for string errors include insertion of digits, deletion of digits, or an incorrectly recognized word. A word error is a subset of a string error and occurs when an individual word within a connected string of words is not recognized. Thus an incorrectly recognized word or a deletion yields a word error, as well as a string error. For example, for the connected string '12345', it is only possible to have one string error. However, for this string, there can be at most 5 word errors.

The connected-word algorithms were first tested on isolated digits to see how well they perform compared to those presented in Chapter 4. Ten recordings of each digit

were produced as testing data for isolated words, giving a total of 100 recordings. In addition, the connected-string algorithms were tested on string lengths of two, three, and four. To test the connected-string algorithms, 100 random strings of length two, three, and four each were generated and recorded for testing data. All the testing data were used in the connected-string algorithm to evaluate the performance of each.

The LB algorithm was first implemented in MATLAB. The algorithm was discussed in Section 5.1. We make the assumption that all the digits in the connected string will have at least a time duration of 150 ms. This is equivalent to ten time frames. To determine the number of levels to evaluate, the total number of time frames for the connected string is divided by ten and then truncated. In addition, duration constraints were used in the implementation as discussed in Section 5.4.1. The gamma value used in equation (5.18) is three. This value was chosen based on experiments performed by Rabiner [23].

The results of the LB algorithm with the durational constraints are shown below in Table 6.1.

Table 6.1 Recognition rates using the LB algorithm in MATLAB

Number of Digits Per string	Word Recognition Rate	String Recognition Rate
1	100%	100%
2	90.5%	85%
3	88%	73%
4	88%	62%

From the table above, we see that the recognition system using the LB algorithm is perfect for the 100 isolated digits tested. However, performance levels decrease as the number of digits in the string increases. The word recognition rate gradually decreases as the string length increases, but it remains relatively high. For a string length of four, the word recognition rate is at 88%. The string recognition rate decreases much more dramatically as the number of digits increases. This can be expected since any individual word error within the string (word replacement or deletion) or any insertion will cause a string error. For a four-digit string, we see that the string recognition rate is about 62%.

The OS algorithm with various modifications was next implemented in MATLAB. The recognition rates for the LR-OS algorithm is given in Table 6.2.

Table 6.2 Recognition rates using the LR-OS algorithm in MATLAB

Number of Digits Per string	Word Recognition Rate	String Recognition Rate
1	100%	98%
2	93.5%	90%
3	92%	82%
4	90.75%	70%

Like the LB algorithm, we see that the recognition system performs quite well on isolated words using the LR-OS approach. However, when more and more digits are concatenated, the string recognition rate decreases. Again, the word recognition rate gradually decreases and achieves a recognition rate of about 90.75% for a 4-digit string.

The string recognition rate again decreases much more dramatically. For a four-digit string, the string recognition rate is about 70%.

To try to improve the recognition rates on the OS algorithm, we next implemented the LRL-OS algorithm. As mentioned before (Section 5.3), it performs two passes to determine the best string. Again, we used the same test strings as before. The recognition rates using this method are shown below in Table 6.3

Table 6.3 Recognition rates using the LRL-OS algorithm in MATLAB

Number of Digits Per string	Word Recognition Rate	String Recognition Rate
1	100%	100%
2	98%	97%
3	95.33%	89%
4	93.5%	79%

Here we see that the recognition rates for the LRL-OS algorithm have improved over those obtained from the LR-OS algorithm. Again, the recognition rates are very high for isolated words as there were no errors for the 100 isolated words tested. Again, we see that the word recognition rate gradually trails off as the number of digits increases in the string. In addition, the string recognition rate improved from 70% to 79% for the LRL-OS algorithm for a four-digit string.

In the LRL-OS algorithm, the recognized string comes from either the left-right pass or the right-left pass. For every string error that occurred, we looked at the results

from both the left-right pass and the right-left pass and compared the two to see if the other had given the correct string. The results are summarized below in Table 6.4.

Table 6.4 Comparison of errors for the LRL-OS algorithm

Number of Digits per string	Number of string errors (from 100)	Number of times the other string is correct
1	0	--
2	3	1
3	11	5
4	23	8
Total	37	14

From the table above, we see that when an error occurs for the LRL-OS algorithm, the other string estimate is correct about 38% of the time. Obviously, we want to introduce a method for estimating the correct string almost all the time. To facilitate this, several changes were made to the original LRL algorithm. For starters, we did not force the next word in a string to start where the previous word ended. Instead we allow two time frames of overlap between individual words. This is valid since there is already a 67% overlap between successive time frames (see Section 2.2.1). In addition, information about the time-duration of the individual words was introduced into the recognition system in order to make a better decision about which string is better (left-right or right-left). With these modifications included, the results obtained with the Modified LRL-OS approach are presented below in Table 6.5.

Table 6.5 Recognition rates using the Modified LRL-OS algorithm in MATLAB

Number of Digits Per string	Word Recognition Rate	String Recognition Rate
1	100%	100%
2	98.5%	98%
3	98.33%	96%
4	98%	91%

Here, we see a drastic improvement in all the recognition rates. Again, the word and string recognition rates for an isolated word are 100% out of the 100 testing words. The word recognition rate decreases slightly as the number of digits in the string increases, to a 98% word recognition rate for a 4-digit string. In addition, the string recognition rate has improved to 91% for a 4-digit string.

Next, we looked at all the errors which occurred to see how well the system performed in choosing the best string between the left-right pass and the right-left pass. This error comparison is shown below in Table 6.6.

Table 6.6 Comparison of errors for the modified LRL-OS algorithm

Number of Digits Per string	Number of string errors (from 100)	Number of times the Other string is correct
1	0	--
2	2	0
3	4	1
4	9	2
Total	15	3

From the table above, we see that for the Modified LRL-OS approach, the other string is correct about 20% of the time for all the errors. From this we see that the Modified LRL-OS approach yielded better results than those obtained before the modifications were incorporated.

For all the methods, we looked at how the errors occurred to discern some type of pattern in the errors. Most of the word insertions were due to the number '6' being inserted in the recognized string. Since the number '6' both starts and ends in the unvoiced fricative /s/, we expect background noise and silent regions to be most likely produced by the HMM representing the number '6'. The reason for this is because unvoiced sounds, like /s/, can be modeled with a random noise generator. In addition, the most common word errors occur where the number '8' is being recognized as a '6', the number '9' is being recognized as a '5', and the number '7' is being recognized as a '9'. In addition, the number '7' frequently resulted in the string '69'.

Figure 6.1 and Figure 6.2 give comparisons among all the connected-string recognition algorithms implemented by showing a plot of the error rates versus the number of digits in the string.

