# A Comparative Study of Various Speech Recognition Techniques

Richard Charles Pitchers
B.Sc. Eng.

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering, in the Department of Electronic Engineering, University of Natal, South Africa.

Durban January 1990

# Preface

The author hereby declares that this thesis represents his own original and unaided work except where specific acknowledgement is made by name or in the form of a reference. The thesis has not been submitted to any other university for degree purposes.

# Abstract

Speech recognition systems fall into four categories, depending on whether they are speaker-dependent or independent of speaker population and on whether they are capable of recognizing continuous speech or only isolated words.

A study was made of most methods used in speech recognition to date. Four speech recognition techniques for speaker-dependent isolated word applications were then implemented in software on an IBM PC with a minimum of interfacing hardware. These techniques made use of short-time energy and zero-crossing rates, autocorrelation coefficients, linear predictor coefficients and cepstral coefficients. A comparison of their relative performances was made using four test vocabularies that were 10, 30, 60 and 120 words in size. These consisted of 10 digits, 30 and 60 computer terms and lastly 120 airline reservation terms.

The performance of any speech recognition system is affected by a number of parameters. The effects of frame length, pre-emphasis, window functions, dynamic time warping and the filter order were also studied experimentally.

# Acknowledgements

I would like to thank all members of staff of the Department of Electronic Engineering, University of Natal who assisted me at various stages during my work on this project. I would especially like to thank Professor Anthony D Broadhurst, under whose supervision this work was performed, for his patience and guidance.

I am grateful for the continual support that I received from my family and a special word of thanks goes to my father for his assistance with the grammatical presentation of this thesis.

I also wish to thank the Council for Scientific and Industrial Research for their financial support.

# Contents

# Chapter 1

## INTRODUCTION

As computers and computer-based machinery become more and more a part of our daily lives, so the ability to communicate with them in more natural ways becomes increasingly desirable. Up to now such communication has been almost entirely by means of keyboards and screens. Since speech is the natural means of communication between people, it is perfectly logical that we desire to speak to computers and for them to talk to us.

The communication between man and machine is a two-way process. The process whereby a machine produces a spoken message is known as **speech synthesis.** In the other direction where a man speaks to the machine, the process whereby the machine interprets the received message is known as **speech recognition.** Closely related to speech recognition is the topic of **speaker recognition.** Although neither speech synthesis nor speaker recognition are within the scope of this study, they are described briefly below.

Speaker recognition involves either deciding which member of a limited population is speaking (speaker identification), or confirming, with some measure of confidence, that the claimed identity of the speaker is in fact correct (speaker verification). Speaker identification becomes less certain as the population size increases whereas speaker

verification is relatively independent of population size. This is due to the fact that for identification, the speaker's voice has to be compared with reference patterns for the entire population. Verification, on the other hand, entails only one comparison, and a decision whether to accept or reject the speaker, based on a statistical confidence level. If speaker verification could be developed to the point where a speaker's voice was as good a proof of his identity as his signature, then it would open up endless possibilities for banking and the 'cashless society'.

Speech synthesis is a reasonably well established art and although early synthesizers sounded very artificial, the speech quality that can now be obtained is of a high standard. In one method of speech synthesis, words are generated by stringing together a number of 'phonemes', the basic unit of speech. Although this can lead to an infinite vocabulary, it generally produces a rather monotonous speech output. Another approach attempts to build a machine with powers of speech comparable to a human. Such systems are based upon a speech production model, and message formation rules are used to generate the required speech output.

Progress in speech recognition seems disappointingly slow by comparison to speech synthesis. Speech recognition is inherently a much more difficult problem. The human speech recognition mechanism is very dependent on intelligence and it is perhaps unfair to expect a machine to match the capabilities of someone who has had a lifetime of training in spelling, grammar, literature and logic - all of which play an important role in word recognition.

## 1.1 Early Speech Recognition Systems

In 1930, the German patent authorities rejected an application for a patent from T. Nemes for a phonetic typewriter, because in their scientific opinion "a phonetic typewriter was impossible in principle" [1]. Although he later proved that his proposal, which used optical rather than electronic techniques, was feasible, he was prevented from developing the idea further by the outbreak of war.

The first real interest in automatic speech recognition started some 35 years ago, mainly with the availability of electronic hardware to perform spectrum analysis of signals. The principle behind almost all earlier speech recognition systems was a comparison of a standard set of stored spectrum patterns, one for each phoneme, with the incoming signal. These 'phoneme recognizers' did not meet with the success that was hoped for.

The results obtained by the first word recognizer (by Davis et al in 1952) [2], based on a procedure of matching whole spoken words against templates of the expected words, were rather impressive: by limiting the vocabulary to ten words, it was able to recognize digits spoken by a single individual with an accuracy of better than 95 percent. The speech signal was split up into two frequency bands, above and below 900Hz, and the principle frequency in each band ($F_1$ and $F_2$) was then calculated by simple zero-crossing counters. A plot (on an oscilloscope display) of $F_1$ and $F_2$ was made and compared with reference plots, and the recognition decision based on the best match.

In 1956, Olson and Belar [1] developed a phonetic typewriter capable of recognizing ten syllables (are, see, a, I, can, you, read, it, so and sir). The speech signal was passed through a bank of eight bandpass filters. The

filtered outputs were then rectified to obtain envelopes which were then routed to a bank of relays via a time sequence switch. This switch would route the signals to a fresh bank of relays every 40 mS. The relays would only close if the current through them exceeded a certain level and so a display showing which relays had closed effectively provided a simple frequency-time plot.

The growth in the use of digital computers in the early sixties led to a renewed interest in the speech field. Computers offered a convenient means of applying digital signal processing techniques and testing elaborate recognition algorithms. The first isolated word speech recognizer using a digital computer was one by Denes and Mathews [3] developed on an IBM 704 computer in 1960. The program normalized all utterances to a standard time duration before converting them to time-frequency patterns, which were cross-correlated with test patterns for each word in the ten-digit vocabulary. In tests with five male speakers, the computer achieved an accuracy of 88% without time normalization and 94% with normalization.

## 1.2 **Advantages of Speech Input Systems** [4]

1. Speed of communication - speech input is much faster than standard manual input for continuous text.
2. Increased reliability - direct data entry from a remote source can increase reliability.
3. Parallel channel - it provides an independent communication channel in hands-busy situations.
4. Freedom of movement - the user may move about freely while doing a task.
5. Untrained users - there is no training in basic skill required since speech is natural for all users.
6. Possible cost saving - where the number of people required to do a task is reduced.

## 1.3 **Applications of Speech Recognition**

Speech input is potentially of benefit in any task that is slowed down by the need for manual data entry. In many applications, the vocabulary required is limited, single-word commands are generally acceptable, and there are only a few speakers, so good accuracy is possible at reasonable cost.

Historically, operation of a machine required learning to manipulate special dials or keys in some specified sequence. Any deviation from this procedure produced errors which were not easily detectable because of the complexities of the rules. A speech recognition system makes it possible for the first time for humans to control mechanical systems with voice commands.

Voice entry can also be applied to many inspection processes. For example, in the case of an automobile inspection, the inspector can walk around the vehicle (with a wireless microphone) and record each defect by saying an item number and a word describing the defect. The defect information can then be collected and printed on an inspection ticket automatically.

Another speech recognition application is in the placing of reservations over the telephone. This is possible by asking specific questions that force the caller's reply to come from a finite vocabulary which can then be recognized. Examples of such applications include cinema and airline ticket bookings.

A 'speech-input typewriter' could be thought of as the ultimate goal of speech recognition. Such a machine is, however, of little use unless it has a very high degree of accuracy. The difficulties associated with speech recognition (discussed in the following chapter) give some insight into why such a typewriter will undoubtedly not become a reality this century and possibly not in my lifetime.

## 1.4 Thesis Outline

The primary goal of this thesis was to study various speech recognition techniques that have been used to date, and then implement three or four of these techniques. A total of four techniques were implemented and a comparison of their performances is given.

Chapter 2 gives an overview of speech recognition. It defines the different categories of speech recognition systems, briefly describes the processes involved in a typical word recognition system and mentions some of the different techniques that can be used.

Chapter 3 focuses on the speech production process and discusses the natures of the different types of speech sound. The important characteristics of the different classes of speech sound are described. Finally a simplified mathematical model of the speech production process is defined.

Speech recognition techniques differ primarily in the features that they extract from the speech signal. Chapter 4 describes in some detail the different features that can be extracted and how they are used in the speech recognition process. Some of the major problems associated with detecting the end points of a word are discussed and methods for detecting these end points are given - the accuracy of this detection being of utmost importance for reliable word recognition. Finally, various methods of matching the unknown spoken word to one of the stored reference patterns are discussed. Dynamic time warping is described in some detail and a computationally efficient algorithm, known as the Ordered Graph Search Approach [5], is also described.

Chapter 5 describes the hardware that was used to implement the speech recognition system. This hardware consisted of an IBM compatible PC, an A/D converter, a microphone and an input filter and amplifier. The algorithms used to implement the various methods of speech recognition are then described with the aid of flowcharts. The four different vocabularies used are discussed briefly.

A comparison of the results obtained using the different techniques is given in chapter 6. Furthermore, the effect of various parameters that influence the performance of a speech recognition system were studied experimentally.

Chapter 7 concludes the report by discussing the results achieved and considering the future prospects of speech recognition.

# Chapter 2

# AN OVERVIEW OF SPEECH RECOGNITION

## 2.1 Classes of Speech Recognition Systems

Speech recognition systems can be divided into four categories, depending on whether they can handle continuous speech or just isolated words and on whether they are speaker-independent or must be 'trained' by the individual who is going to use them.

### 2.1.1 Isolated-Word Recognition

Isolated speech recognition systems can be defined as those systems that require a short pause before and after utterances that are to be recognized as entities. The minimum duration of a pause that separates independent utterances is of the order of 100 ms. Anything shorter than 100 ms can be confused with stop-gaps (periods of silence in the middle of a word).

In an isolated-word speaker-dependent system, every word that is to be recognized by the system has to be first pronounced by the user, in order to create word templates. Once this has been done, an incoming signal is identified as the word whose template matches best with the entire input signal.

In speaker-independent systems, clustering techniques are applied to large collections of isolated word samples, in order to fabricate speaker-independent templates. A large number of persons are selected as representative of the different ways of speaking and are required to utter each word of the vocabulary 3 to 5 times. Acoustically similar samples are grouped together to form a reference template. It has been found that 8 - 14 templates per word give a fairly adequate representation of the different ways of pronouncing a word.

### 2.1.2 **Continuous Speech Recognition**

Continuous speech recognition systems are much more difficult to realize than isolated speech recognition systems. When words are spoken in a normal connected fashion, it is generally not possible to determine where one word ends and the next one starts independently of identifying what the words are. For example, in the sequence 'six teenagers' it would be difficult to be sure that the first word was 'six' and not 'sixteen' until the last syllable of the phrase has been spoken, and 'sixty' might also have been a possible choice before the /n/ occurred. It is obvious that the simple matching of word patterns between already-specified end points is no longer possible for continuous speech.

## 2.2 The Speech Recognition Process

Even though speech recognition systems vary greatly in detail, they all use the same basic recognition process. A spoken word is converted into an electrical signal by a microphone and the signal is processed to extract a set of identifying features. The features are then compared to a library of templates representing the machines vocabulary. A word is recognized if it matches one of the templates stored in the machine's memory.

Figure 1.1 shows the four main subsections making up a speech recognition system - namely feature extraction, a pattern similarity measurement, a decision based on this measurement, and the creation of a set of reference templates.

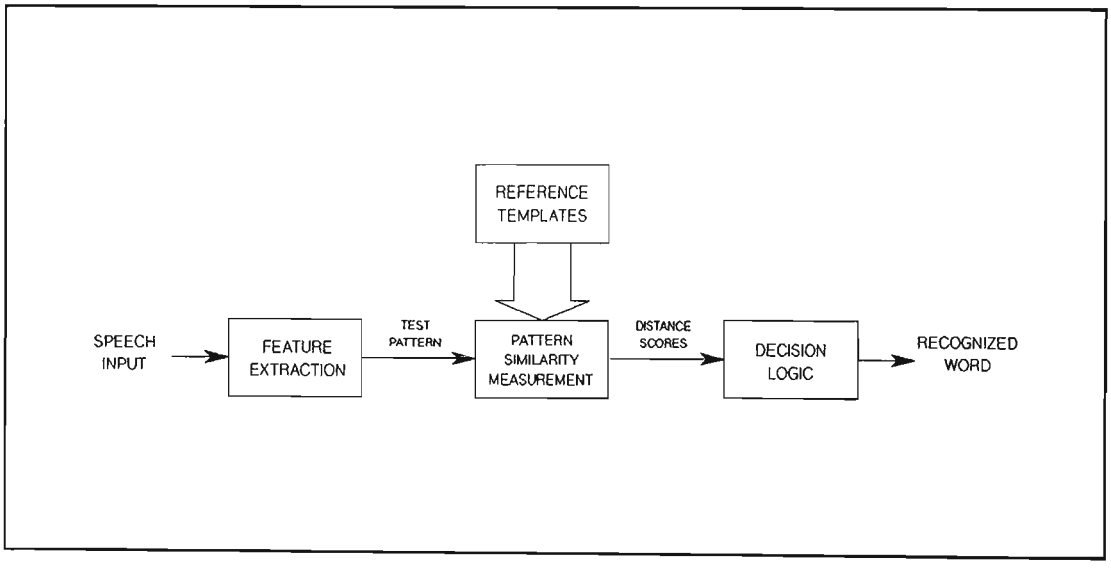



**Fig. 2.1 Block Diagram of Word Recognition System** [6].

The analysis stage of speech recognition consists of extracting identifying characteristics from the speech signal. After analysis, speech signals must be compared to reference templates before they can be recognized. One

difficulty with template matching is that speech elements are rarely the same length as the templates they are supposed to match. Pattern comparison is dealt with in more detail at a later stage.

The stored templates are usually created by using the recognition machine itself in a training mode. To train a speech recognition system, a speaker repeats each word several times (typically 3 to 5 times) to enable the machine to compute an average template for that word and that speaker.

## 2.3 **Speech Recognition Techniques**

The various approaches to recognition differ principally in what features they extract from the input and in the algorithms that do the matching.

These different techniques, discussed in detail in chapter 4, include: time domain measurements such as energy, zero-crossings, bandpass filter outputs; frequency domain measurements such as spectral coefficients, cepstral coefficients, spectral derivative; Stochastic models; and LPC parameters.

## 2.4 The Complexity of Speech Recognition

### 2.4.1 Practical Considerations

It is informative to look at some of the practical aspects of a speech recognition system intended for operational use. A high-quality wide-range microphone will naturally pick up any background noise in the immediate vicinity of the individual attempting to use the speech recognition system. One solution to this problem is to remove the interfering noise by placing the individual in an acoustically shielded environment. This restriction is more often than not too severe. An alternative method of removing interfering sounds is to eliminate the noise at the microphone itself. A close-talking noise-cancelling microphone mounted on a lightweight headset will achieve an acceptable degree of noise cancellation with minimal degradation of the speech signal.

Another critical factor to be considered is the detection of extraneous signals caused by breath noise. A strong tendency exists to exhale at the end of isolated words and to inhale at the beginning. Inhaling produces no significant direct air blast on the microphone, whereas exhaling can produce signal levels comparable to speech levels.

Background noise and breath noise can make the task of accurately determining the beginning and ending of the word even more formidable. Accurate word boundary detection is the single most critical factor in an isolated word recognition system. Endpoint detection is discussed in further detail in section 4.13.

### 2.4.2 Speech-related Difficulties

When humans listen to speech they do not hear an unambiguous sequence of sounds, which can be decoded one by one into phonemes and grouped into words. In normal conversation we are able to ignore false starts to words, hesitations and mild stuttering, all of which are extremely common.

All sorts of information is taken into account by a person listening to somebody speaking. The listener very often knows what a speaker is likely to speak about, either from what he knows about the speaker or, if the conversation has been in progress for some time, from the previous context. Very often an entire word can go completely unheard and yet it is obvious what was said.

Another difficult task for any speech recognition system, a task that the human handles with ease, concerns the ambiguity associated with homonyms (words with the same pronunciation as another but a different spelling and meaning). For example the words 'to', 'two', and 'too' or the words 'bare' and 'bear'.