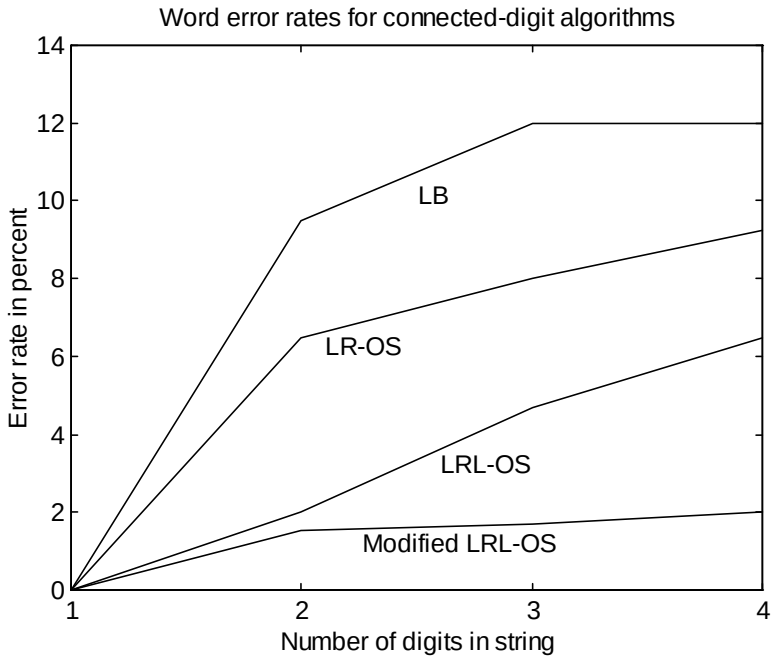


Figure 6.1 Word error rates for the MATLAB implementations

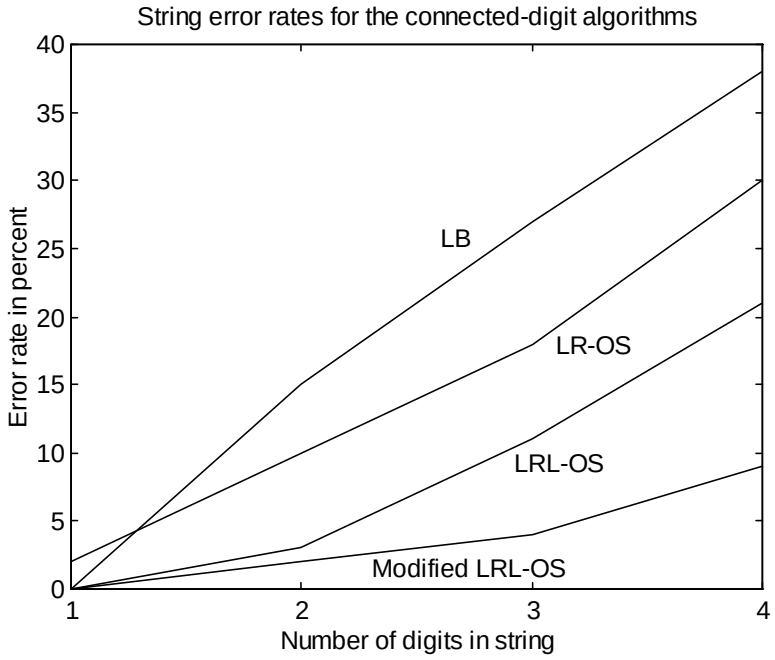


Figure 6.2 String error rates for the MATLAB implementations

6.1.2 DSP Implementation

The LR-OS algorithm was next implemented on the ADSP-2181. Many of the same issues discussed in Section 4.2 also apply to this implementation. The recognition system was tested with the same data as the PC MATLAB implementation by connecting the output of the PC sound card directly to the codec. The results from this testing are presented in Table 6.7.

Table 6.7 Recognition rates using the ADSP-2181 LR-OS implementation

Number of Digits Per string	Word Recognition Rate	String Recognition Rate
1	99%	99%
2	94.5%	92%
3	93%	82%
4	92.75%	79%

From the table above, we see that the results of the LR-OS algorithm on the ADSP-2181 are comparable to the results from the MATLAB implementation, reported in Table 6.2. In fact, the results from the ADSP-2181 are slightly better than those obtained with MATLAB. However, this is probably not statistically significant. For isolated words, the system performs exceptionally well. However, both the string recognition rates and the word recognition rates decrease as the number of digits in the string increases. Figure 6.3 and Figure 6.4 compare error rates for the MATLAB and ADSP-2181 implementations of the LR-OS algorithm.

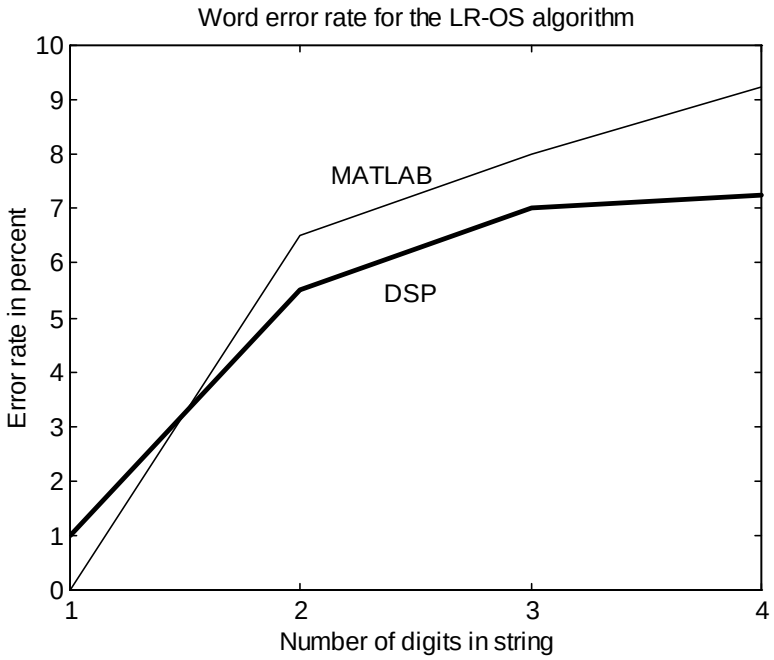


Figure 6.3 Word error rate for the LR-OS method

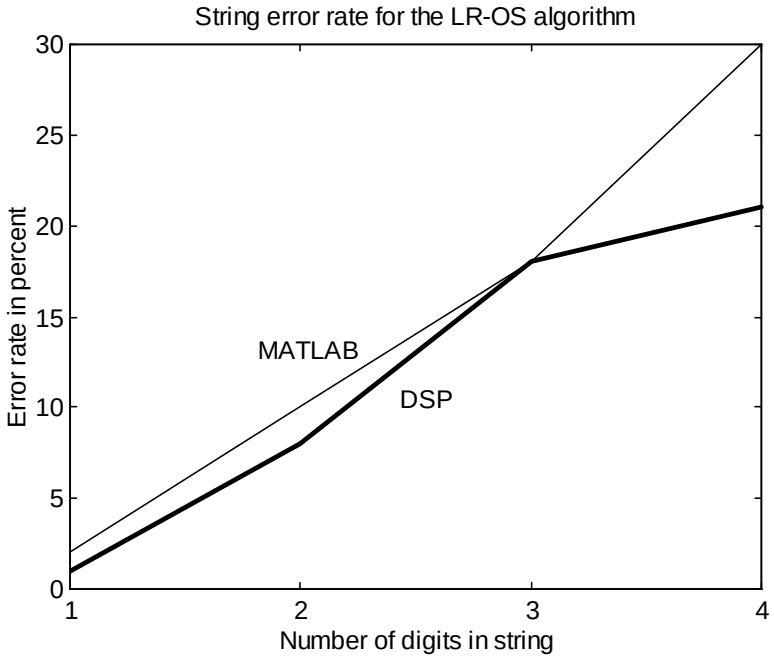


Figure 6.4 String error rate for the LR-OS method

Most of the differences between the results obtained from the ADSP-2181 and those obtained from MATLAB can be attributed to when the first time frame occurs. For a few of the test utterances, inconsistent results were observed when the speech utterance was tested over and over. The only difference in the calculations can be when the first frame was detected. This can vary from one test to another since the window length is 45 ms in duration (498 samples). In addition, at some point along the algorithm, the two best models were likely to be "close" where, depending on when the first frame occurs, one model can be chosen over the other or vice versa. Other differences can be attributed to the finite word length calculations used in the algorithm. However, as demonstrated in Section 4.2.2, this error is typically very small and will only be a factor when the two best models are "close".

7. Conclusion and Future Work

The material presented in this thesis describes the implementation of a speaker-dependent speech recognition system to recognize strings of words using continuous Hidden Markov Modeling. For now, the vocabulary consists of the ten digits (0 through 9) of the English language. The main motivation for this is to incorporate a voice recognition system into a hands-free telephone system.

The basic speech recognition system is presented in Chapter 2. There are two basic phases for any speech recognition system. In the training phase, templates are created or parameters are modeled from multiple utterances of the same linguistic unit, depending on the type of recognition system being employed. The recognition phase uses these templates or parameters to find a match on a test string to perform the recognition. Chapter 3 discusses the theory behind hidden Markov modeling, which is the main algorithm for the speech recognition system. The Viterbi method for recognition and the Viterbi reestimation procedure for training are the main algorithms discussed. In addition, the mixture density problem is looked at, which is how the observation probability density function is being modeled as.

In Chapter 4, the implementation of the speech recognition system on isolated words is discussed. The training utterances are taken in from a PC with a 16-bit sound card. The Viterbi reestimation procedure with the EM algorithm for estimating the observation pdf is used to estimate the parameters of the HMM. The basic structure of the HMM used in this implementation is a left-to-right model with 7 states and 4 mixture

density components. The speech recognition system is implemented on MATLAB and on the ADSP-2181, a digital signal processor manufactured by Analog Devices. The speech recognition system performed exceptionally well on isolated words with recognition rates exceeding 99% in both the MATLAB and the ADSP-2181 implementations.

Chapter 5 discusses the connected word recognition problem. Two algorithms are presented to aid in this problem. The level-building algorithm extends the Viterbi decoding procedure by building a number of levels. The number of words in the string is found by determining which level produced the best likelihood. The recognized string is then found by backtracking. The one-stage algorithm is simpler in the sense that each word in the string is determined in one stage, instead of building a series of levels. A set of within-model and between-model transition rules govern where the boundaries between each word occur. The boundary analysis performed on the speech signal (see Section 5.2.1) determines these transition rules. The OS algorithm can be further divided into two algorithms: a left-right OS algorithm and a left-right-left OS algorithm. The LR-OS algorithm makes one pass through the connected string (starting at the first time frame) to perform the recognition. The LRL-OS algorithm makes two passes through the connected string (one starting at the first time frame and one starting at the last time frame) and then chooses the one that produced the greater likelihood. In addition, durational and syntactic constraints are discussed, which aid in increasing recognition rates.