We are all familiar with the fact that in a crowded room, such as at a cocktail party, people can converse with the group of people in their immediate vicinity, even though there is a lot of competing speech from all the other people in the room. Firstly, having two ears enables some directional discrimination to be used. The other important factor is that by being able to see the speaker, it is possible to correlate the acoustic signal with observed lip movements, and with other gestures which may be used to supplement the speech.

### 2.4.3 **The Need for Artificial Intelligence**

From the above it is clear that the human hearing process consists of far more than just 'hearing'. The human perceptual and cognitive systems must be enormously complex to be able to perform the task of linguistic processing. Because verbal communication is inextricably intertwined with human intellect, matching the human speech recognition capabilities would require duplicating the capabilities of the human brain [7]. It is obvious that it is not possible for a machine to emulate human performance without the machine having a very high degree of artificial intelligence.

# Chapter 3

## THE SPEECH PRODUCTION MECHANISM

### 3.1 Introduction

Before being able to apply digital signal processing techniques to speech signals, it is necessary to understand the fundamentals of the speech production process. A speech signal is composed of a sequence of sounds, the order of which is governed by the rules of language. The study of these rules and their implications falls within the subject of **linguistics**, and the classification and generation of the individual sounds is known as **phonetics.**

### 3.2 Phonemes and Allophones

The phoneme can be thought of as the basic unit of a spoken language in that it represents the smallest segment of sound such that it cannot be further broken up. More precisely, a phoneme is the smallest unit of speech such that if one phoneme in a word were substituted for another, there might be a change in meaning. The need for phonemes arises from the many different ways of representing one particular sound. For example the '**f**' sound in '**f**un', 'o**ff**', '**ph**one' and 'rou**gh**'.

Phonemes are written down using a set of internationally recognized symbols and it is usual to write them between oblique lines. The phonetic symbols of many English consonants are the same as the letters of the conventional alphabet; they are /b/, /d/, /f/, /g/, /h/, /k/, /l/, /m/, /n/, /p/, /r/, /s/, /t/, /v/, /w/, /z/. A table of all other phonetic symbols and their interpretations is given in appendix A.

A phoneme is not itself a sound since it can often be produced or pronounced in different ways. For example the phoneme /p/ in 'pit' and 'pull'. Different sounds that are represented by the same phoneme are known as allophones of that phoneme. Allophones are written between square brackets to distinguish them from phonemes.

## 3.3 **The Basics of Speech Production**

The main organs responsible for producing speech are the lungs, larynx (consisting of the glottis and vocal folds), pharynx (throat), nose and mouth. These are illustrated by the cross-section shown in Fig. 3.1 [8].

The glottis is an air valve formed by the vocal folds (also known as the vocal cords) and is covered by the epiglottis, a flap of soft tissue. In order to speak, pressure must build up below the glottis and when this pressure is great enough, a pulse of air is released. The modulation of the air stream through the vocal cords is known as phonation. The larynx performs the intricate task of holding the vocal folds in the correct position and state of tension for phonation.

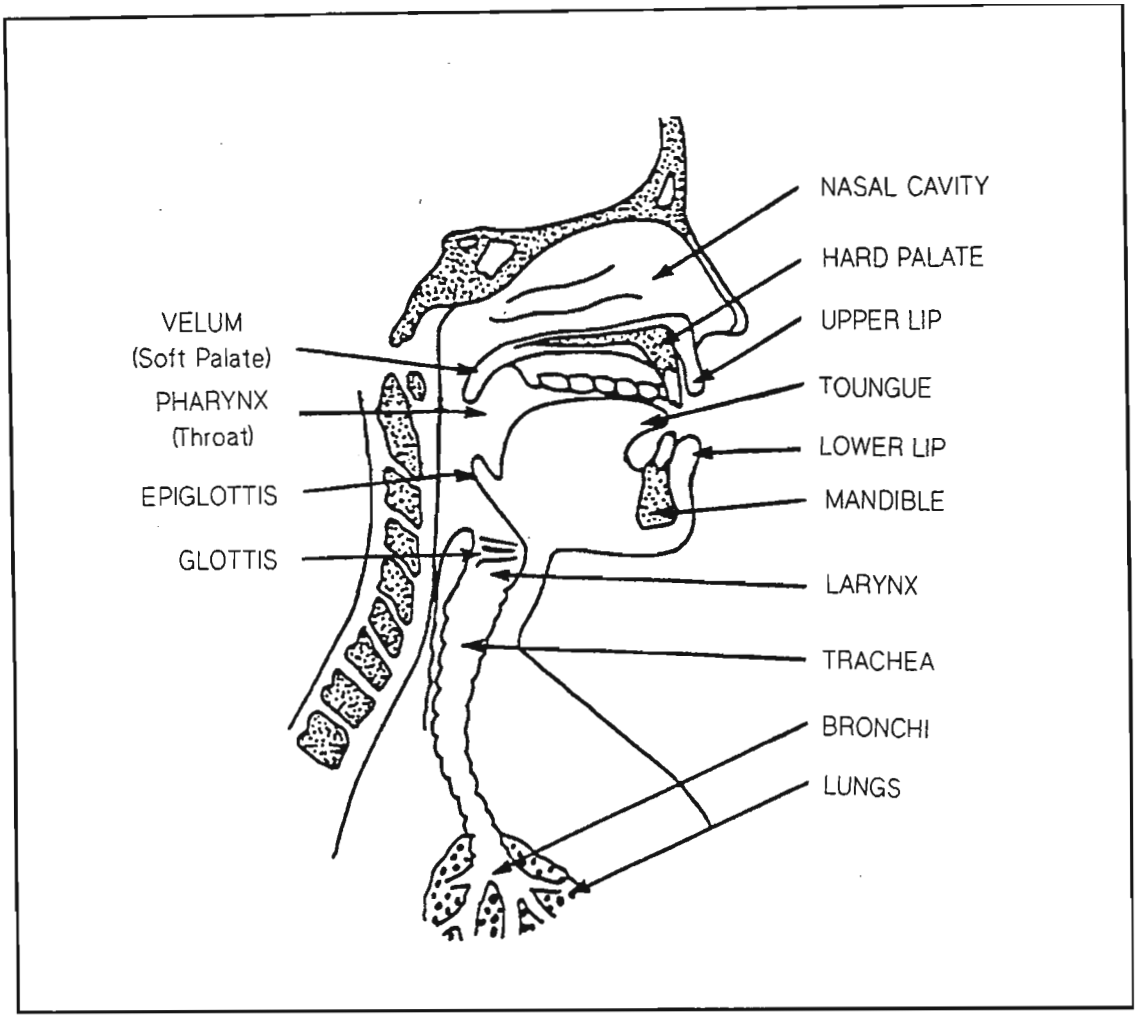



**Fig. 3.1 Cross-section of the human head, showing the vocal organs [8].**

The frequency of vibration (pitch) of the sound depends on the tension in the vocal folds - tense vocal cords vibrate at a higher frequency than relaxed ones. The loudness of the speech is obviously related to lung pressure. As the pressure increases, the vocal folds have to be squeezed together more tightly to retain it.

## 3.4 Classes of Speech Sounds

Speech sounds can be categorized into three distinct classes according to their mode of excitation. These are voiced sounds, fricatives and plosive sounds.

### 3.4.1 Voiced Sounds

Voiced sounds include all vowels, semi-vowels, diphthongs and the nasal consonants. They are generated by adjusting the tension of the vocal cords so that they vibrate as air is forced through the glottis.

The mouth, nose and throat are the resonating cavities of the vocal tract. The nasal cavity has a relatively fixed shape, but the oral cavity can adopt a wide variety of shapes, by movement of the jaw, tongue, lips and soft palate.

As sound propagates through the vocal tract, the frequency spectrum is shaped by the frequency selectivity of the vocal tract. This effect is very similar to the resonance effects with organ pipes. In the context of speech production, the resonance frequencies of the vocal tract are called **formant frequencies.**

### 3.4.1.1 **Vowels**

During a vowel sound, the air in the vocal tract vibrates at three or four frequencies simultaneously, the fundamental frequency being determined by the rate of vibration of the vocal cords. The values of the formant frequencies are determined by the cross-sectional area and position of the constriction in the vocal tract as well as by the size of the lip opening.

From a speech recognition point of view, information about formants is useful for the analysis of vowel sounds. Table 1 gives average values of the first three formant frequencies of vowels for male speakers as measured by Peterson and Barney [9].

| Phonetic Symbol | Typical Word | $f_1$ (Hz) | $f_2$ (Hz) | $f_3$ (Hz) |
|---|---|---|---|---|
| /i:/ | beet | 270 | 2290 | 3010 |
| /I/ | bit | 390 | 1990 | 2550 |
| /E/ | bet | 530 | 1840 | 2480 |
| /ae/ | bat | 660 | 1720 | 2410 |
| /A/ | but | 520 | 1190 | 2390 |
| /o/ | hot | 730 | 1090 | 2440 |
| /o:/ | bought | 570 | 840 | 2410 |
| /U/ | foot | 440 | 1020 | 2240 |
| /u:/ | boot | 300 | 870 | 2240 |
| /E:/ | bird | 490 | 1350 | 1690 |

**Table 1 Formant Frequencies for the Vowels**

### 3.4.1.2 **Semi-Vowels**

The group of semi-vowels, so-named because of their vowel-like nature, consists of /w/, /l/, /r/ and /y/. A semi-vowel is usually followed by a vowel and is characterized by a gliding change in position as required by the following phoneme.

### 3.4.3.3 **Diphthongs**

A diphthong may be defined as a monosyllabic vowel-like sound that starts at or near the articulatory position for one vowel and moves toward the position for another. Examples of the five diphthongs in English are given in Table 2 together with the variation of their first two formant frequencies as measured by Holbrook and Fairbanks [10].

| Phonetic Symbol | Typical Word | $f_1$ (Hz) | $f_2$ (Hz) |
|---|---|---|---|
| /eI/ | pray | 355 - 560 | 2050 - 2350 |
| /oU/ | loan | 580 - 425 | 1000 - 740 |
| /aI/ | buy | 730 - 525 | 1250 - 2100 |
| /aU/ | how | 800 - 620 | 1400 - 840 |
| /oI/ | toy | 530 - 490 | 820 - 1900 |

**Table 2 Formant Frequencies for Diphthongs**

#### 3.4.3.4 **Nasal Consonants**

The nasal consonants /m̃/, /n/ and /ŋ/ are characterized by a total constriction at some point in the oral cavity, causing the airflow to be through the nose rather than the mouth. The three nasal consonants are distinguished by the point at which the constriction is made. For /m/ it is at the lips, for /n/ the constriction is just behind the teeth, and for /ŋ/ it is just forward of the velum.

### 3.4.2 **Fricatives**

Fricative sounds are made by forcing a steady flow of air through a constriction made at some point in the vocal tract. The location of the constriction serves to determine which fricative sound is produced. Fricatives may be either voiced or unvoiced sounds depending on whether or not the vocal cords are excited (vibrating). The unvoiced fricatives are /f/, /th/, /s/ and /sh/, while /v/, /z/ and /zh/ are voiced fricatives. For the voiced fricatives there are two sources of excitation. One at the point of constriction in the vocal tract (as for unvoiced fricatives) and another at the glottis caused by the vibration of the vocal cords. Fricative sounds generate a broad continuous spectrum.

### 3.4.3 **Plosive Sounds**

Plosive sounds, also known as stop consonants, are produced by making a complete closure in the vocal tract (usually near the top), building up pressure behind the closure and abruptly releasing it. Plosives are also either voiced or unvoiced sounds. The difference between the unvoiced plosives, /p/, /t/ and /k/, and their voiced counterparts /b/, /d/ and /g/ is that during the period of closure of the tract, the vocal cords vibrate for the voiced sounds.

## 3.5 A Model of the Speech Production Mechanism

This chapter has attempted to show something of the nature of the speech production mechanism. To sum up, speech is a time-varying sound wave whose form at any given time is dependent on the current shape of the vocal tract and also on the mode of excitation of the vocal tract. From this it is possible to define a simplified model of this speech production mechanism.

To produce a speech-like signal the mode of excitation and the resonance properties of the linear system must change with time. For most speech sounds the general properties of the excitation and vocal tract remain fixed for periods of 10-20ms. Thus, the model involves a slowly time-varying linear system excited by a signal that changes from a periodic nature for voiced speech to white noise for unvoiced speech.

An all-pole model has been shown to be a very good representation of the vocal tract effects for the majority of speech sounds [11]. Although the generation of nasals and fricatives requires both resonances and anti-resonances (poles and zeros), Atal [12] has shown that the effect of a zero of the transfer function can be achieved by including more poles.

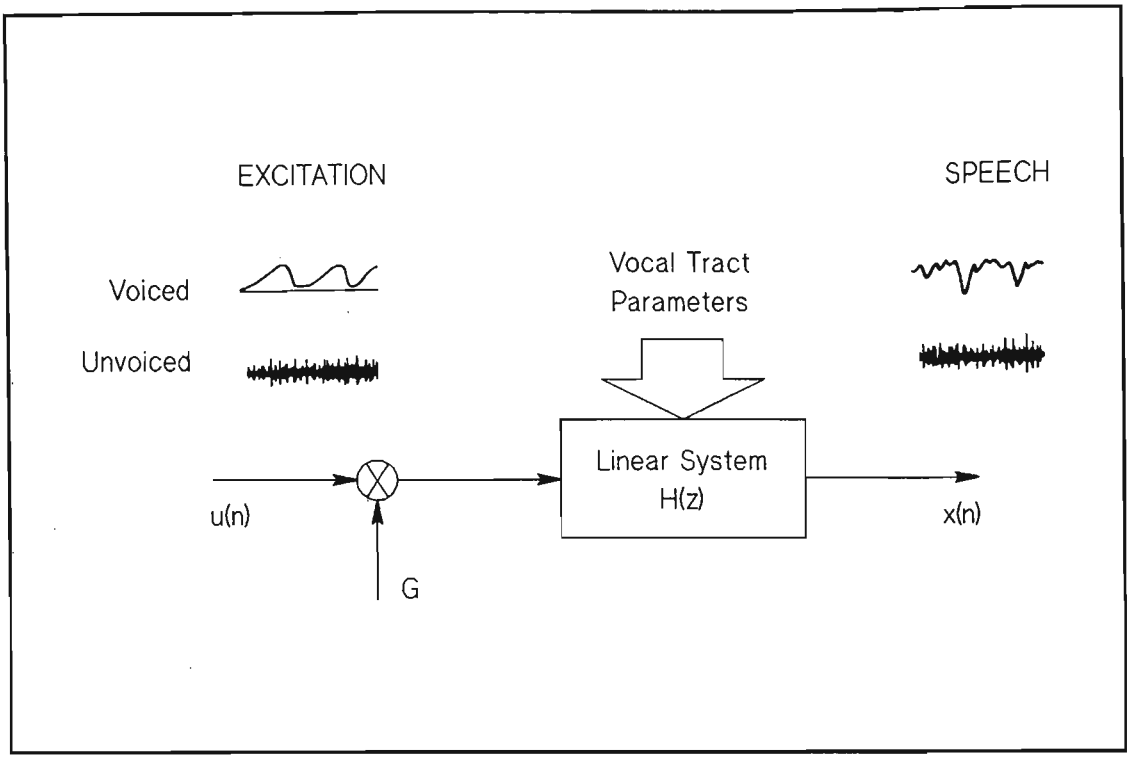



**Fig. 3.2 Basic model of the Speech Production Mechanism** [11,13].

The model illustrated in Fig. 3.2 is an all-pole model and the poles of the transfer function H(z) correspond to the resonances (formants) of speech. The input and output in the above model can be represented by a transfer function, H(z), of the form

$$H(z) = \frac{G}{1 - \sum_{k=1}^{N} a_k z^{-k}} \qquad \ldots(3.1)$$

where G and $\{a_k\}$ depend upon the shape of the vocal tract.

This model forms the basis for the discussion on the technique of Linear Predictive Coding in section 4.11.

# Chapter 4

## SPEECH RECOGNITION TECHNIQUES

### 4.1 Introduction

As already stated, speech recognition systems differ principally in the features that they extract and in the pattern comparison algorithms that they use. This chapter describes most speech recognition techniques that have been implemented for isolated word systems. A discussion of pattern comparison and the associated topic of word boundary detection concludes the chapter.

Techniques used in speech recognition systems generally include the following: time-domain methods, frequency-domain methods, stochastic models, linear predictive coding techniques, or some combination of these methods. Time-domain speech processing methods involve the waveform of the speech signal directly. Frequency-domain techniques, on the other hand, involve (either explicitly or implicitly) some form of spectrum representation. The third method uses models to generate sets of features representing the reference words. Linear predictive coding is a technique that generates an all-pole model of the short-time speech spectrum and leads to a simple analysis procedure in the time domain. These techniques are all explained in more detail later in this chapter.

## 4.2 The Concept of Short-time Processing

The underlying assumption in most speech processing schemes is that the properties of the speech signal change relatively slowly with time. This allows short segments of the speech signal (typically 10-20 ms) to be processed as if they were short segments from a sustained sound with periodic properties. Such analysis of a finite segment of a speech signal is known as short-time analysis and the segments are usually referred to as analysis frames.

In order to isolate a short segment of the signal, the signal is multiplied by a window function w(n).

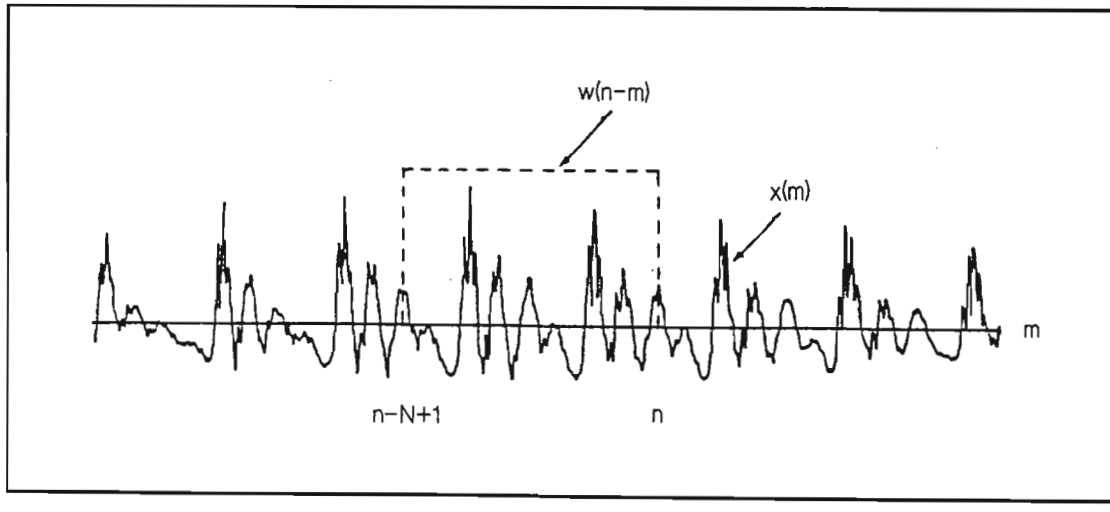



**Fig. 4.1 Rectangular Window of N Samples**

## 4.3 **Short-time Energy**

Since the amplitude of a speech signal varies appreciably with time, the short-time energy of the signal provides a convenient representation that reflects these amplitude variations. The short-time energy of a frame of N samples is defined as

$$E_n = \sum_{n=0}^{N-1} [x(n).w(n)]^2 \qquad ...(4.1)$$

This expression can be written as

$$E_n = \sum_{n=0}^{N-1} x^2(n).h(n) \qquad ...(4.2)$$

where $h(n) = w^2(n)$

That is, the signal $x^2(n)$ is filtered by a linear filter with impulse response h(n).

The length of the window function determines the nature of the short-time energy representation. If the window is very long then $E_n$ will change very little with time. If the window is too short, however, it will not provide sufficient averaging to produce a smooth energy function. What is needed is a window of short duration that will reflect rapid changes in amplitude. Window functions are dealt with in more detail in section 4.9.3.

## 4.4 Short-time Average Zero-Crossing Rate

The number of zero-crossings in a frame simply refers to the number of times that the signal changes polarity (crosses the zero axis). The rate at which zero-crossings occur is a simple measure of the frequency content of a signal. If the sampling frequency is $f_s$, the number of samples per frame is N, and the zero-crossing count is Z, then the signal frequency, f, is

$$f = f_s \cdot \frac{Z}{2.N} \qquad \ldots(4.3)$$

Since speech signals are broadband signals, the interpretation of an average zero-crossing rate is obviously much more meaningful if the signal has first been passed through a number of bandpass filters in order to get a series of outputs of narrower bandwidth.

Even without the use of bandpass filters, a short-time zero-crossing rate contains valuable information. Since the energy of voiced speech is generally concentrated below 3kHz, whereas for unvoiced speech most of the energy is found at higher frequencies, zero-crossing rates can be used to distinguish between these two broad categories of speech sound.

A certain degree of care must be taken in the sampling process when measuring zero-crossing rate. This is due to the fact that the zero-crossing rate is strongly affected by DC offset in the analog-to-digital converter, 50Hz mains hum in the signal, and any noise that may be present in the digitizing system.

## 4.5 Short-time Autocorrelation Function

The short-time autocorrelation function for a frame of N samples is defined mathematically as

$$R_n(k) = \sum_{n=0}^{N-1-k} [x(n)w(n)]\,[x(n+k)w(n+k)] \qquad \ldots(4.4)$$

where w(n) is the window function.

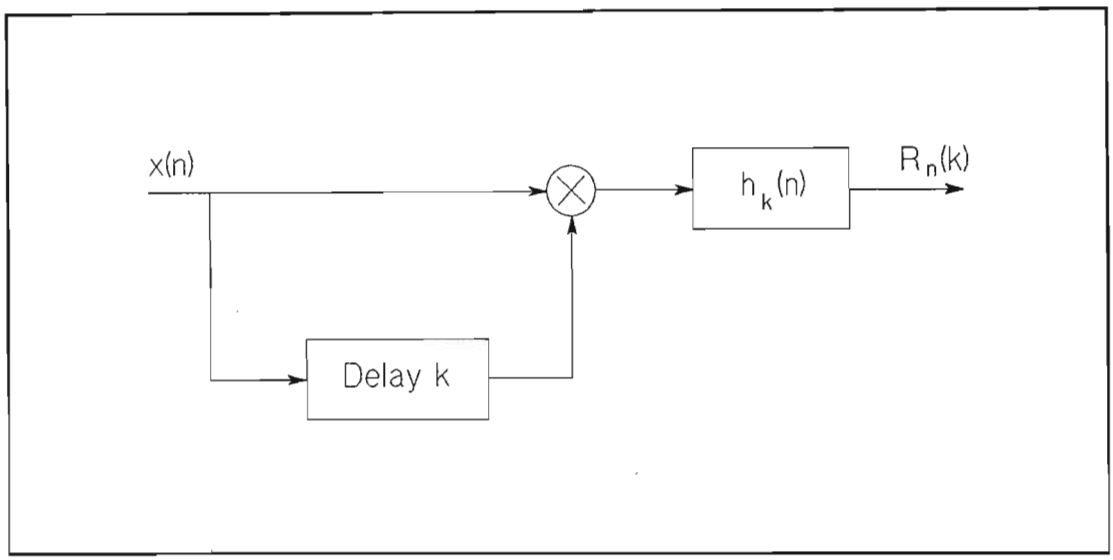



**Fig. 4.2 Block Diagram Representation of the Short-Time Autocorrelation Function**

Since for a periodic signal of period P, x(n) = x(n+P), the autocorrelation function as defined above attains a maximum at samples 0, P, 2P,... Therefore the period of a signal can be estimated by finding the location of the first maximum in the autocorrelation function. That is, finding the first non-zero value of k that maximizes $R_n(k)$ in equation 4.4.

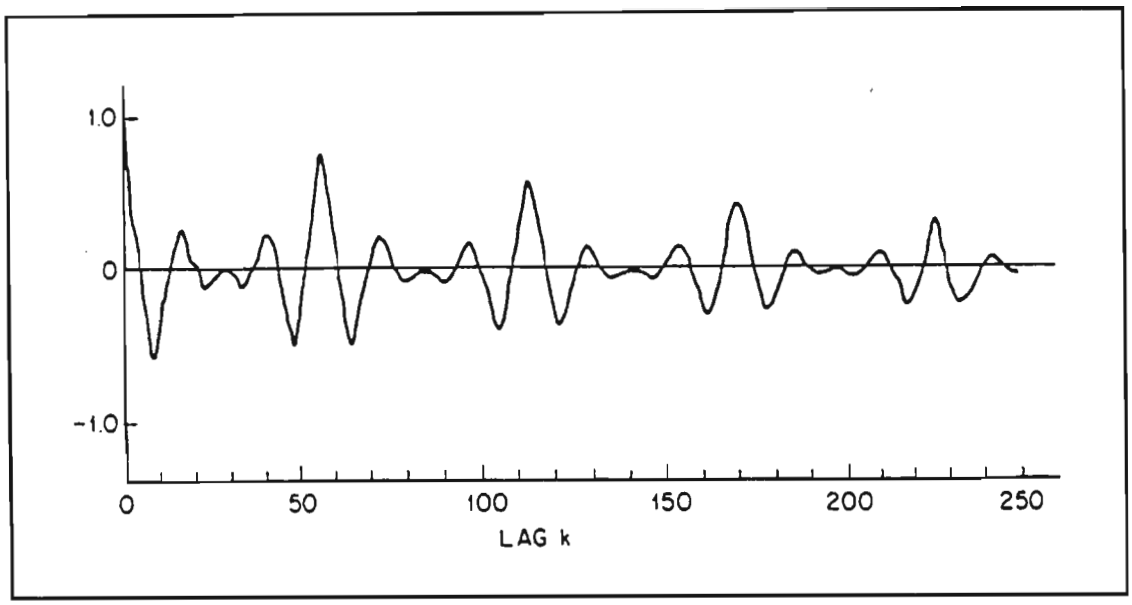



**Fig. 4.3 Autocorrelation function for voiced speech using a rectangular window with N = 400.** [11]

Figure 4.3 above shows an example of the autocorrelation function computed (using equation 4.5) for a 400 sample frame of speech sampled at 10kHz. The autocorrelation was evaluated for lags $0 < k < 250$. Note that the peaks occur at multiples of about 58 indicating a period of 5.8 ms or a fundamental frequency of approximately 170 Hz.

Because of the nature of the autocorrelation function as defined in equation 4.4, it should be obvious that the tapering effects of a Hamming window will not improve the results of the autocorrelation function. Best results are obtained using a rectangular window which does not taper the signal's magnitude at the two ends of the analysis frame.

Looking at equation 4.4, it is apparent that the calculation of the $k^{th}$ autocorrelation lag requires $2(N-k)$ multiplications and $(N-k)$ additions. Although this can be reduced by taking advantage of some special properties of the equation, it is still computationally very severe.

## 4.6 Pitch Period Estimation

A pitch detector provides an insight into the nature of the excitation source for speech production. It is therefore more useful for recognizing speakers than recognizing words spoken by a particular speaker. It does, however, find application in speaker-independent speech recognition systems, and for this reason is discussed briefly.

One of the major problems in pitch detection is that although there is generally a fairly prominent peak (in the frequency domain) at the pitch period, there are also peaks due to the formant frequencies of the signal. Thus for reliable pitch detection, it is highly desirable to greatly reduce, or entirely eliminate, the effects of the formants. The technique of removing the spectral shaping in the waveform due to the formants is known as spectral flattening.

To partly eliminate the effects of the higher formants on the signal spectrum, most methods of pitch detection use a low-pass filter with a sharp cut-off around 900 Hz. This will, in general, preserve a sufficient number of pitch harmonics for accurate pitch detection, but will eliminate the second and higher formants [14].

A wide variety of methods have been proposed for spectrally flattening the effects of the first formant, the most commonly used being the process of centre clipping. Here, signal values below the clipping level are set to zero, whereas those above the clipping level are offset by the clipping level. It can be seen from figure 4.4 that if the clipping level is appropriately set, most of the waveform structure due to the formants can be totally eliminated.

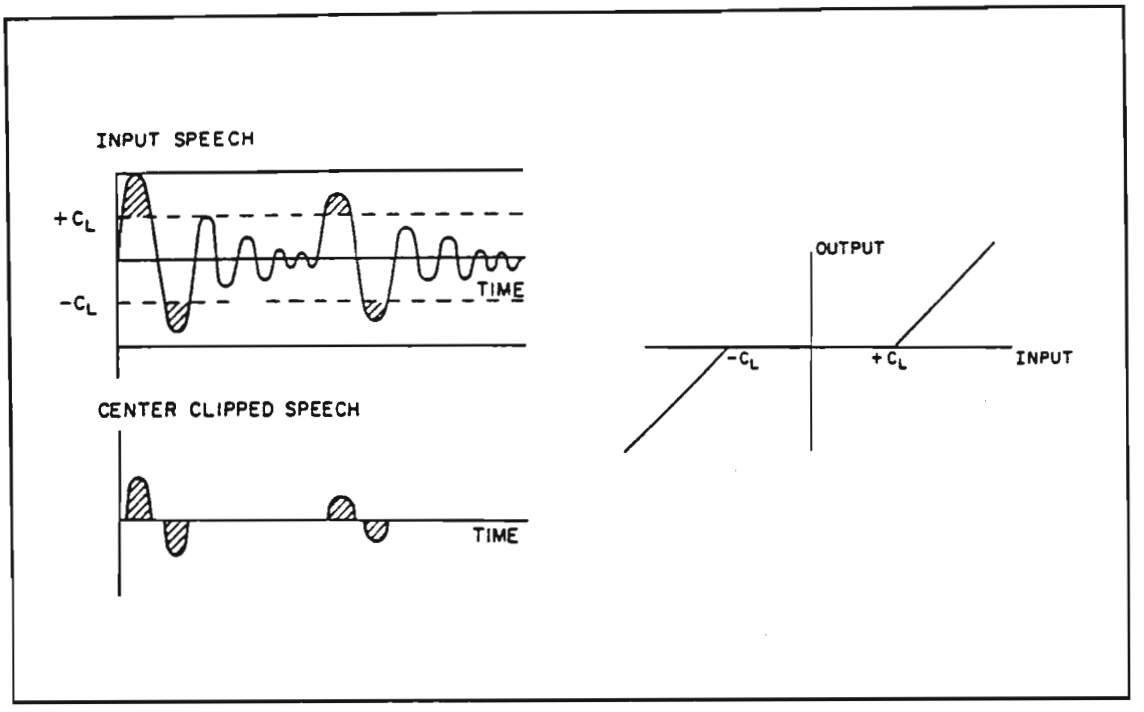



**Fig. 4.4 Input-output characteristic and typical operation of a centre clipper [15].**

Although several techniques have been developed for determining the pitch period from the spectrally flattened signal, the autocorrelation methods have generally met with the most success [16].

## 4.7 **Pre-emphasis**

There is an overall -6dB/octave trend in speech as frequency increases [17]. In other words the signal power is reduced by a factor of 4 for each doubling of frequency. A +6dB/octave lift prior to processing the speech signal is known as pre-emphasis. Extensive experimental evidence has shown that pre-emphasis also serves to reduce the variance of the distance caculations [18].

This effect can be achieved by a filter with a transfer function

$$H(z) = 1 - az^{-1} \qquad \ldots(4.5)$$

where the constant parameter 'a' typically has a value of between 0.9 and 1. This may be represented in the time domain as

$$y(n) = x(n) - ax(n-1) \qquad \ldots(4.6)$$

Figure 4.5 shows the frequency response for the above operation with a = 0.95 and a sampling frequency of 10kHz.