The implementation of the connected word recognition system is discussed in Chapter 6. Both the LB and the OS algorithms were implemented and compared in

MATLAB. Recognition rates varied for the different methods. The best recognition rates were obtained with the LRL-OS algorithm using durational constraints. A maximum string recognition rate of 91% for a four-digit string was obtained with this procedure. In addition, the LR-OS algorithm was implemented on the ADSP-2181. The recognition rates for this implementation are comparable to those obtained in MATLAB.

While the recognition performance on connected digits yields acceptable performance on MATLAB for string lengths of four, there is still a lot of room for improvements. In addition, a training program can be implemented on the digital signal processor like the ADSP-2181 to complement the recognition phase of the program developed in this research. By incorporating the training algorithm into the digital signal processor, we can expect to see improvements in the recognition rates for strings since the speech utterances will also be taken in a similar fashion.

To improve recognition rates, more than one model can be used to represent a word. For example for any particular word, training utterances taken from words in isolation can be used for one model, while training utterances extracted from a string of words can be used for another model. In addition, a different training algorithm could be incorporated to improve recognition rates. The current training algorithm in this research incorporates a maximum likelihood estimator (MLE) with the Viterbi reestimation method. The algorithm estimates the model parameters by maximizing the likelihood over all the training utterances. However, this may not be the best method for training the system. The overall likelihood on the same training utterances from another model representing a different utterance may be similar to the overall likelihood from the model

representing the training utterance. Thus, we can expect errors to occur more frequently when this happens. Another training algorithm, called the maximum mutual information estimator (MMIE), attempts to find model parameters which maximize the difference in the overall likelihood among all the models [1][24].

The speech recognition system developed in this thesis can also be extended for a wider range of applications. A larger vocabulary can be incorporated to extend the capabilities of the system. If the system is to recognize sentences, syntactic constraints can be used to increase performance. For an even larger vocabulary, the models should represent subunits of words which are concatenated together to form the words in the vocabulary. In addition, the system can be extended into a speaker independent system. The training for this type of system should come from many different people, so as to represent the general population. In addition, multiple models should be used to represent one word to represent the different types of voices in the population. For example, one set of models can be generated from male speakers while another is generated from female speakers. In addition, models can also be generated based on different types of accents. With any speaker independent system, we can expect recognition rates to decrease since the user usually is not the one who trains the system.

Appendix A

The results of the training analysis as described in Section 4.1.1 for the second and last elements are presented below. Both the segmental *k*-means training algorithm and the EM training algorithm were run on a known testing set.

Figures A.1 through A.3 show the true histogram of the second element for all the observations and the histogram of this observation based on Viterbi decoding for the three states. Figures A.4 though A.6 show the true pdf and the reestimated pdf from both training procedures for the second element. Figures A.7 through A.9 show the true histogram of the third element for all the observations and the histogram of this observation based on Viterbi decoding for the three states. Figures A.10 though A.12 show the true pdf and the reestimated pdf from both training procedures for the third element.

Here, we see that the histograms reveal that, regardless of which method is incorporated, the histogram of the observation based on Viterbi decoding is very similar to the true histogram. However, when the pdf's were compared, we see that the ones produced using the EM algorithm match up better than the one produced using the segmental *k*-means algorithm.