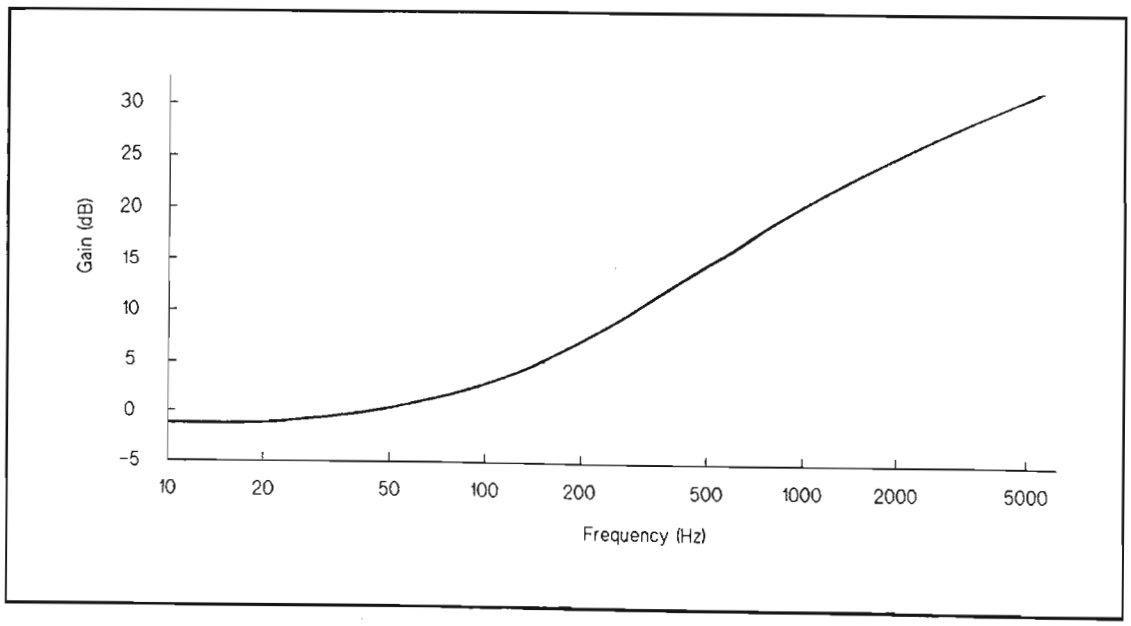



**Fig. 4.5 Frequency Response due to Pre-emphasis**

Pre-emphasis is really only of use when applied to voiced sounds and there is no need for it in unvoiced speech. The parameter 'a' can be forced to be close to 1 for voiced speech and close to 0 for unvoiced speech by expressing it in terms of the autocorrelation of the incoming signal, as

$$a = R(1) / R(0) \qquad \ldots(4.7)$$

where R(1) and R(0) are correlations of the signal with itself delayed by one sample and with no delay respectively. It should be obvious that the high sample-to-sample correlation in voiced speech results in R(1) not being much smaller than R(0) and hence 'a' is close to 1. On the other hand, little or no sample-to-sample correlation in unvoiced speech results in the ratio being close to zero.

## 4.8 Statistical Models for Speech Recognition

Practically all speech recognition methods exploit the fact that repeated utterances of the same word usually exhibit similar acoustic patterns. It is normal for some parts of a pattern to vary more from occurrence to occurrence than other parts. It may so happen, however, that the value of a particular feature may be quite critical at a particular frame of one word, and yet it may be of little significance in some part of a different word.

It is possible, therefore, that the performance of a speech recognition system can be improved by taking into account the variability of the pattern. The matching of a particular word should not be penalized if the parts that match badly are parts which are known to vary extensively from utterance to utterance.

All speech recognition systems that are based on a process of feature matching choose the best matching word on the basis of finding the template which gives the lowest cumulative distance between the frames of the test and reference patterns. The approach taken here, however, is to define some **model** which can generate patterns of features to represent the word. Every time the model for a particular word is activated, it will produce a set of feature vectors that will represent an example of the word it defines. The best matching word is then chosen to be that word whose model is most likely to produce the set of feature vectors observed for some unknown utterance. It is therefore not a 'distance' that is calculated, but a probability that the model of a reference word could have produced the observed sequence of feature vectors.

Such a model that is governed only by a set of probabilities is known as a stochastic process. A further property of the model is that at any time, it's possible actions depend only on it's current state and not on any previous sequence of events. This classifies the model as a first-order Markov process and, because the sequence of feature vectors are modeled as a probabilistic function of an underlying process whose transitions are not directly observable, the model is usually referred to as a hidden Markov model (HMM).

HMM methods are computationally more efficient than techniques such as dynamic time warping (discussed in section 4.14.2). Although their performance is slightly inferior [32], they have been widely used in commercial systems where the time taken to recognize a word is of utmost importance.

## 4.9 Fourier Analysis of Speech

Fourier analysis is widely used in signal processing and the theory is assumed known. Only it's relevance to speech recognition is discussed here.

### 4.9.1 The Discrete Fourier Transform

In speech analysis it is the discrete Fourier transform (DFT) - the Fourier transform of a finite-length sequence - that is of importance. A sequence x(n) of length N that is zero outside the interval $0 \leq n \leq N-1$ can be expressed exactly by a discrete Fourier transform of the form

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-jwkn} \qquad k = 0,1,..,N-1 \qquad ...(4.8)$$

$$x(n) = 1/N \sum_{n=0}^{N-1} X(k)\, e^{jwkn} \qquad n = 0,1,..,N-1 \qquad ...(4.9)$$

where $w = 2\pi/N$

A fast Fourier transform (FFT) is invariably used in the computation of the DFT of a sequence. An FFT is nothing more than an algorithm for evaluating the DFT in a faster, more efficient manner than by evaluating it directly.

### 4.9.2 **Short-time Fourier Transform**

Short-time Fourier analysis deals with finite-length, time-varying sequences and it is especially applicable to speech since the speech waveform varies relatively slowly with time. A window function can be used to isolate a short segment of the speech signal. It is useful to define the DFT in terms of a window function w(n-m).

If x(m) is a long time-varying speech sequence and the short section (or frame) to be analyzed is located at a particular time index n, then the time-varying Fourier transform may be defined as

$$X_n(e^{jw}) = \sum_{m=-\infty}^{\infty} x(n)\ w(n-m)\ e^{-jwm} \qquad \ldots(4.10)$$

This is the Fourier transform of the windowed sequence x(n)w(n-m).

### 4.9.3 **Window Functions and their Properties**

Window functions are not usually rectangular in shape. Remember that the set of samples in a frame is regarded as one period of a periodic signal. Usually, however, the first and last samples of the window do not 'join up' as they would in a real periodic signal. This can lead to inaccuracies in the computed spectrum.

The problem may be reduced by tapering the set of samples so that the ends are effectively forced to join up. A widely used windowing function with this tapering characteristic is the **Hamming window** [13] which is defined as

$$\begin{aligned} w(n) &= 0.54 - 0.46\cos(2\pi n/(N-1)) && 0 \le n \le N-1 \\ &= 0 && \text{otherwise} \end{aligned} \quad \ldots(4.11)$$

Taking the Fourier transforms of the rectangular and Hamming window functions, it can be shown [11] that the bandwidth of a Hamming window is about twice the bandwidth of a rectangular window of the same length. Further, the Hamming window gives much greater attenuation outside the passband than the comparable rectangular window.

It is fairly obvious that the frequency resolution of any analysis will be adversely affected if the window duration is less than one pitch period. At a sampling rate of 10kHz, the pitch period of a high pitched female voice can be as low as 25 samples and that of a very low pitched male voice can be as high as 250 samples [11]. Accepting that these are the extremes, a practical choice of frame length is 100-200 samples (ie. 10-20 ms duration).

## 4.10 Filter-bank Analysis

### 4.10.1 Description

Another method used in speech recognition splits the speech signal into its frequency components by means of a bank of filters. The analysis can be provided by a set of band-pass filters whose range of centre frequencies covers the frequencies most important for speech perception (300 to 5000 Hz).

A suitable distance measure used to compare two words is the sum of the squared differences between the logarithms of the power levels in the corresponding channels. (This is the square of the Euclidean distance).

### 4.10.2 Implementation

One method is to use a bank of analogue filters to acquire the required frequency components. Since the outputs of each filter must be individually sampled to allow further digital processing of the signals, this method is not particularly attractive. Another method is to use digital filters to do the necessary frequency separation. The most convenient method, however, is to use the outputs from short-time Fourier transforms. This analogy may be justified as follows:

Equation 4.10 can be rewritten as

$$X_n(e^{jw}) = e^{-jwn} \sum_{m=-\infty}^{\infty} x(n-m)\ w(m)\ e^{jwm} \qquad \ldots(4.12)$$

where w(m) is again the window function used.

If we define $h(n) = w(n)\ e^{jwn}$ ...(4.13)

then equation 4.12 can be expressed as

$$X_n(e^{jw}) = e^{-jwn} \sum_{m=-\infty}^{\infty} x(n-m)\ h(m) \qquad ...(4.14)$$

If we now define $y(n) = X_n(e^{jw})\ e^{jwn}$ ...(4.15)

then from equation 4.14 we see that

$$y(n) = \sum_{m=-\infty}^{\infty} x(n-m)\ h(m) \qquad ...(4.16)$$

Thus y(n) is simply the output of a bandpass filter with impulse response h(n) as given by equation 4.13.

A set of bandpass filters can be implemented by choosing the centre frequencies $w_k$ so that they are uniformly spaced over the entire frequency band to be analyzed. Thus for N channels

$$w_k = \frac{2\pi k}{N} \qquad k = 0,1,...N-1 \qquad ...(4.17)$$

## 4.11 Linear Predictive Coding

### 4.11.1 Introduction

Linear predictive analysis, or linear predictive coding (LPC), is one of the most powerful speech analysis techniques. It provides extremely accurate estimates of speech parameters such as pitch, formants and spectra [11]. Linear predictive coding of speech is a technique that uses the fact that any speech sample x(n) can be represented as a linear combination of its past outputs and its past and present inputs. If u(n) is some unknown input, then this can be represented mathematically as

$$x(n) = \sum_{k=1}^{p} a_k\, x(n-k) + G \sum_{l=0}^{q} b_l\, u(n-l) \qquad \ldots(4.18)$$

where $a_k$, $b_l$, k, l and the gain G are the parameters of this speech production model.

Equation 4.18 can also be specified in the frequency domain by taking the Z-transform of it. If H(z) is the transfer function of the system then

$$H(z) = \frac{X(z)}{U(z)} = G\, \frac{1 - \sum_{l=1}^{q} b_l z^{-l}}{1 - \sum_{k=1}^{p} a_k z^{-k}} \qquad \ldots(4.19)$$

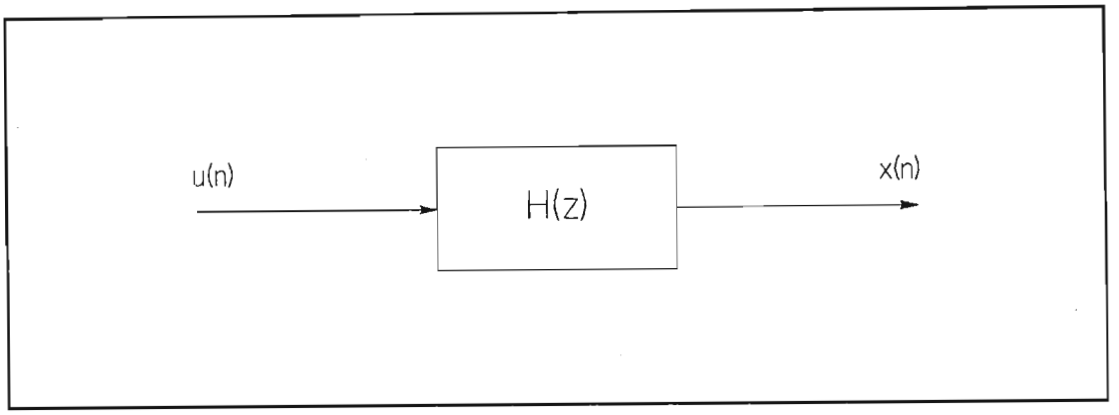



**Fig. 4.6 Simplified Speech Model in Frequency Domain**

Two special cases of this model, where H(z) is as defined in equation 4.19, are of interest:

(i) all-pole model ($b_l = 0$) - autoregressive model
(ii) all-zero model ($a_k = 0$) - moving average model

A short-time spectral representation does, in general, include contributions from both poles and zeros. Because an all-pole model can approximate the important effects of zeros on the spectrum [19], practically all linear predictive coding techniques use the all-pole model.

### 4.11.2 All-pole Linear Prediction Model

#### 4.11.2.1 Model Definition

In the all-pole model, it is assumed that the signal x(n) may be given as a linear combination of past outputs and the present input u(n).

$$x(n) = \sum_{k=1}^{p} a_k x(n-k) + G\,u(n) \qquad \ldots(4.20)$$

where $a_k$ and the gain G are the parameters of this model which is shown in the time domain in figure 4.7.

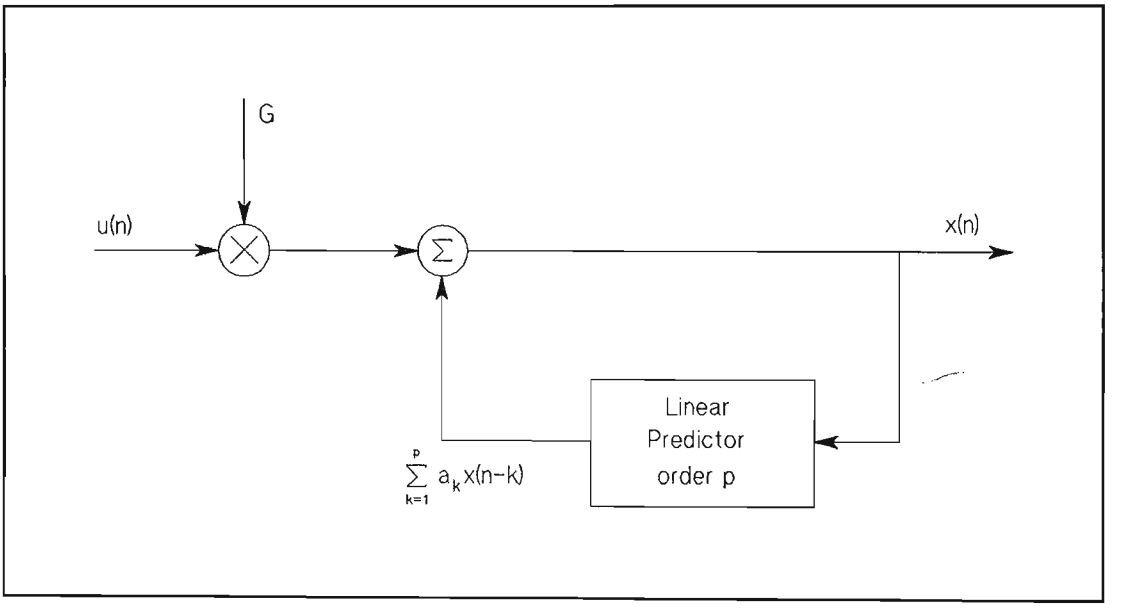



**Fig. 4.7 Discrete all-pole model in the time domain**

The transfer function H(z) for the all-pole model is therefore

$$H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} \qquad \ldots(4.21)$$

Thus the linear filter is completely specified by a gain factor G and a set of p coefficients $\{a_1, a_2, \ldots, a_p\}$ known as the linear predictive coefficients (p is also the order of the filter).

The major advantage of this model is that the filter coefficients $\{a_k\}$ can be estimated in a very straightforward and computationally efficient manner. For nasal and fricative sounds, where both poles and zeros are necessary in the vocal tract transfer function, by making the order p high enough, the all-pole model provides a good representation of the speech spectrum [11].

Given a particular signal x(n), the problem is to determine the predictor coefficients $a_k$ and the gain G in some manner. Since only the predictor coefficients are used in speech recognition algorithms based on linear predictive coding, the computation of the gain constant G is not described here.

#### 4.11.2.2 **Determining the Predictor Coefficients**

In the case of speech recognition, the input signal u(n) is totally unknown and therefore the output signal x(n) can be predicted only approximately from a linear combination of past samples (outputs). This approximation x(n) is defined as

$$x(n) = \sum_{k=1}^{p} a_k x(n-k) \qquad \ldots(4.22)$$

The error between the actual signal value x(n) and the predicted value x(n) is then defined as

$$\begin{aligned} e(n) &= x(n) - x(n) \\ &= x(n) - \sum_{k=1}^{p} a_k x(n-k) \qquad \ldots(4.23) \end{aligned}$$

or $$e(n) = x(n) + a(1)x(n-1) + \ldots + a(p)x(n-p) \qquad \ldots(4.24)$$

where $a(k) = -a_k$

The constants $\{a_k\}$ are the linear prediction coefficients and e(n) is sometimes called the prediction residual. It is obvious when comparing equations 4.20 and 4.23 that $e(n) = Gu(n)$.

The total squared error is denoted by E, where

$$E = \sum_{n=n_0}^{n_1} e(n)^2 = \sum_{n=n_0}^{n_1} [x(n) - \sum_{k=1}^{p} a_k x(n-k)]^2 \qquad \ldots(4.25)$$

The basic approach is to find a set of predictor coefficients $\{a_1, a_2, \ldots a_p\}$ that will minimize the above error. E is minimized by setting

$$\frac{\delta E}{\delta a_k} = 0 \qquad \ldots(4.26)$$

thereby obtaining the equations

$$\sum_{k=1}^{p} a_k \sum_{n=n_0}^{n_1} x(n-k)x(n-i) = \sum_{n=n_0}^{n_1} x(n)x(n-i) \qquad \ldots(4.27)$$

$$1 \leq i \leq p$$

Two distinct methods for the estimation of the parameters are obtained by specifying the range of the summation over n.

#### 4.11.2.2.1 **Autocorrelation Method**

This method requires the signal to be set to zero outside the analysis interval and that a suitable window, such as a Hamming window, be used to reduce the abrupt change in signal values occurring at the beginning and end of the analysis interval. Such a window is necessary since otherwise the prediction error is likely to be large at the beginning of the interval because we are trying to predict the signal from samples that have arbitrarily been set to zero. Likewise at the end we are trying to predict zero from samples that are non-zero.

In the autocorrelation method we have $x(n)=0$ everywhere outside of the interval $0 \leq n \leq N-1$ and the limits $n_0$ and $n_1$ may thus be defined as $-\infty$ and $\infty$ respectively. Now the autocorrelation function of a signal $x(n)$ is

$$R(i) = \sum_{n=-\infty}^{\infty} x(n)x(n+i) \qquad \ldots(4.28)$$

Equation 4.27 can now be rewritten as

$$\sum_{k=1}^{p} a_k R(i-k) = R(i) \qquad 1 \leq i \leq p \qquad \ldots(4.29)$$

The coefficients $R(i-k)$ form what is often known as an autocorrelation matrix. This is a symmetric Toeplitz matrix (one where all the elements along each diagonal are equal). This special property is exploited in algorithms used to determine the LPC coefficients $a_k$ in equation 4.29.

Because the signal x(n) is multiplied by a window function w(n) to obtain a signal that is zero outside some interval $0 \leq n \leq N-1$, the autocorrelation function (equation 4.28) may in fact be written as

$$R(i) = \sum_{n=0}^{N-1-i} x(n)x(n+i) \qquad 1 \leq i \leq N \qquad \ldots(4.30)$$

Only the values of R(i) for i = 1, 2, ..., p are needed for the solution.

#### 4.11.2.2.2 **Covariance Method**

In contrast with the autocorrelation method, here we assume that the error E in equation 4.25 is minimized over a finite interval. The limits $n_0$ and $n_1$ are defined so that no data points {x(n)} outside of the range $0 \leq n \leq N-1$ are needed for calculation purposes. Equation 4.27 then reduces to

$$\sum_{k=1}^{p} a_k \phi(i,k) = \phi(i,0) \qquad 1 \leq i \leq p \qquad \ldots(4.31)$$

where

$$\phi(i,k) = \sum_{n=p}^{N-1} x(n-i)x(n-k) \qquad \ldots(4.32)$$

is the covariance of the signal x(n) in the given interval. The coefficients $\phi(i,k)$ in equation 4.32 form a covariance matrix. It will be a symmetric, positive semi-definite matrix and will be singular if the input data sequence {x(n)} satisfies a linear homogeneous difference equation of order p or less.

#### 4.11.2.2.3 **Lattice Methods**

Both the covariance and autocorrelation methods require the computation of a correlation matrix and the solution of a set of linear equations. A third class of methods, called lattice methods, has, in a sense, combined these two processes into a recursive algorithm for determining the linear predictor parameters. The lattice method is, however, the least computationally efficient method for solving the LPC equation. It was mentioned here for completeness and is not discussed in any detail.

### 4.11.3 **Frequency Domain Interpretation of LPC**

The discussion of linear predictive analysis has thus far been mainly in terms of time domain representations (difference equations and correlation functions). The coefficients of the linear predictor are also the coefficients of the system function that models the combined effects of vocal tract response, glottal wave shape, and radiation. Thus, from the set of predictor coefficients it is possible to determine the frequency response of the model for speech production simply by evaluating H(z) for $z = e^{jw}$. That is

$$H(e^{jw}) = \frac{G}{1 - \sum_{k=1}^{p} a_k e^{-jwk}} = \frac{G}{A(e^{jw})} \qquad \ldots(4.33)$$

If $H(e^{jw})$ is plotted as a function of frequency, then peaks will occur at the formant frequencies. Thus linear predictive analysis can be viewed as a method of short-time spectrum estimation.

To illustrate the nature of the spectral modelling capability of linear predictive spectra, figure 4.8 shows a comparison of a spectrum obtained by Fourier analysis with one obtained by linear predictive analysis - the smooth curve being the LPC spectrum. It can be seen that the LPC spectrum matches the signal spectrum much more closely near the spectrum peaks (ie. in the regions of large signal energy) than near the spectral valleys.

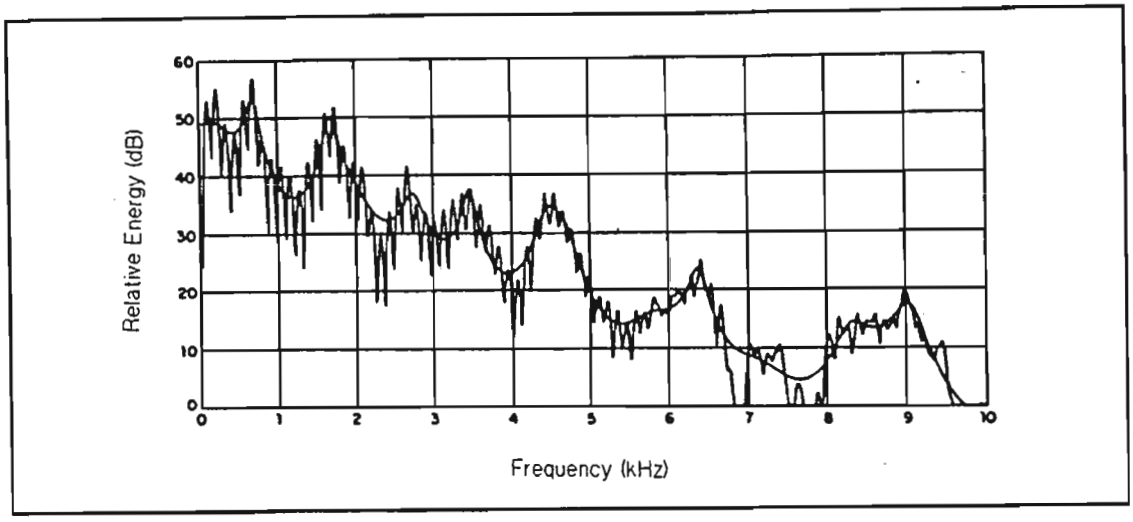



**Fig. 4.8 Illustration of 28-pole LPC spectrum fit to an FFT-computed signal spectrum.** [11]

### 4.11.4 Choosing the Order p and Frame Length N

The order p of the linear predictive analysis can effectively control the degree of smoothness of the resulting spectrum. This is illustrated in figure 4.9 which shows the Fourier transform of a speech segment and linear predictive spectra for various orders.

The choice of p depends primarily on the sampling rate and is essentially independent of the LPC method being used. The speech spectrum being analyzed can generally be represented as having as average density of 2 poles (ie. one complex pole) per kiloHertz of vocal tract contribution. Therefore for a sampling frequency of 10 kHz, a total of 10 poles are required to represent this contribution to the speech spectrum. In addition a total of 3-4 poles are required to adequately represent the source excitation spectrum and the radiation load. This is verified by Atal and Hanauer [12] in figure 4.10 which shows a plot of the normalized rms prediction error versus the predictor order p for sections of voiced and unvoiced speech sampled at

10 kHz. Although the prediction error steadily decreases as p increases, the error essentially flattens off for p of the order of 13 onwards.

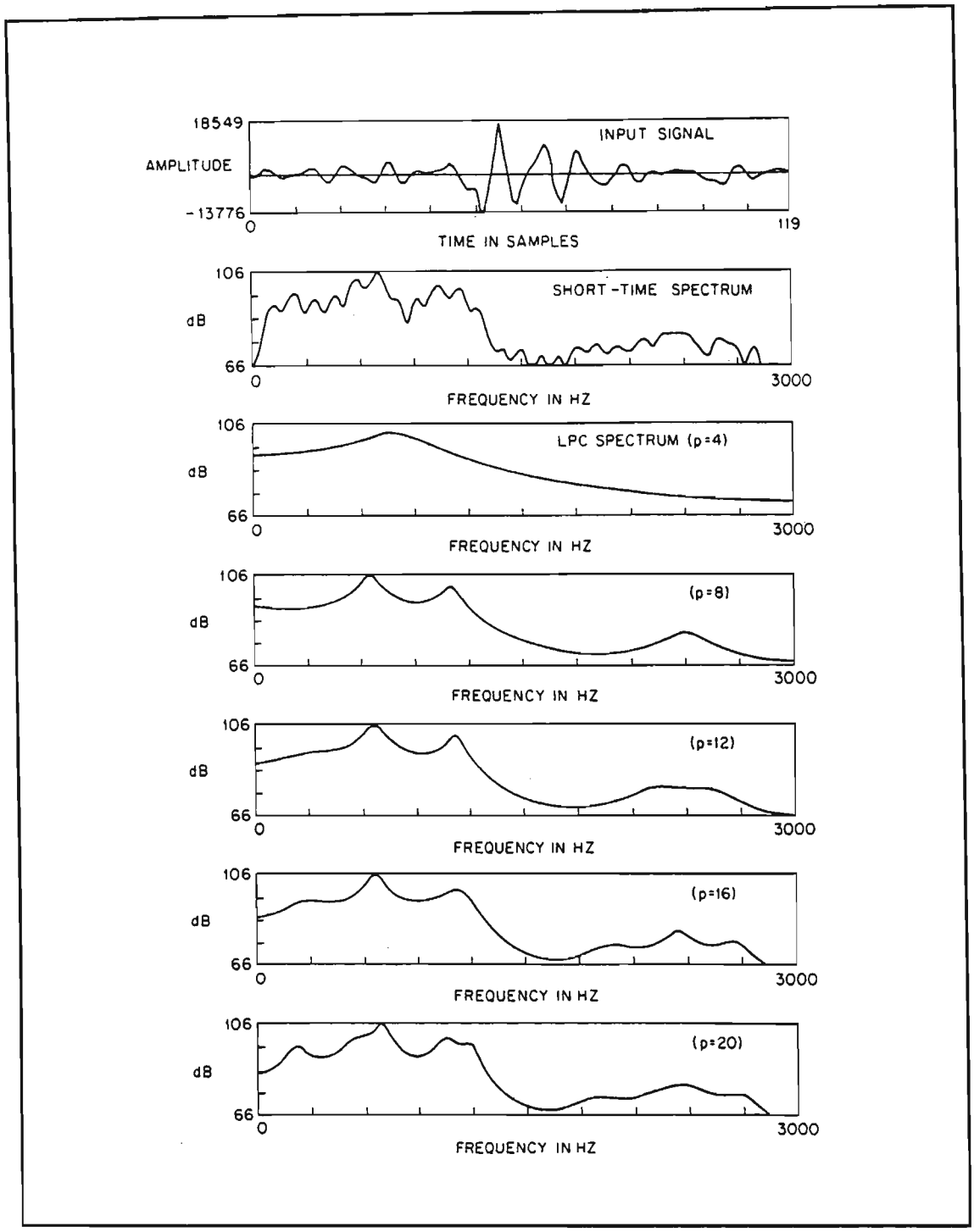



**Fig. 4.9 LPC spectra for various values of predictor order p [11].**

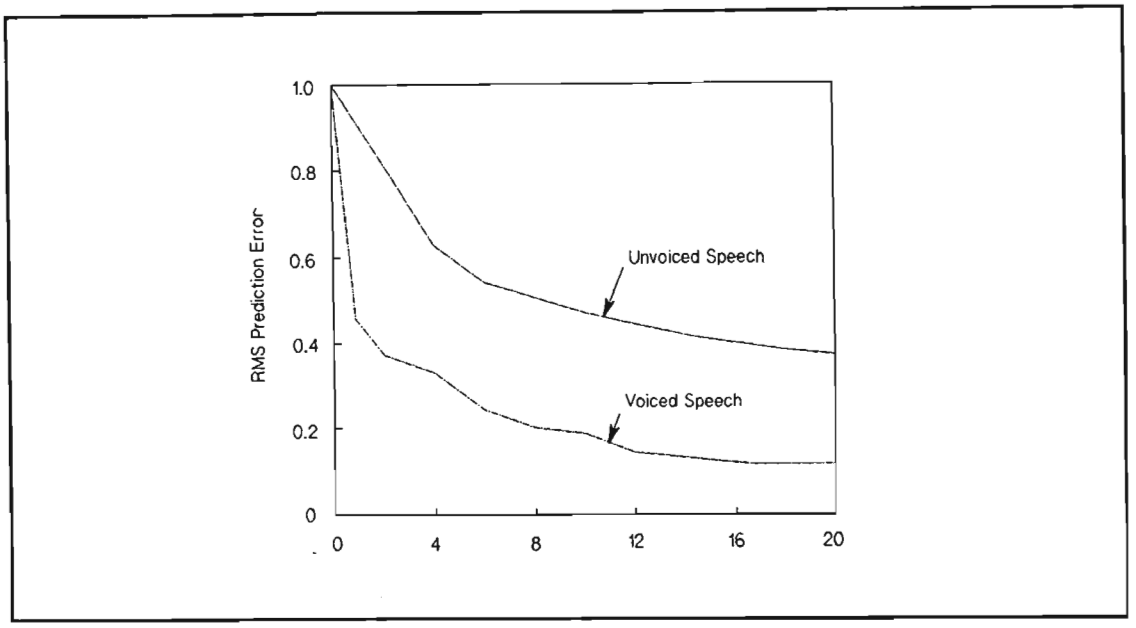



**Fig. 4.10 Variation of RMS prediction error with the number of predictor coefficients, p.** [12]

The choice of frame length N is a very important consideration in the implementation of an LPC analysis system. Since the computational load (for all three methods) is proportional to the N, it is advantageous to keep N as small as possible. For the autocorrelation method it has been shown that N must be of the order of several pitch periods to ensure reliable results. Because a window is used to weight the speech in the autocorrelation method, the frame duration must be sufficiently long so that the tapering of the window does not seriously effect the results. Analysis durations from N = 150 to N = 500 samples (at a 10 kHz rate) have been used in LPC implementations of the autocorrelation methods.

For the covariance and lattice methods, where there is no windowing required, there are no real limitations on how small the frame size must be. Generally values of N used for these two methods are comparable to those for the autocorrelation method.

### 4.11.5 **LPC Distance Measure**

A set of reference linear predictive coefficients needs to be compared to the set calculated from an unknown utterance and a similarity score given. Each reference word in the library of words to be recognized can be expressed as a time pattern of LPC coefficients, which is called a reference pattern. For each reference word an average distance between its LPC coefficients and those of the unknown utterance is computed, and the reference word whose average distance is the lowest is chosen as the spoken word. The measure of distance between LPC parameter sets is an important factor in the success of a recognition system.

Ikatura [22] defined a fairly sophisticated measure in which the total log prediction residual of an input signal is minimized by optimally registering the reference LPC coefficients onto those of the input using a dynamic programming algorithm.