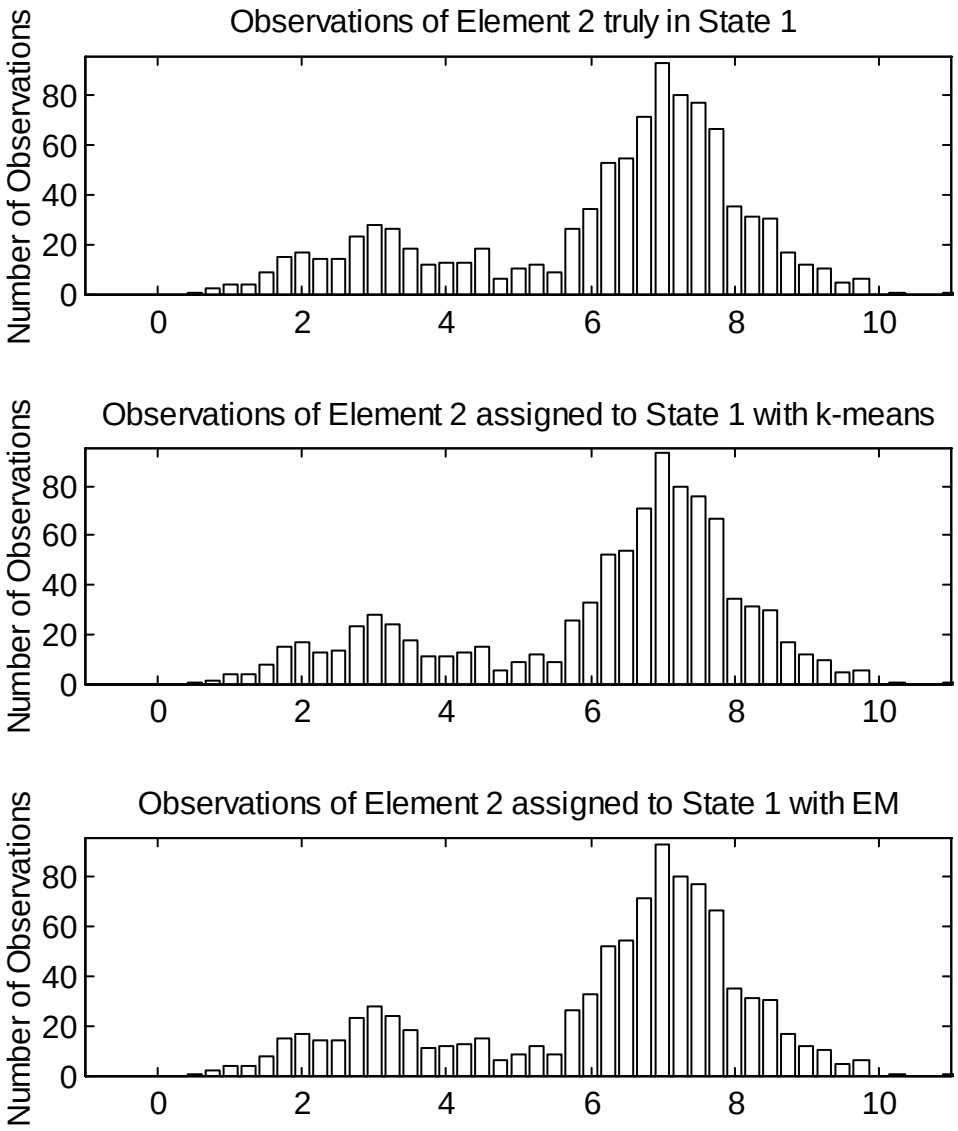


Figure A.1 Histograms for Element 2, State 1

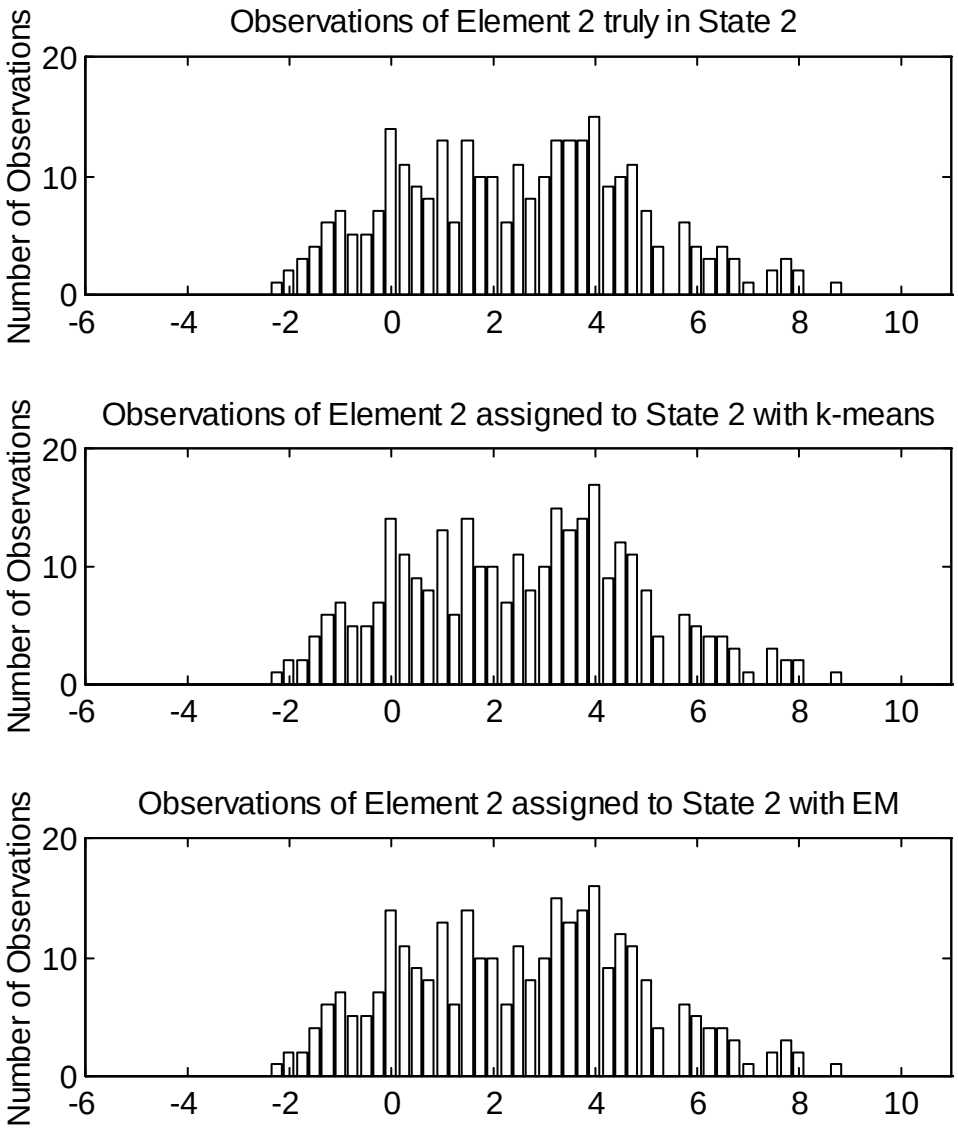


Figure A.2 Histograms for Element 2, State 2

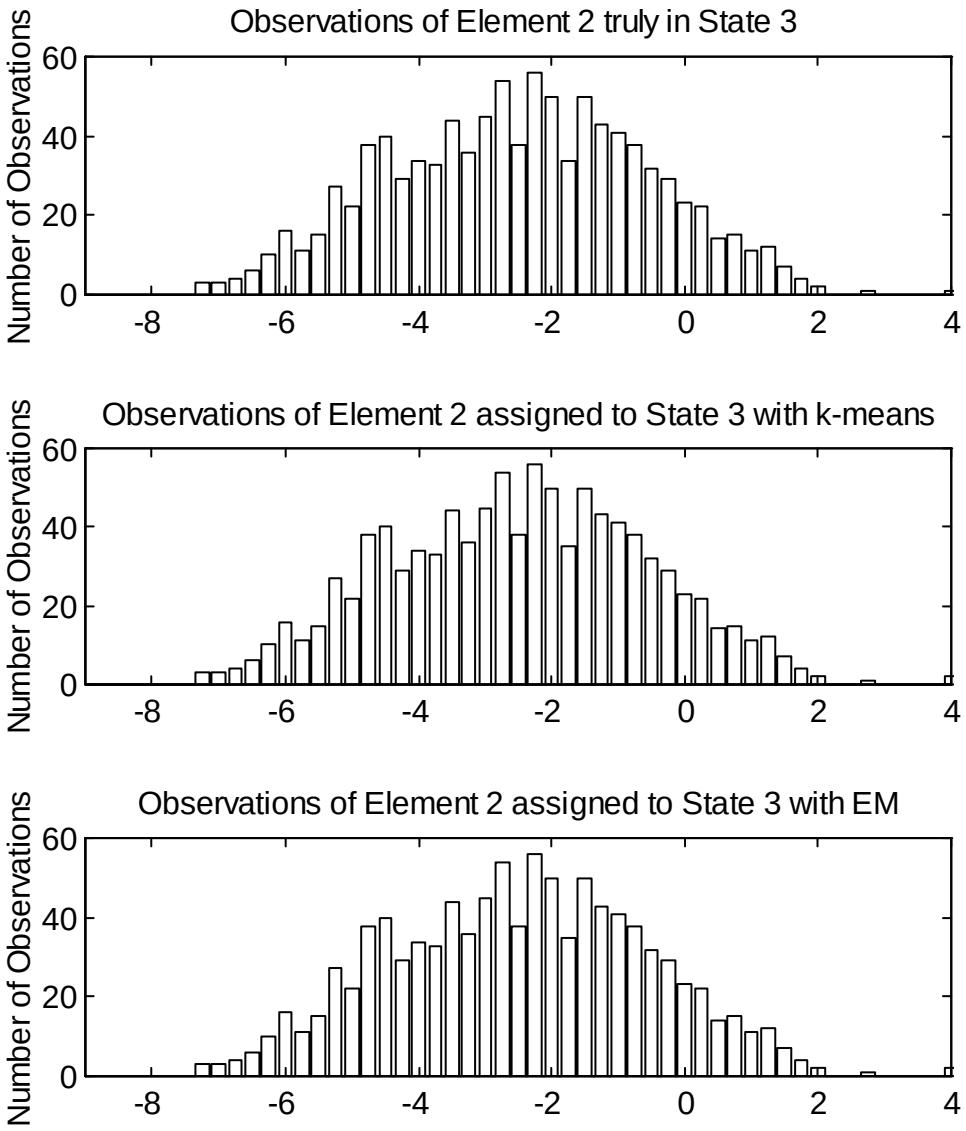


Figure A.3 Histograms for Element 2, State 3

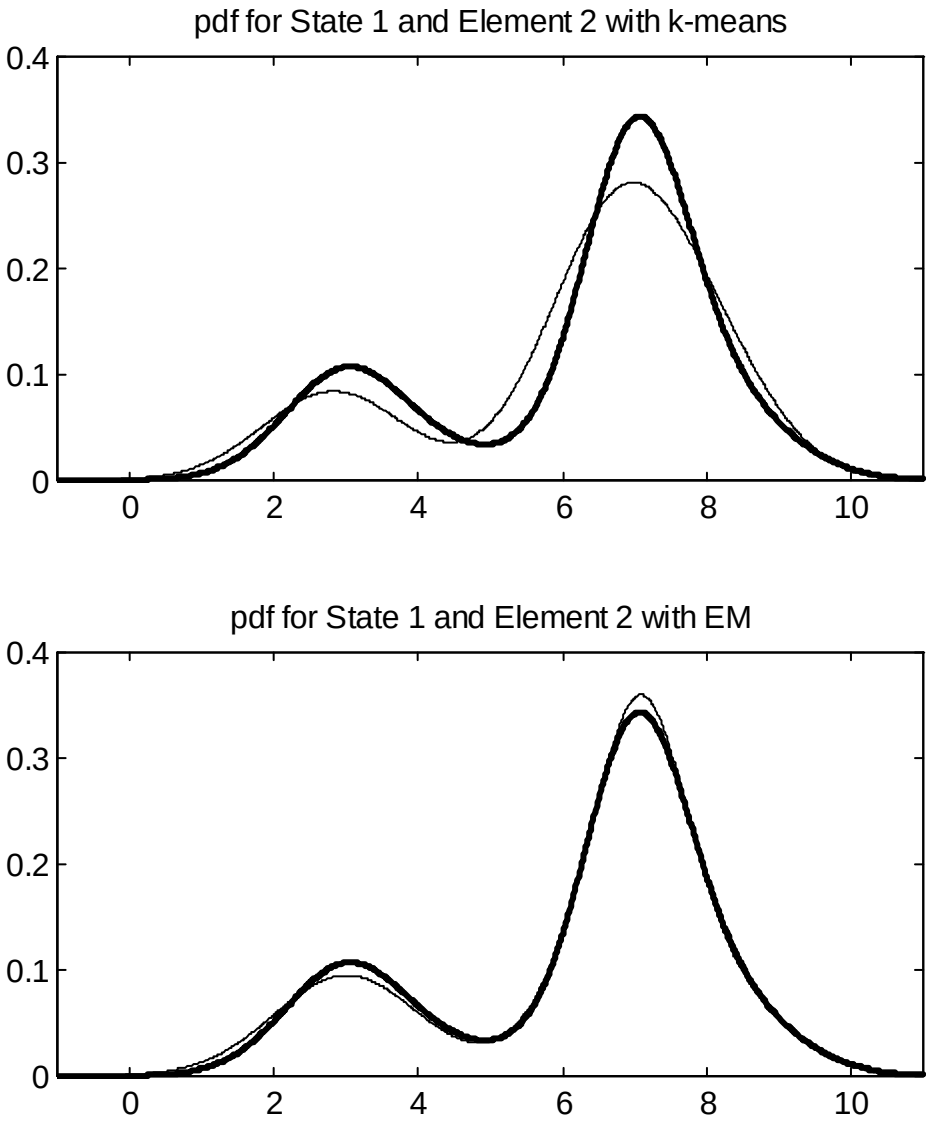


Figure A.4 True (▬) and Estimated (—) observation pdf's for Element 2, State 1

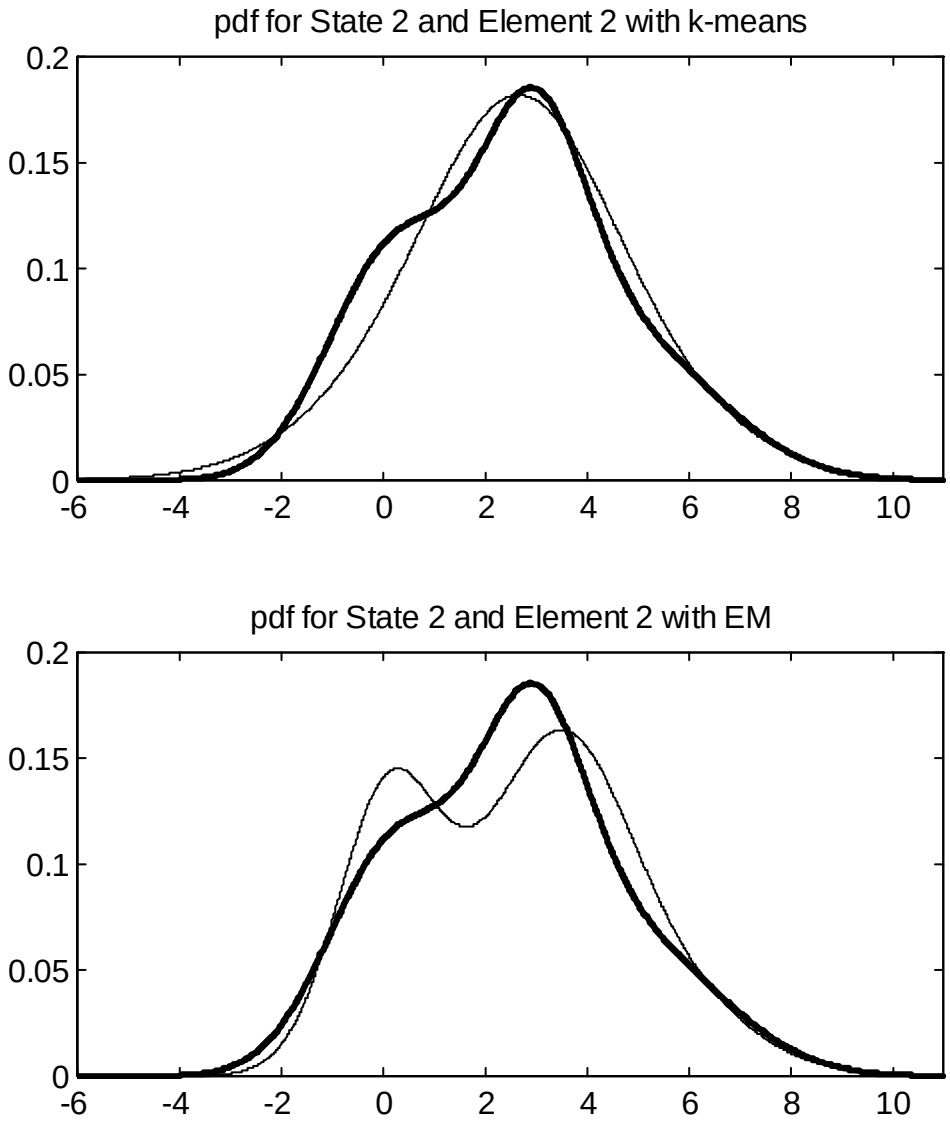


Figure A.5 True (▬) and Estimated (—) observation pdf's for Element 2, State 2

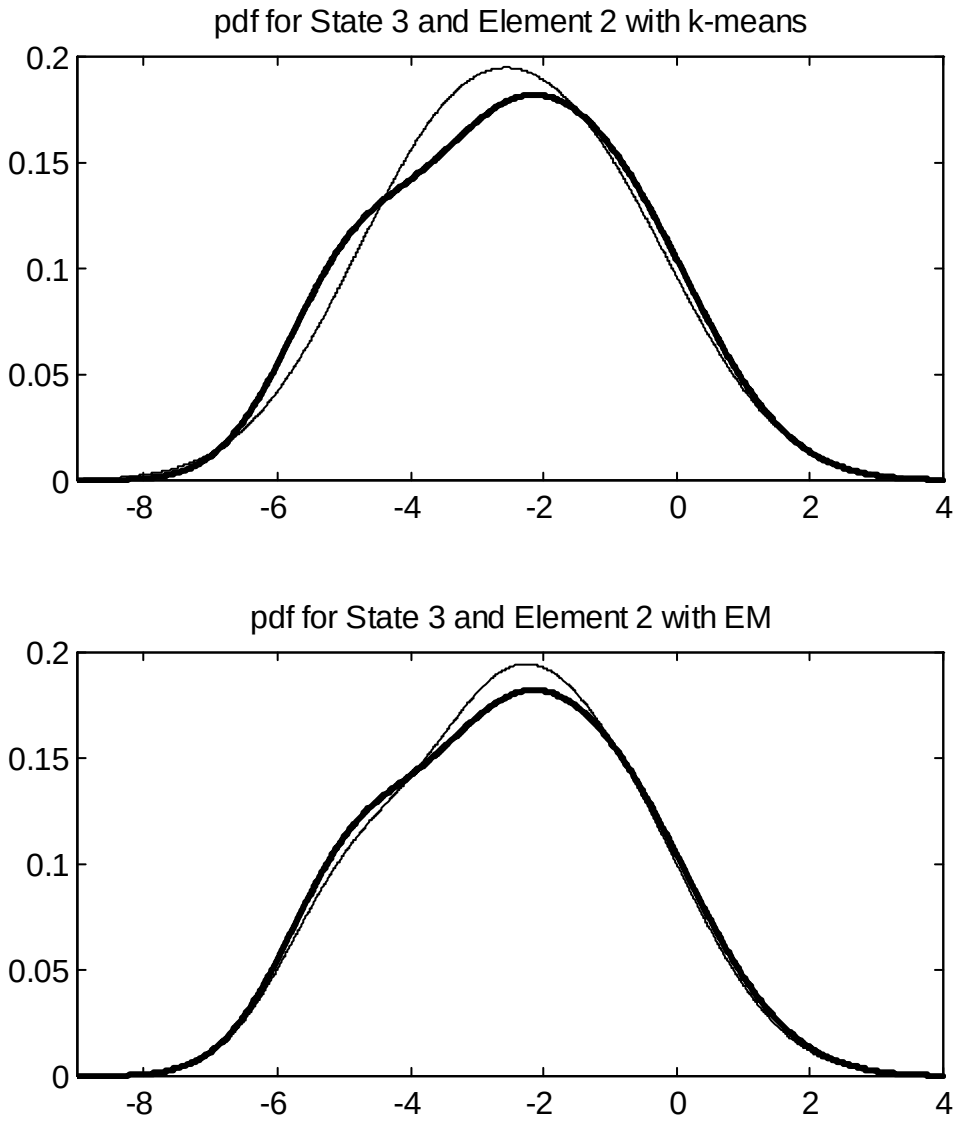


Figure A.6 True (▬) and Estimated (—) observation pdf's for Element 2, State 3

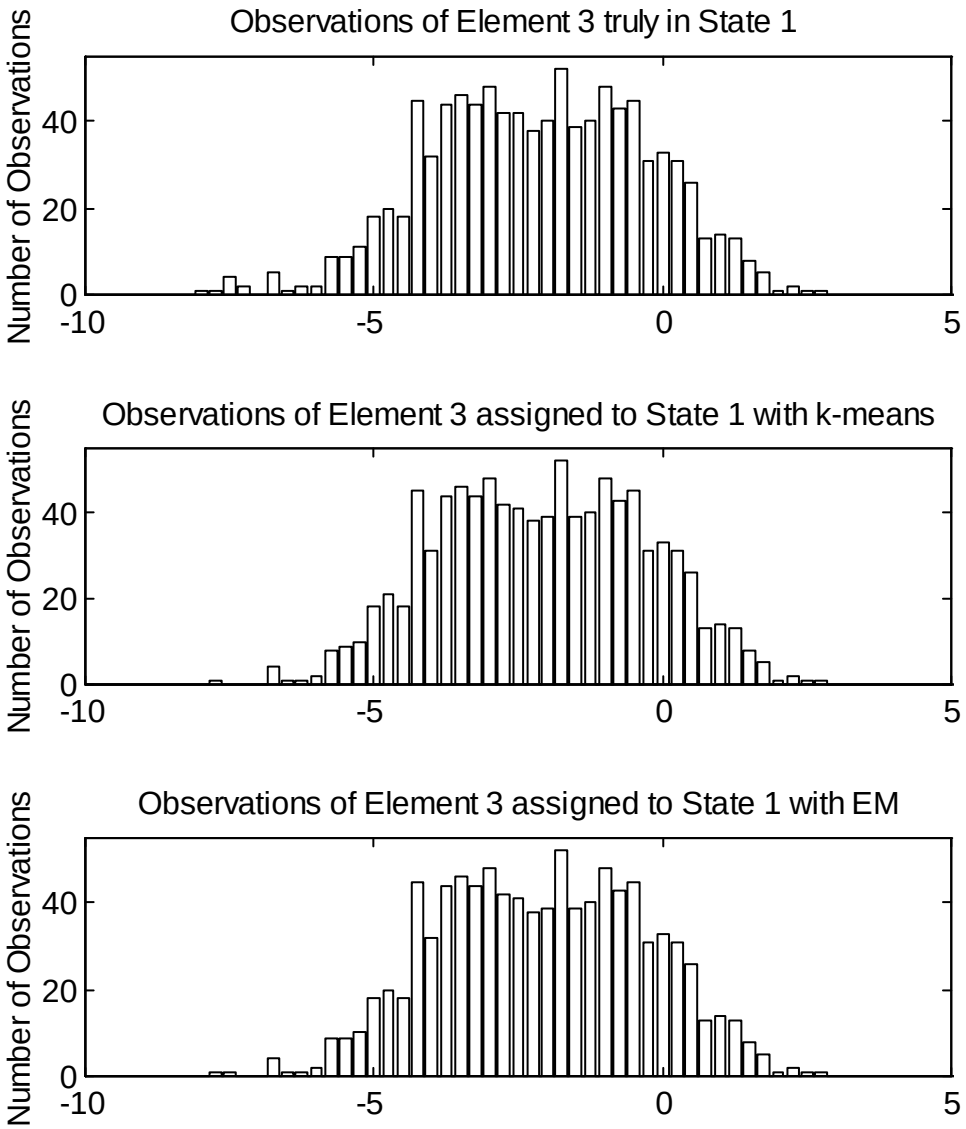


Figure A.7 Histograms for Element 3, State 1

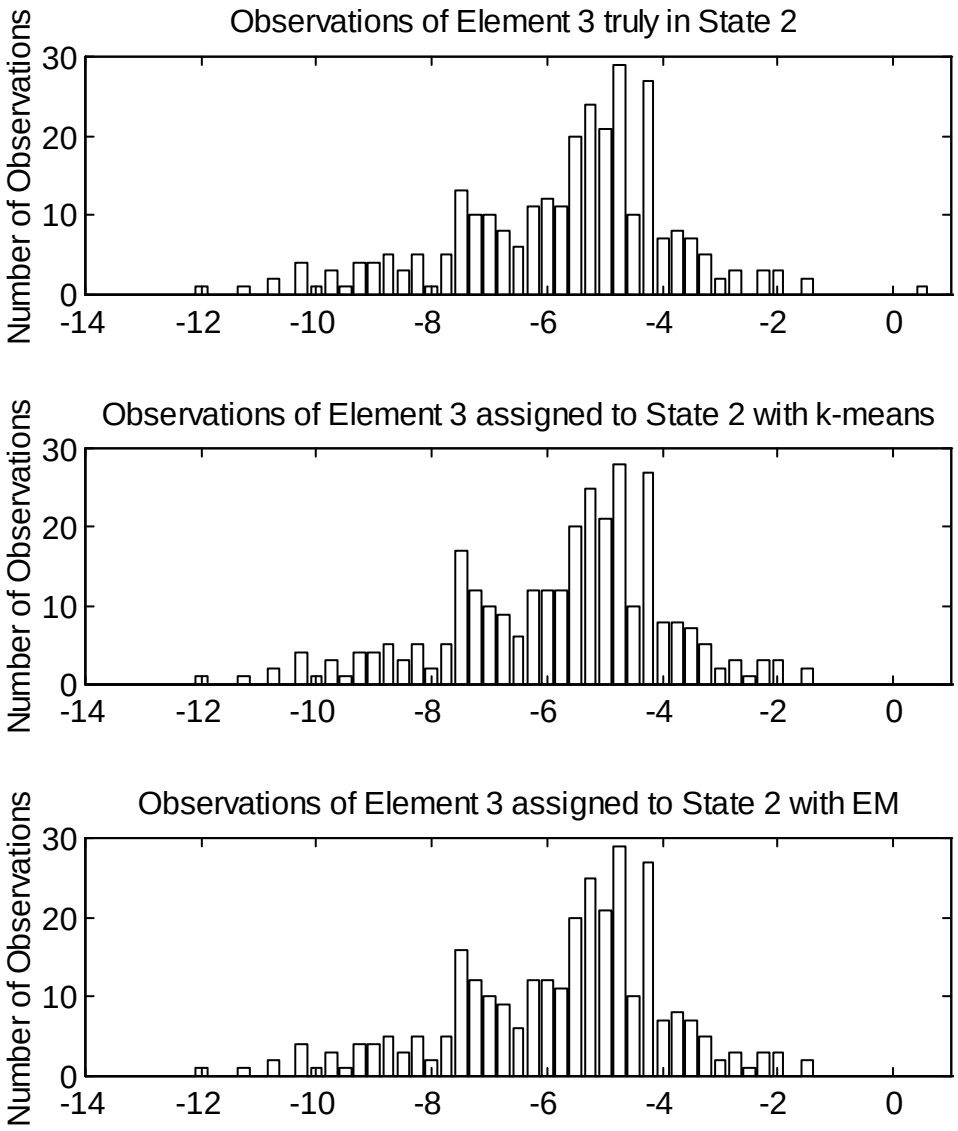


Figure A.8 Histograms for Element 3, State 2

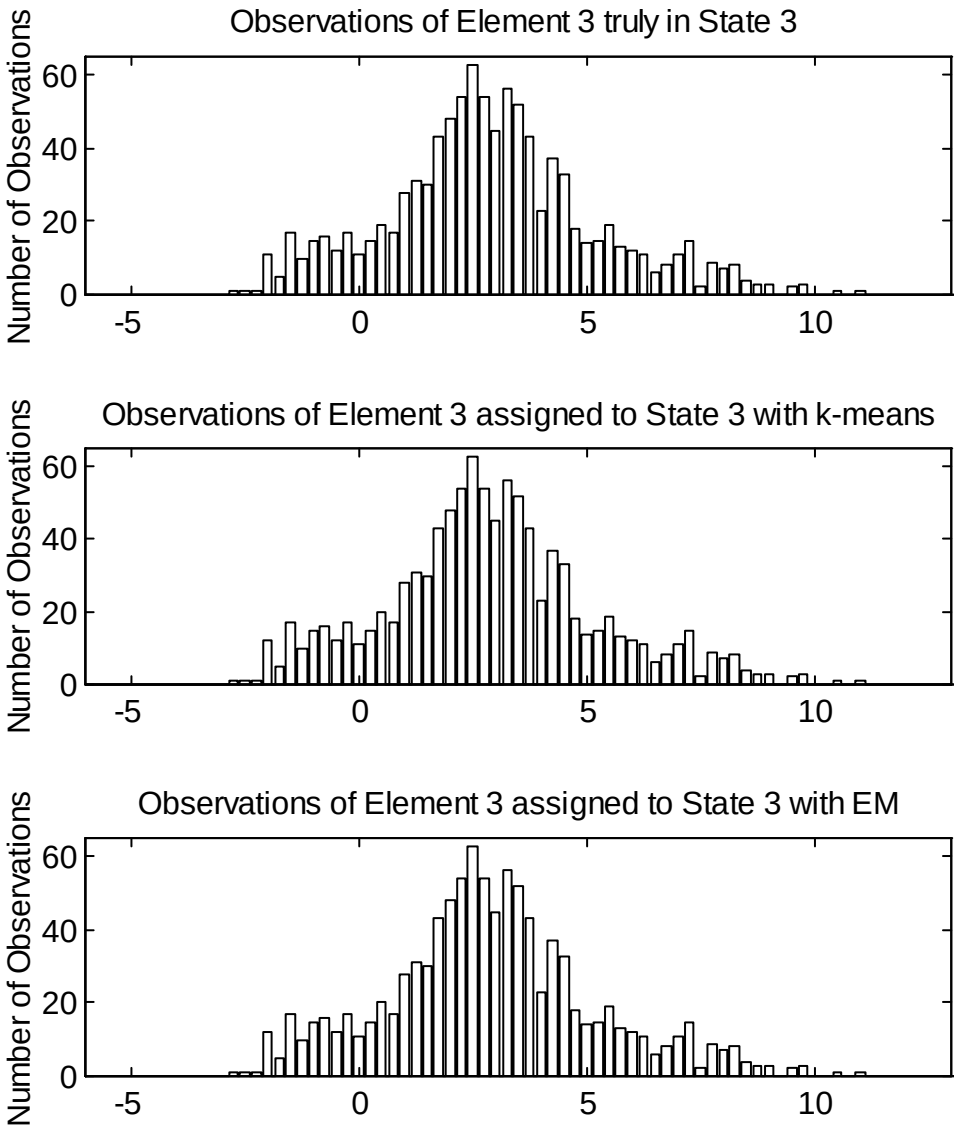


Figure A.9 Histograms for Element 3, State 3

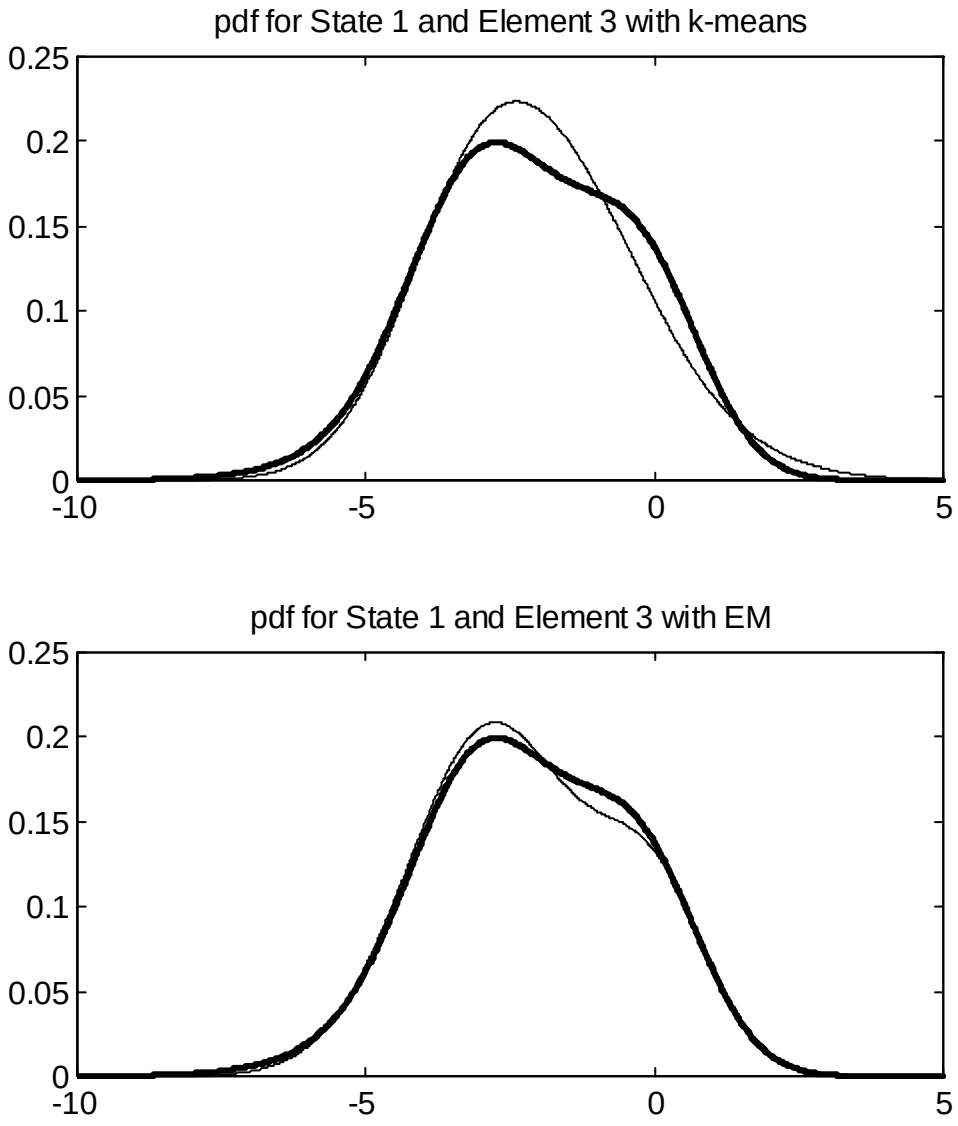


Figure A.10 True (▬) and Estimated (—) observation pdf's for Element 3, State 1

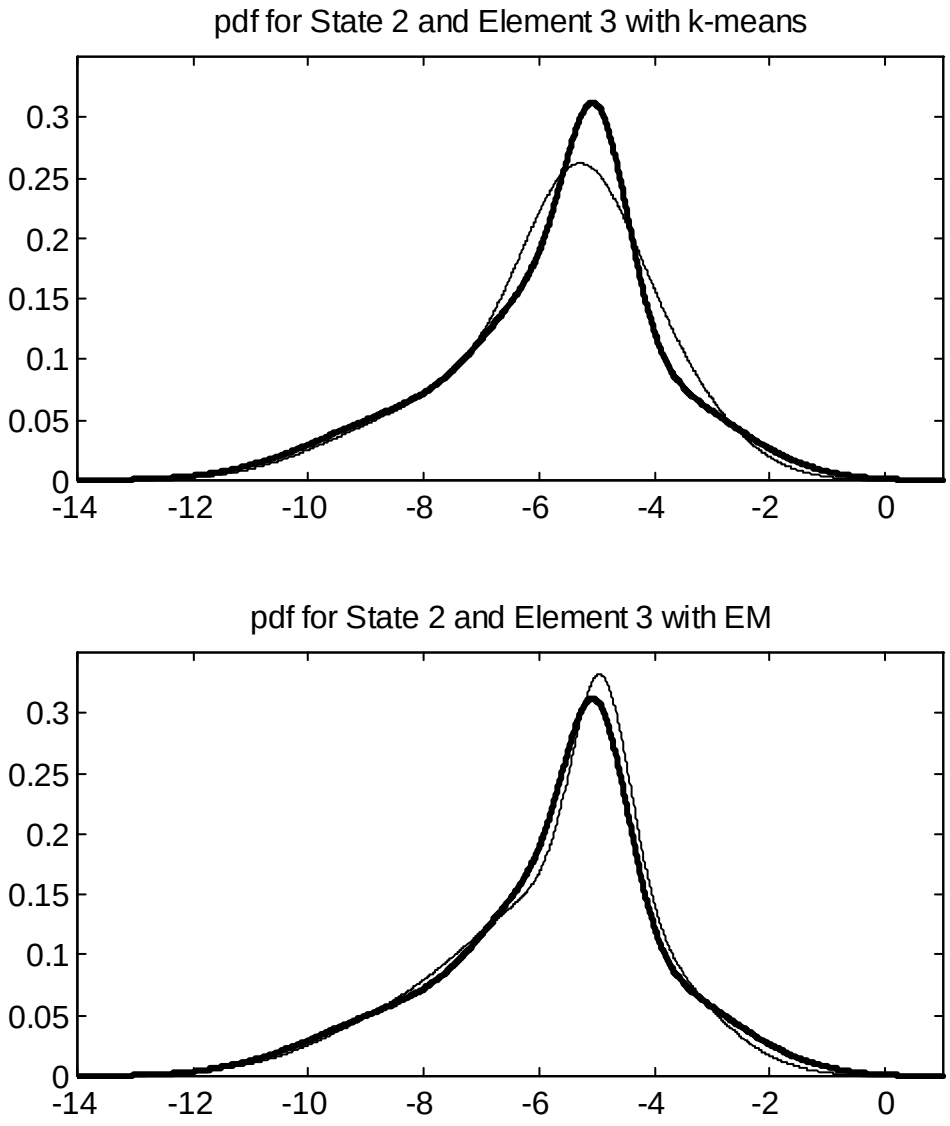


Figure A.11 True (▬) and Estimated (—) observation pdf's for Element 3, State 2

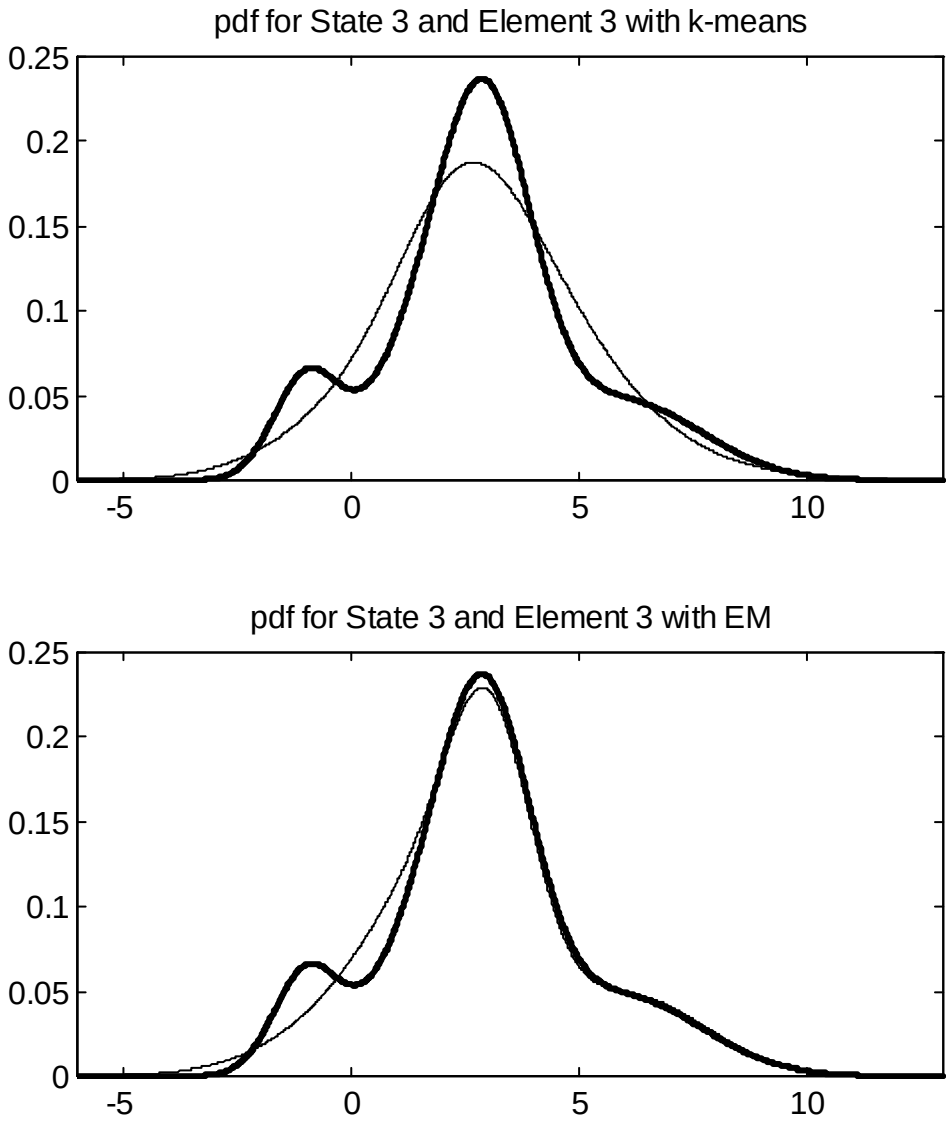