The set of LPC coefficients for one frame of the reference word is defined as

$$A_r = \{1, a_r(1), a_r(2), \ldots, a_r(p)\} \qquad \ldots(4.34)$$

where p is the order of the LPC filter and $A_r$ is a row vector. Similarly the LPC coefficients of one frame of the test word are

$$A_t = \{1, a_t(1), a_t(2), \ldots, a_t(p)\} \qquad \ldots(4.35)$$

Ikatura proposed the following distance measure

$$d = \log_e \frac{A_r \, \mathbf{R}_t \, A_r^T}{A_t \, \mathbf{R}_t \, A_t^T} \qquad \ldots(4.36)$$

where

$$\mathbf{R}_t = \begin{bmatrix} r(0) & r(1) & r(2) & r(3) & \ldots & r(p) \\ r(1) & r(0) & r(1) & r(2) & \ldots & r(p-1) \\ r(2) & r(1) & r(0) & r(1) & \ldots & r(p-2) \\ r(3) & r(2) & r(1) & r(0) & \ldots & r(p-3) \\ \vdots & & & & & \vdots \\ r(p) & \ldots & \ldots & \ldots & \ldots & r(0) \end{bmatrix}$$

is an autocorrelation matrix of one frame of the test word whose elements are defined by

$$r(i) = \frac{1}{N} \sum_{n=1}^{n-i} x(n)x(n+i) \qquad \ldots(4.37)$$

With some mathematical manipulation, equation 4.36 can be rewritten in the form

$$d = \log_e (A_r A_r) + \log_e [(B_r R_t) / (A_t R_t)] \qquad \ldots(4.38)$$

where $R_t$ is a vector of autocorrelation coefficients of the test frame, (XY) represents an inner product of two vectors, and $B_r$ is a vector $\{1, b_r(1), \ldots, b_r(p)\}$ where

$$b_r(i) = 2 \sum_{j=0}^{p-i} a_r(j)a_r(j+i) / (A_r A_r) \qquad \ldots(4.39)$$

Since the distance d in equation 4.38 represents a distance between one frame of a reference word and another frame of the test word, it should be written more precisely as

$$d_k(n,m) = \log_e (A_{rk}(m)A_{rk}(m)) + \log_e [(B_{rk}(m)R_t(n)) / (A_t(n)R_t(n))] \quad ...(4.40)$$

This corresponds to the distance between the $m^{th}$ frame of the $k^{th}$ reference word and the $n^{th}$ frame of the test word. The total distance between the $k^{th}$ reference word and the test word is then given by

$$D(k) = \sum_{n=1}^{N} d_k(n, w(n)) \quad ...(4.41)$$

where w(n) is the warping function that matches the reference word onto the test word.

If during the distance computation D(k) becomes sufficiently large, then the reference pattern can be immediately rejected at an early stage without examining all the frames. The threshold for rejection should be as low as possible under the constraint that the probability of false rejection is sufficiently small.

## 4.12 **Cepstral Analysis of Speech**

A cepstrum is a spectrum of the logarithm of a frequency spectrum and it produces a smoothed version of the original frequency spectrum. The cepstrum c(n) of a segment of speech is defined as the Fourier transform of the logarithm of the Fourier transform of the input sequence.

$$c(n) = 1/2\pi \int_{\pi}^{\pi} \log|X(e^{jw})|\ e^{jwn}\ dw \qquad \ldots(4.42)$$

The cepstral coefficients c(i) can be computed recursively from the set of linear predictor coefficients a(i) by the following relations [23].

$$c(1) = -a(1)$$

$$c(i) = -a(i) - \sum_{k=1}^{i-1} (1 - k/i)\ a(k)\ c(i-k) \qquad \ldots(4.43)$$

$$1 < i \leq p$$

where p is the order of the LPC coefficients.

A squared Euclidean distance measure may be used to compare two words and is defined as

$$d_{CEP} = \sum_{i=1}^{p} (c_t(i) - c_r(i))^2 \qquad \ldots(4.44)$$

where $c_t(i)$ and $c_r(i)$ are the cepstral coefficients of the unknown test word and the reference word respectively.

## 4.13 **Endpoint Detection**

### 4.13.1 **Requirements**

Accurate location of the endpoints of an isolated word is of utmost importance for reliable recognition. A further advantage of good endpoint detection is the fact that the proper location of regions of speech can substantially reduce the amount of processing required [25].

This task of separating speech from background noise is not a trivial one except in the case of an extremely quiet environment, such as an anechoic chamber, where the energy of even the lowest level speech sound exceeds the background noise energy, making a simple energy measurement sufficient. Such ideal recordings conditions are, however, not generally found where speech recognition systems are likely to be implemented.

A system is required which is able to detect the endpoints of the spoken words in almost any environment. It is obvious then that a simple energy measurement will not suffice and that some additional information about the speech signal is needed. One possible solution is to use, in addition to the energy content, the zero-crossing rate of the signal.

### 4.13.2 **Problems Associated with Endpoint Detection**

In order to understand the difficulties associated with the detection of the endpoints of words, it is helpful to study the waveforms of various categories of sounds. The start of any utterance may generally be detected by the eye by a change in the pattern of the waveform. Figure 4.11 shows a waveform of the word 'eight' in which the speech is easily distinguished from the silence by a radical change in the waveform energy at the start of the utterance.

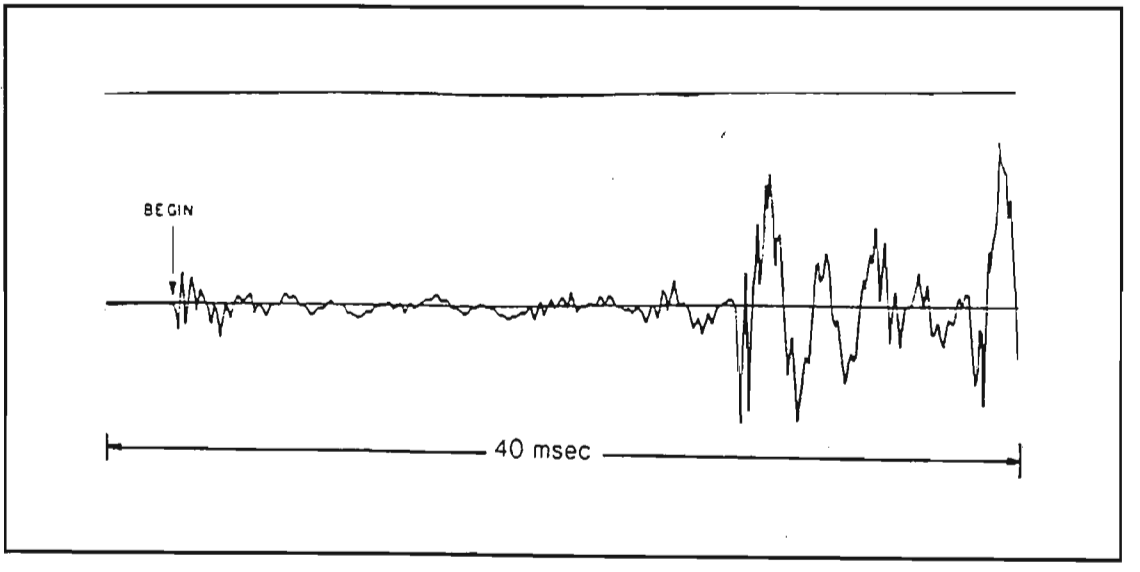



**Fig 4.11 Waveform of the beginning of the word 'eight'** [26]

Figure 4.12 shows an example of the word 'six' in which the start of the word is again easily located by the eye. In this case, however, it is the frequency and not the energy that is radically different at the start of the utterance. It is thus necessary to use the zero-crossing rate to detect the beginning of the word in this case.

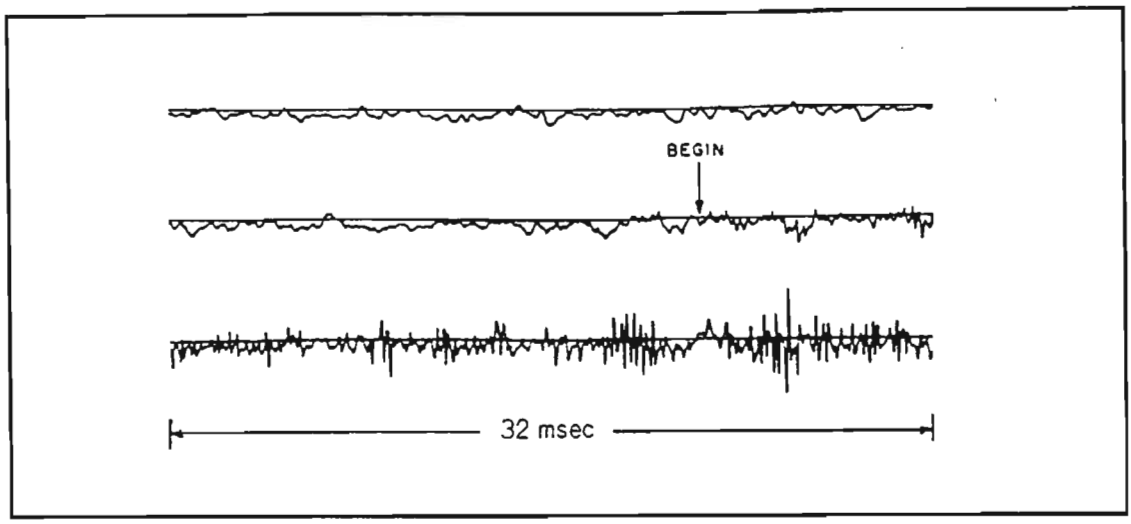



**Fig 4.12 Waveform of the beginning of the word 'six'** [26]

The above two examples are very nice in that the endpoints are every easily determined by marked changes in either energy or frequency. This is not always the case and figure 4.13 illustrates a word in which it is extremely difficult to locate the beginning of the speech signal. The eye would naturally select point B as the beginning of the utterance. This is incorrect, however, in that it completely misses the weak fricative /f/ at the beginning of the word 'four'. In fact, point A is the beginning of the word.

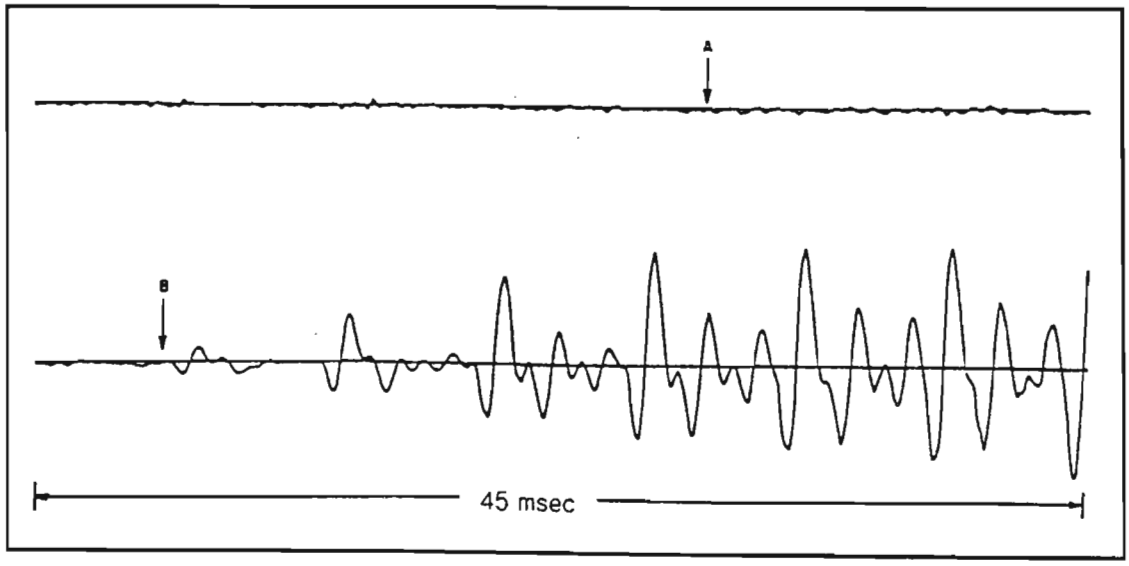



**Fig 4.13 Waveform of the beginning of the word 'four'** [26]

There are numerous other sound categories in which problems are encountered when locating the endpoints. These include [11]:

1. Weak fricatives (f, th, h)
2. Weak plosive bursts (p, t, k)
3. Nasals at the end of a word.
4. Voiced fricatives which become devoiced at the end of words.
5. Trailing off of vowel sounds at the end of an utterance.

How are these 'problem' sounds dealt with? The problem is, in fact, not as severe as it seems since it is necessary to isolate only enough of the word so that reasonable acoustic analysis of this isolated word is sufficient for accurate recognition. It is therefore not essential to know exactly where the word begins or ends, but instead it is important to include all significant acoustic events within the utterance.

The algorithm used to locate the endpoints is described in section 5.3.3.2.

## 4.14 Pattern Comparison

In all speech recognition systems using the feature-based approach, the feature vectors of the unknown word must be compared to a library of reference patterns. A distance (or similarity) score, telling how good the match was, is then calculated and the word having the lowest score is the winner.

There are various methods of pattern comparison, the simplest being one which time-aligns the reference template with the unknown test pattern in a linear fashion. The most successful pattern comparison method for speech recognition has been the so called dynamic programming or dynamic time warping algorithm [22] in which reference and test patterns are dynamically time-aligned to get the best possible match. A third and relatively new method is based on hidden Markov models.

A speaker does not, in general, speak at precisely the same rate for different repetitions of the same word. This makes the task of matching input words to reference words somewhat complicated. This difficulty may be overcome by stretching or compressing (warping) the time scale of the unknown spoken word so that it equals that of the template. This warping may be done in either a linear or non-linear fashion, with the constraint that the endpoints of the two words must match. Figure 4.14 illustrates the endpoints of two words, a reference word of M frames and a test word of N frames.

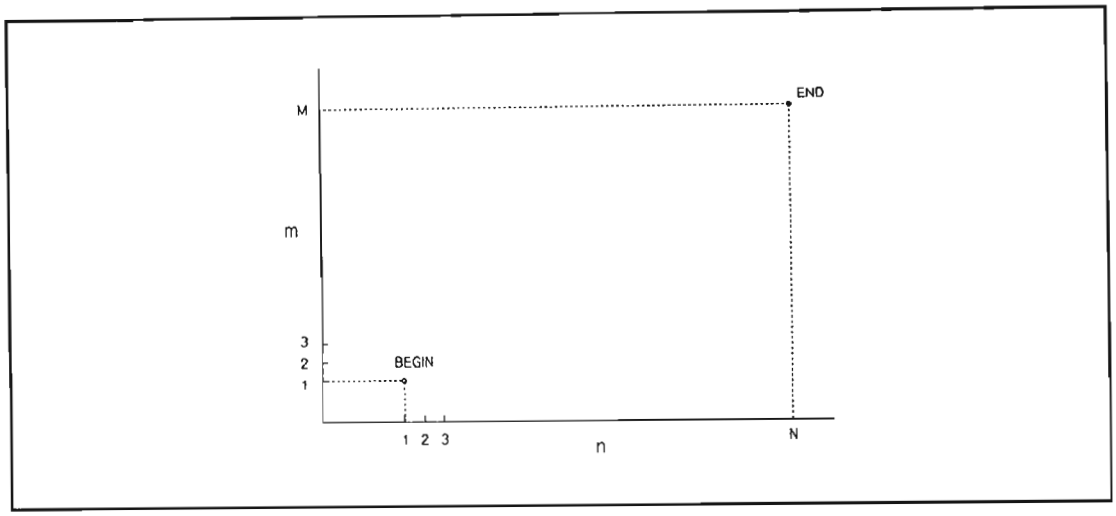



**Fig. 4.14 Endpoints of Two Words to be Compared**

An input word and a reference word are represented in figure 4.14 by the axes of a grid where n and m are the frames of the input and reference templates respectively. The two points on the grid corresponding to the end points of both templates can be seen.

### 4.14.1 **Linear Time-Alignment**

The simplest method of comparing two patterns is to take a linear path between the two endpoints. Such a path is defined by

$$m = (n-1)(M-1)/(N-1) + 1 \qquad ...(4.45)$$

Each frame n of the input word is then compared to frame m of the reference word. A distance score may then be obtained by summing the distances between each frame. Although this method does have the advantage of requiring very little computation, it does not account for any variation in the speed of the speech.

### 4.14.2 **Dynamic Time Warping**

In this method of pattern comparison, the unknown word is warped in a non-linear fashion so as to obtain the best possible match with the reference pattern. Dynamic time warping is also often referred to as dynamic programming.

Conceptually the process of time warping is as illustrated in figure 4.15. The time scale **t** of a reference utterance is warped in such a way as to line up it's significant characteristics with the same characteristics in the measured utterance. The warping function is thus of the form

$$Y = \alpha t + q(t) \qquad \ldots(4.46)$$

where q(t) is the nonlinear time warping function, and $\alpha$ the average slope of the time warping function.

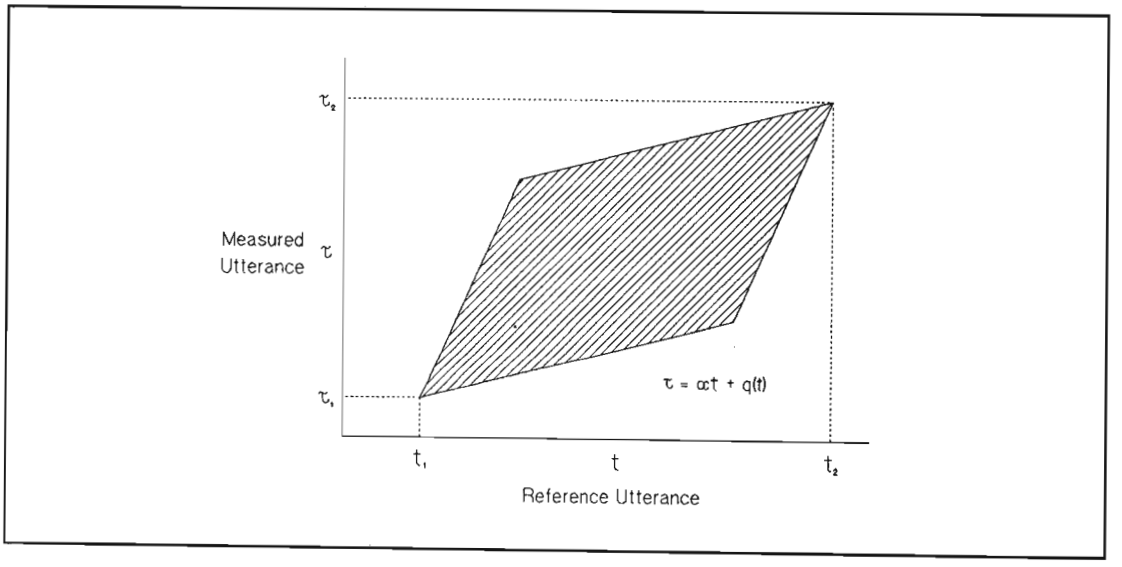



**Fig. 4.15 Illustration of Time Warping**

Time alignment is performed on speech data which is represented as a time sequence of feature vectors (eg. linear prediction coefficients) corresponding to the frames of the speech signal. This procedure requires the calculation of the local distance between each possible reference and test frame in order to determine the optimal time alignment path relating reference and test frames.

The problem, then, is to choose a path between these endpoints which will match the pairs of frames together in such a way as to optimize the distance score. In other words, we wish to choose a time warping function w such that

$$m = w(n) \qquad \ldots(4.47)$$

The boundary conditions on w(n) are:

$$\begin{aligned} w(1) &= 1 \quad \text{(beginning points)} \\ w(N) &= M \quad \text{(ending points)} \end{aligned} \qquad \ldots(4.48)$$

To limit the degree of non-linearity it is reasonable to place a constraint on the amount of allowed distortion. In order to guarantee that the average slope the warping function lies between ½ and 2, and guarantee path monotonicity it is necessary to impose the Ikatura local path constraints [22]:

$$\begin{aligned} w(n+1) - w(n) &= 0, 1 \text{ or } 2 \quad \text{if } w(n) \neq w(n-1) \\ &= 1 \text{ or } 2 \quad \text{if } w(n) = w(n-1) \end{aligned} \qquad \ldots(4.49)$$

These local constraints together with the endpoint conditions lead to the following set of global constraints that force the warping function to lie within the parallelogram in figure 4.16 with sides of slope 1:2

$$m \leq \max \{ (n-1)/2 + 1,\ M - (N-n)*2,\ 1 \}$$
$$m \geq \min \{ (n-1)*2 + 1,\ M - (N-n)/2,\ M \} \qquad ...(4.50)$$

Thus the slope of the warping function can be either 0, 1 or 2 if at the previous grid index the warped index changed, or 1 or 2 if at the previous grid index the warped index remained constant. A similarity or distance measure is used to determine the path of the warping function which locally minimizes the maximum total distance, subject to the continuity constraints of equation 4.50.

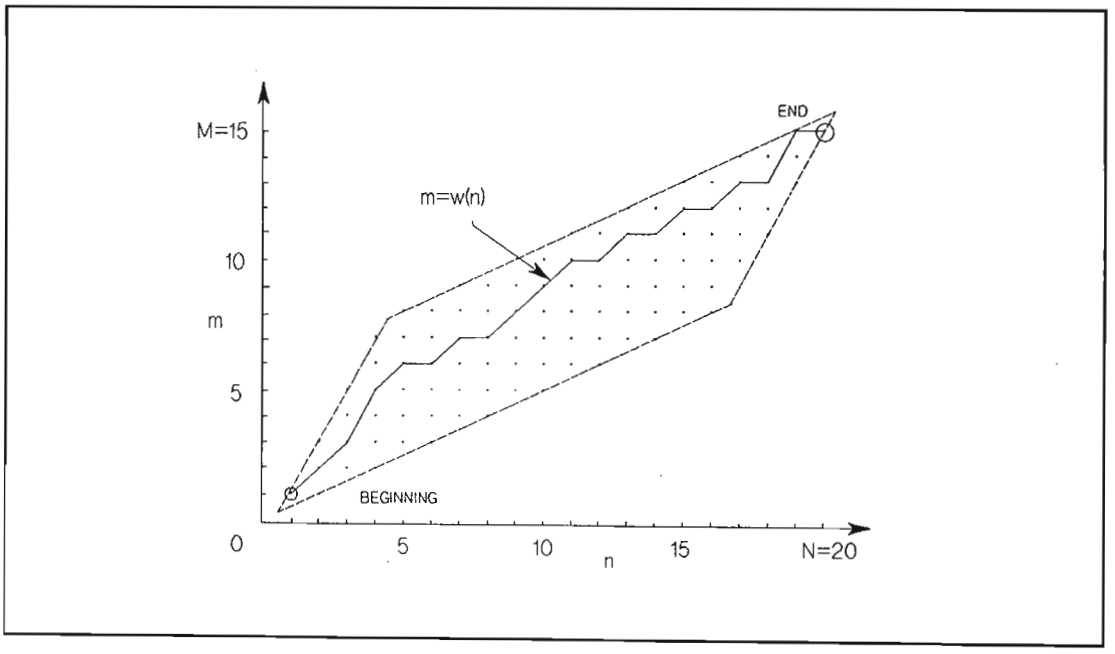



**Fig. 4.16 An Example of the Time Warping Function. The parallelogram shows the possible domain of (n,m) coordinates. [22]**

The example of figure 4.16 shows the domain of possible grid coordinates (n,m) and a typical warping function w(n) for warping a 20 point (N = 20) reference to a 15 point (M = 15) input utterance.

Let us define d(n,m;k) to be the distance between the $n^{th}$ frame of the input and the $m^{th}$ frame of the $k^{th}$ reference pattern. Denote the minimum value of the sum of d(n,m;k) for all possible choices of the time warping function by

$$D(k) = \min_{\{w(n)\}} \sum_{n=1}^{N} d(n,m;k) \qquad \ldots(4.51)$$

D(k) is a distance between the input utterance and the $k^{th}$ word in the vocabulary. A decision can be made on the basis of the minimum distance among D(k), k = 1,2,...,K.

Although dynamic time warping (DTW) provides a reliable time alignment between reference and input patterns, the computation of the optimal alignment path is a very lengthy process. Several alternative procedures have been proposed for reducing the computation of DTW algorithms. However, these alternative methods generally suffer from a loss in the optimality of the alignment path found.

### 4.14.3 Ordered Graph Search (OGS) Approach to DTW

Brown and Rabiner [5] proposed an approach to dynamic time warping which shows essentially no loss in recognition accuracy with a 3:1 reduction in distance computation. The algorithm takes advantage of ordered tree and graph searching techniques to find the best path with substantially reduced computation of local distances. (A local distance being a distance between one frame of a reference word and one frame of the unknown word).

The nodes of figure 4.17 represent local distances, and allowable node transitions are represented by branches of the directed graph (digraph). The efficiency of the digraph search is achieved by omitting local distance calculations associated with nodes which are not searched. Conventional dynamic programming, on the other hand, involves the calculation of **all** local distances within the global constraints. Under certain readily obtainable conditions, the warping path determined by a digraph search can be shown to be optimal [5], and at a cost of about $^1/_3$ of the computation.

The $i^{th}$ node is designated by it's coordinates, i = (n,m), and the starting and ending nodes are s = (1,1) and t = (N,M) respectively. For any path passing through node i, the path cost (accumulated distance along the path) is denoted as

$$f(i) = g(i) + h(i) \qquad \ldots(4.52)$$

where g(i) is the minimal cost of the path from node s to node i, and h(i) is the minimal cost of the path from node i to node t.

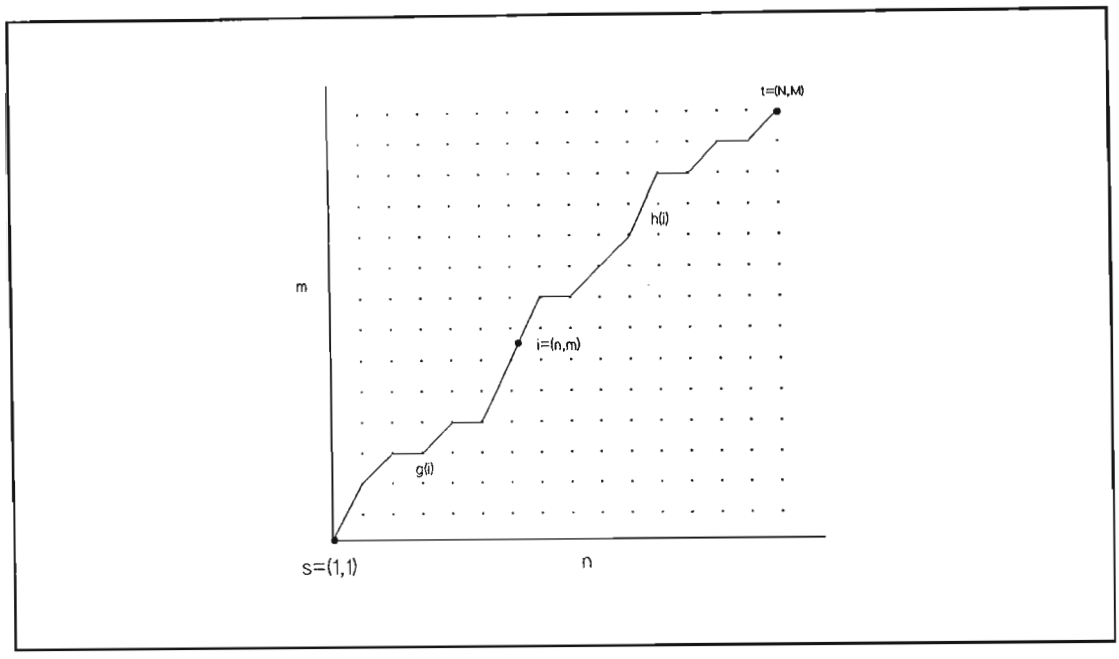



**Fig. 4.17 Illustration of the nodal structure and a typical path for ordered graph searching.**

For a directed search through the grid, the cost g(i) along the path to node i is known exactly. The cost h(i) from node i to node t, however, is not known and must be estimated. Thus an estimate of a minimal cost path passing through node i is

$$\hat{f}(i) = g(i) + \hat{h}(i) \qquad ...(4.53)$$

where $\hat{h}(i)$ is the estimate of h(i). The process of finding a minimal cost path though the grid involves the building up of a series of nodes for which the exact cost from the start node is known, and a cost to the terminal node is estimated. The node which currently provides the smallest cost estimate is expanded until the terminal node t is reached.

The cost h(i) from node i to the end node t in figure 4.17 is not known exactly and must be estimated by $\hat{h}(i)$. For the warping path to be optimal we require $\hat{f}(i) < f(i)$, and hence $\hat{h}(i)$ must under-estimate the true path cost h(i). If, however, the test and reference words are of different word classes, then we would prefer a gross over-estimator. We can, therefore, improve the efficiency by forcing $\hat{h}(i)$ to become large when preliminary results indicate that the test and reference words come from different classes. A potential mismatch can be detected early in the search by observing the average value of g(i) per test frame. A large value of g(i) will point to such a mismatch and indicates that we should make the estimate $\hat{h}(i)$ large.

Brown and Rabiner [5] have shown that this estimator function may be given by

$$\begin{aligned} \hat{h}(i) &= 0.7\, g(i)\, (N-n)/n \quad \text{if } g(i)/n < \beta \\ &= 2.0\, g(i)\, (N-n)/n \quad \text{if } g(i)/n > \beta \end{aligned} \qquad \ldots(4.55)$$

The values of 0.7 and 2.0 were determined experimentally, and $\beta$ is between 0.6 and 0.7. The chosen value of $\beta$ is essentially the largest reasonable average distance one would expect to encounter when comparing test and reference words of the same class.

Figure 4.18 gives an example of the digraph search. Initially node s is expanded into three possible successor nodes. Node 1 is found to have the smallest distance f and so this node is expanded to it's two possible successor nodes. Finally node 20 is found and state t is reached with a minimal cost path.

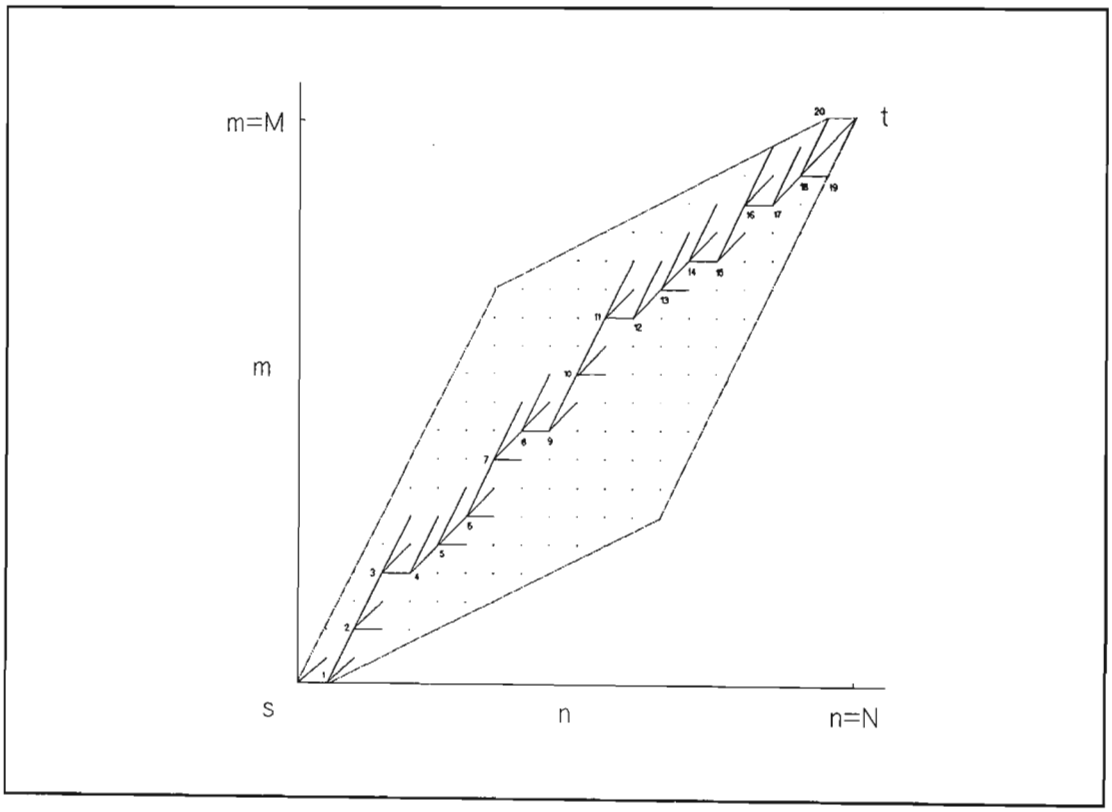



**Fig. 4.18 Illustration of the computation to determine a path from node s to node t, using OGS method.**

# Chapter 5

## IMPLEMENTATION OF THE SPEECH RECOGNITION SYSTEM

### 5.1 Introduction

A speech recognition system was implemented on an IBM PC using suitable interface hardware. The speech recognition techniques implemented used the following parameters when comparing an unknown word to one of the reference words:

1. Zero-crossing and energy differences
2. Autocorrelation coefficients
3. LPC coefficients
4. Cepstral coefficients

All measurements were performed by the author in an anechoic chamber with an area of $40m^2$ and a height of 4.0m. Informal comparisons were made of these results with the recognition rates achieved by other users in order to ensure that the results were not biased in any way. The effects of frame length, pre-emphasis, Hamming windows and dynamic time warping were also analyzed.