Figure A.12 True (▬) and Estimated (—) observation pdf's for Element 3, State 3

Bibliography

[1] J. R. Deller Jr., J. G. Proakis, and J. H. L. Hansen, *Discrete Time Processing of Speech Signals*, Macmillan Publishing Company, New York, 1993.

[2] G. Bristow, *Electronic Speech Recognition*, Collins Professional and Technical Books, London, 1986.

[3] S. Young, "A Review of Large-Vocabulary Continuous-Speech Recognition", *IEEE Signal Processing Magazine*, vol. 18, pp. 45-57, Sept. 1996.

[4] R. Haeb-Umbach, H. Ney, "Improvements in Time-Synchronous Beam Search for 10000-Word Continuous Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 353-356, 1994.

[5] X. Aubert, H. Ney, "Large Vocabulary Continuous Speech Recognition Using Word Graphs", *Proceedings on the International Conference on Acoustics, Speech, and Signal Processing*, vol.1, pp. 49-52, Detroit, 1995.

[6] P. S. Gopalakrisnan, L. R. Bahl, R. L. Mercer, "A Tree Search Strategy for Large Vocabulary Continuous Speech Recognition", *Proceedings on the International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 572-575, Detroit, 1995.

[7] Analog Devices, *Digital Signal Processing Applications using the ADSP-2100 Family Volume 2*, P T R Prentice Hall, 1995.

[8] J. D. Markel and A. H. Gray, *Linear Prediction of Speech*, Springer-Verlag, 1976.

[9] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *IEEE Proceedings*, vol. 77, pp. 257-285, Feb. 1989.

[10] Y. Tokhura, "A Weighted Cepstral Distance Measure for Speech Recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, pp. 1414-1422, Oct. 1987.

[11] Alan V. Oppenheim and R. W. Schafer, *Discrete Time Signal Processing*, Prentice Hall, 1989.

[12] L. B. Jackson, *Digital Filters and Signal Processing Third Edition*, Kluwer Academic Publishers 1996.

[13] J. G. Proakis and D. G. Manolakis, *Introduction to Digital Signal Processing*, Macmillan Publishing Company 1988.

[14] B. S. Atal, "Effectiveness of Linear Prediction Characteristics for Speech Recognition", *JASA*, vol. 55, no.6, pp. 1304-1312, June 1974.

[15] S. B. 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, pp. 357-366, Aug. 1980.

[16] L. R. Rabiner and M. R. Sambur, "Speech Endpoint Algorithm", *Bell System Technical Journal*, vol. 54, pp. 302-315, Feb. 1975.

[17] L. R. Rabiner and B. H. Juang, “An Introduction to Hidden Markov Models”, *IEEE Acoustic, Speech, and Signal Processing Magazine*, pp. 4-16, Jan. 1986.

[18] T. K. Moon, "The Expectation-Maximization Algorithm", *IEEE Signal Processing Magazine*, vol. 13, pp. 47-60, Nov. 1996.

[19] R. A. Redner and H. F. Walker, "Mixture Densities, Maximum Likelihood and the EM Algorithm", *Society for Industrial and Applied Mathematics*, vol. 26, pp. 195-239, Apr. 1984.

[20] A. Padmanabhan, *Continuous HMM Connected Digit Recognition*, M.S. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061, Sept 1996.

[21] Analog Devices, *ADSP-2100 Family User's Manual, Third Edition*, Analog Devices, Sept. 1995.

[22] Analog Devices, *ADSP-2100 Family EZ-KIT Lite Reference Manual, First Edition*, Analog Devices, May 1995.

[23] L. R. Rabiner and S. E. Levinson, "A Speaker-Independent, Syntax-Directed, Connected Word Recognition System Based on Hidden Markov Models and Level Building", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 33, pp. 561-573, June 1985.

[24] Y. Normandin, R. Cardin, and R. De Mori, "High-Performance Connected Digit Recognition Using Maximum Mutual Information Estimation", *IEEE Transactions on Speech and Audio Processing*, vol. 2, pp. 299-311, Apr. 1994.

Vita

Panaithep Albert Srichai was born on November 27, 1973 in Bronx, NY. Shortly after his birth, his family moved to Oregon, IL where he attended kindergarten and first grade. In 1980, his family decided to relocate to Man, WV where he complete his secondary education in 1991. Upon graduation, he attended Virginia Polytechnic Institution and State University obtaining a Bachelor's degree in Electrical Engineering in 1995 with summa cum laude honors. After his graduation from undergraduate school, he directed his career from engineering to medicine. After a year of pursuing the field of medicine, he decided to redirect his career from medicine to engineering. This decision was inspired by working at Hewlett Packard in California as a production engineer. This also lead him to continue his education for a Master's degree in Electrical Engineering at Virginia Polytechnic Institute and State University. During his last year of graduate school, he wrote his thesis on the topic of speech recognition. Albert has accepted a position with Hewlett Packard as a Production Engineer located in Rohnert Park, CA.