A description of the overall system, as well as of the different methods used follows, and a comparison of the results obtained is given in chapter 6. A block diagram of the speech recognition system, which was named **Big Ears**, is illustrated in figure 5.1 and a photograph of the working system is shown in figure 5.2.

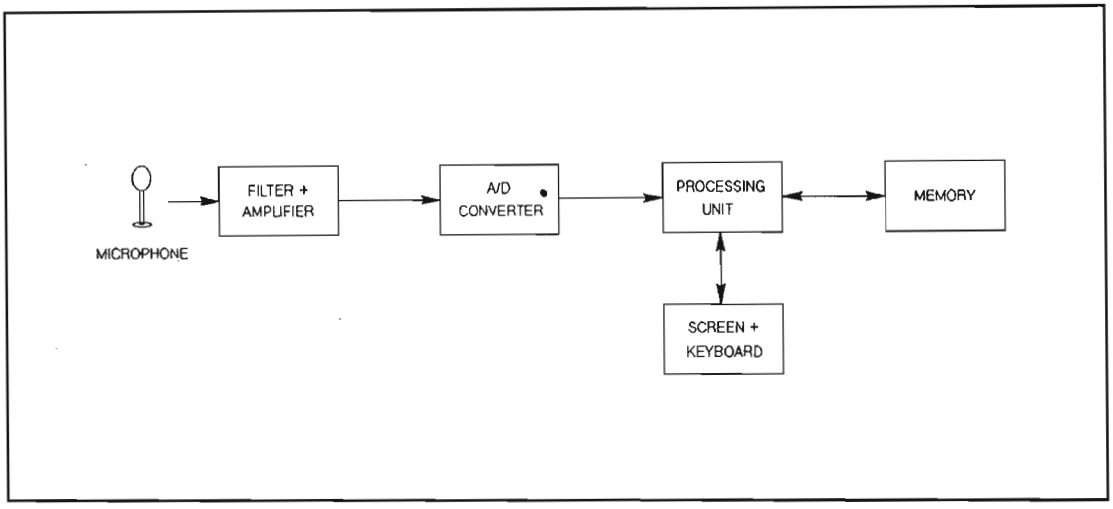



**Fig. 5.1 Block Diagram of the 'Big Ears' Speech Recognition System**

**Fig 5.2 Photograph of 'Big Ears' Speech Recognition System**

## 5.2 Hardware

### 5.2.1 Processing Hardware

In view of the intense signal processing involved in speech recognition, an obvious choice might have been a system using dedicated signal processing chips, such as the TMS 320 series. However, no such hardware was available for use in this project and furthermore, processing speed was not of prime importance since the word recognition was not expected to take place in real time.

Instead, a standard microprocessor was used. An IBM compatible PC was found to provide a good environment for the development and debugging of software. In addition, it has a well documented I/O interface allowing the use of custom hardware.

Most of the development and testing was performed on an 8MHz PC-XT (Intel 8088 and 8087 processors) although for the final implementation and measurements, a 12MHz PC-AT (Intel 80286 and 80287 processors) with 1Mb of expanded memory was used.

### 5.2.2 Analogue-to-Digital Converter

Before the speech signal could be processed by the computer it had to be converted into digital form by a suitable analogue-to-digital (A/D) converter. An important parameter in this conversion is the resolution of the A/D converter and the associated quantization noise. For example, an 8-bit A/D converter converts an input signal into only 256 levels, introducing noise into the measurement. Witten [17] shows that at least 11 bits are needed for adequate representation of speech signals. Most speech recognition systems use 12-bit A/D converters, giving 4096 quantization levels.

An ST4303 12-bit A/D converter was used to digitize the speech signal. The ST4303 is a board designed for data acquisition applications in the STD BUS environment and thus has an STD BUS interface. In order to communicate with the ST4303 from an IBM PC it was necessary to design an interface card that would convert the STD BUS control signals to the IBM PC standard. This interface card fitted inside the PC and connected to the ST4303 board via a ribbon cable. (The specifications for the ST4303 A/D converter are given in Appendix B, together with the circuit diagram for the interface card that was designed).

Another important parameter in the sampling of speech is the sampling rate. It is well known that if the highest frequency component of a signal has a frequency $f_c$, then the sampling frequency must be at least $2f_c$ (known as the Nyquist frequency). Although speech signals can contain components at frequencies of up to 10kHz, the analysis bandwidth can safely be reduced to between 3.5kHz and 5kHz and most recognition systems use sampling rates of between 7kHz and 10kHz [29,31]. It is interesting to note that the bandwidth of a telephone system is only 3.4kHz.

### 5.2.3 Input Filtering and Amplification

Since the output of the microphone was only of the order of millivolts, the signal had to be amplified to lie within the +-10V input range of the A/D converter. Two inverting amplifiers (one fixed and one variable) were cascaded to give a variable gain of 100x to 10000x. The circuit diagram for this amplifier is given in appendix C.

It was necessary to filter the speech signal before sampling for two reasons. Firstly, a low-pass filter with a cut-off frequency of at least half the sampling frequency was required to avoid aliasing problems and secondly, a notch or high-pass filter was needed to filter out the mains interference (hum) at 50Hz.

A sampling rate of 10kHz was used and a $6^{th}$ order Chebyshev low-pass filter was designed with a 3dB cut-off frequency of 3.2kHz. This provided 40.1dB attenuation at 5kHz.

At the lower end of the frequency scale, a high-pass filter was chosen in preference to a notch filter. Although a notch filter would have taken care of mains hum, a high-pass filter was found to be a better alternative since most of the ambient noise in a fairly quiet environment lies below 100Hz [30]. In view of this, and the fact that very little speech energy lies below 200Hz, a high-pass filter with a 3dB cut-off at 175Hz was used. A fourth-order Butterworth filter was designed and gave 20.3dB attenuation at 100Hz and 44.5dB at 50Hz.

Figure 5.3 shows the combined frequency response of the above two filters and their circuit diagrams may be found in Appendix C.

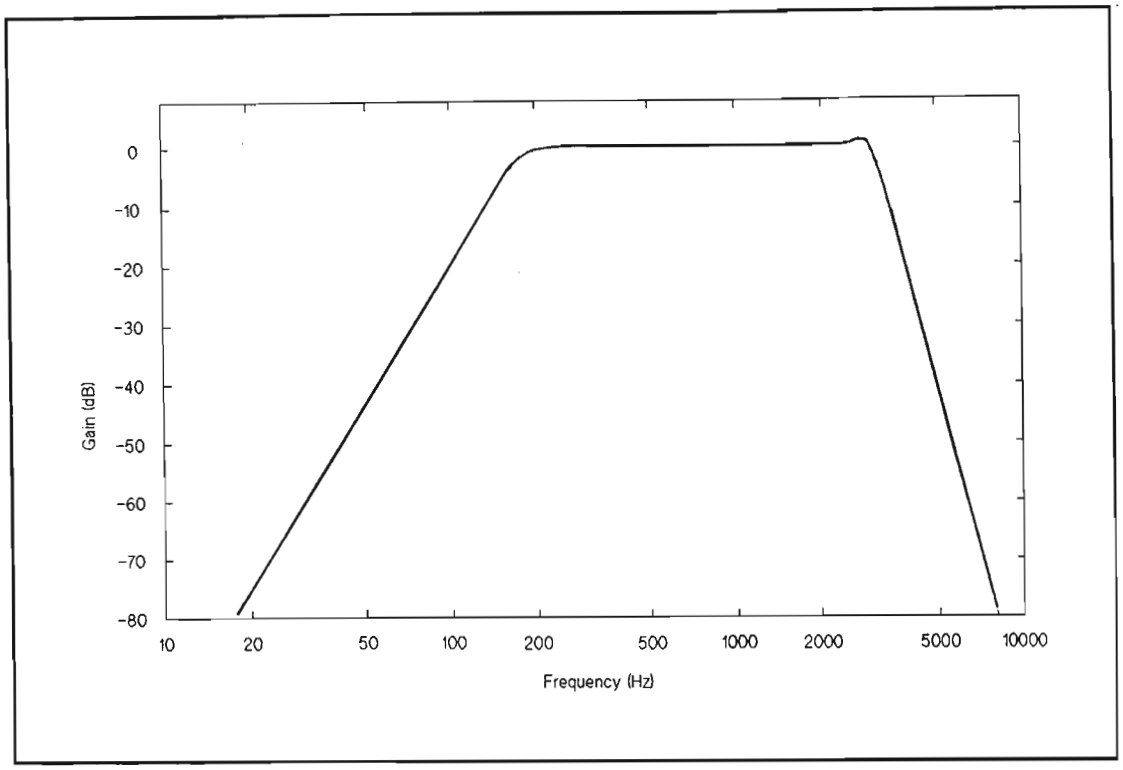



**Fig. 5.3 Overall Frequency Response of Input Filters**

### 5.2.4 Microphone

The microphone used for all measurements was a Brüel & Kjær precision condenser microphone (type 4165). It is designed to have a linear frequency response and fits onto a Brüel & Kjær sound level meter (type 2218). The output was of the order of tens of millivolts for a speech input.

## 5.3 Software

### 5.3.1 Introduction

Implementing the speech recognition system on an IBM PC meant that there was a wide range of programming languages available. Firstly it was necessary to decide whether to program in 8086 assembler or to use a high level language such as Pascal, C or Fortran.

In view of the complexity of some of the algorithms used, it would have taken a braver (and perhaps more foolish!) person than myself to have written all the software in assembly language. Most of the software for the speech recognition system was written in the 'C' programming language. When implementing the speech recognition system on a PC-XT, however, it was not possible to attain the 10kHz sampling frequency required without writing an assembly language procedure to control the A/D converter. This was later replaced with a procedure written in 'C' when using the faster PC-AT.

The software for the Big Ears speech recognition system consists of a main program called BIG_EARS.C and a header file, called BIG_MESS.H, that contains all the menus and messages to be displayed. It needs to be pointed out that the name of this header file is derived from the word 'message' and that nothing else is implied! Although the display software contained in BIG_MESS.H was written for an EGA colour monitor, it will also run on a monochrome monitor.

The two photographs in figures 5.4 and 5.5 show two of the many display screens.


**Fig. 5.4 Menu Option Requesting Recognition Technique**

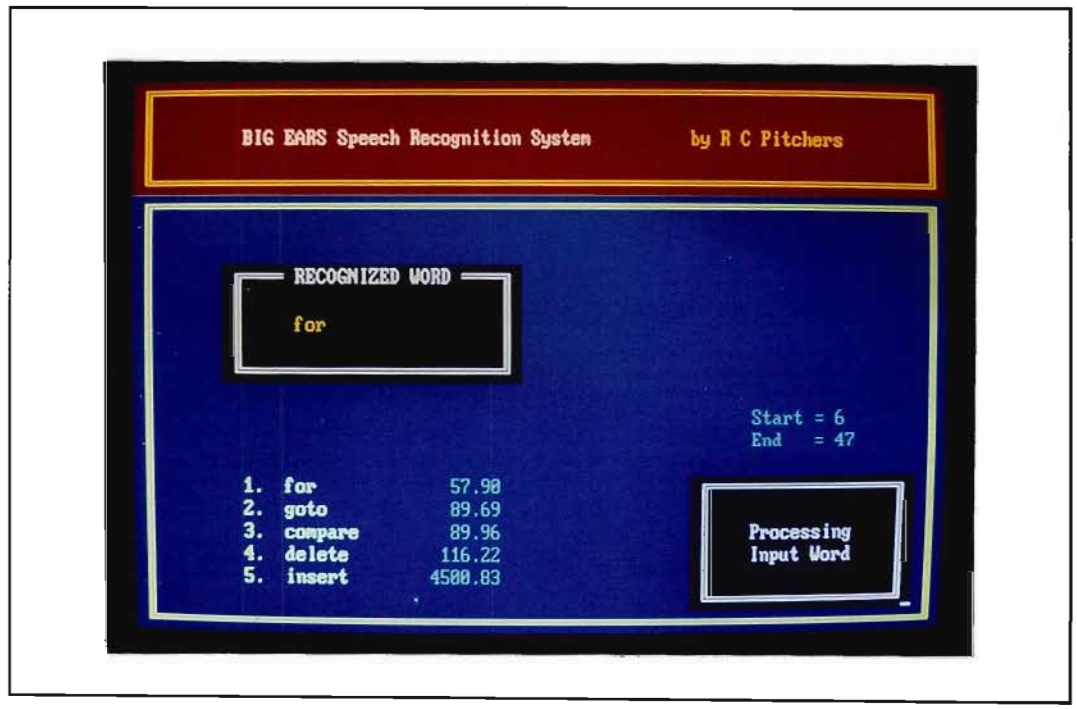



**Fig. 5.5 Display of Previously Recognized Word While Processing Current Word**

### 5.3.2 **Main Program**

The two main functions of the BIG_EARS.C program are to train the speech recognition system and to recognize words using one of four algorithms. The main program consists of an initialization section followed by various prompts (to be discussed later in this section). In addition there are six main procedures:

1. Train_System()
2. Recognize_Word()
3. Save_Vocabulary()
4. Load_Vocabulary()
5. Sample_Word()
6. Frame_Distance()

The first four procedures are called directly by the main program while Sample_Word() is called by both Train_System() and Recognize_Word(). In addition, Recognize_Word() makes calls to the Frame_Distance() procedure.

Brief descriptions are given for most of these procedures together with top-level flowcharts and, should the reader desire a more detailed description, the source code listings may be found in Appendix D. The procedures used to save and load vocabularies and allocate and map expanded memory (EMS) are not described here although the source listing for these procedures may also be found in Appendix D. Similarly, the header file containing the menu displays is not described.

A flow-chart of the main program is given in figure 5.6. After initializing the ST4303 A/D board, a lookup table that is used in the Hamming window calculation is built. This lookup table serves to substantially reduce the number of multiplications required when placing a Hamming window over a frame of samples.

The digital value obtained by the A/D converter for an input of zero volts was found to vary slightly off the theoretical value of 2048, depending on the input filtering and amplification used. Since this value is used as an offset to be subtracted from all sampled data when converting the data to signed integer values, it is important that it be as accurate as possible. In order to calculate the exact magnitude of this digital zero, the user is required to place a short-circuit across the input to the filter/amplifier circuit while a number of samples are taken. The average of these samples is then used as a zero-offset.

May I at this point apologize to any female readers who may be offended by the exclusive use of masculine terminology when referring to 'the user'. It is for the sake of grammatical convenience and no offense is intended.

After an introductory message, the user is asked whether he would like to train the system or load an existing vocabulary in order to go directly into recognition mode.

If the user chooses to train the speech recognition system, then the Train() procedure is called, after which the vocabulary is saved by making a call to Save_Vocabulary(). If, on the other hand, the user chooses to go into recognition mode, he is prompted to enter the name of the vocabulary he wishes to use. This vocabulary is then loaded by means of a call to the Load_Vocabulary() procedure.

At this point the user has either trained the system or loaded an existing vocabulary. After choosing whether or not to use dynamic time warping in the recognition process, the user must select one of the four recognition techniques available. Having done this, any word spoken into the microphone is recognized and displayed on the screen. The speech recognition system will continue to recognize words until any key is hit, at which point the program terminates.

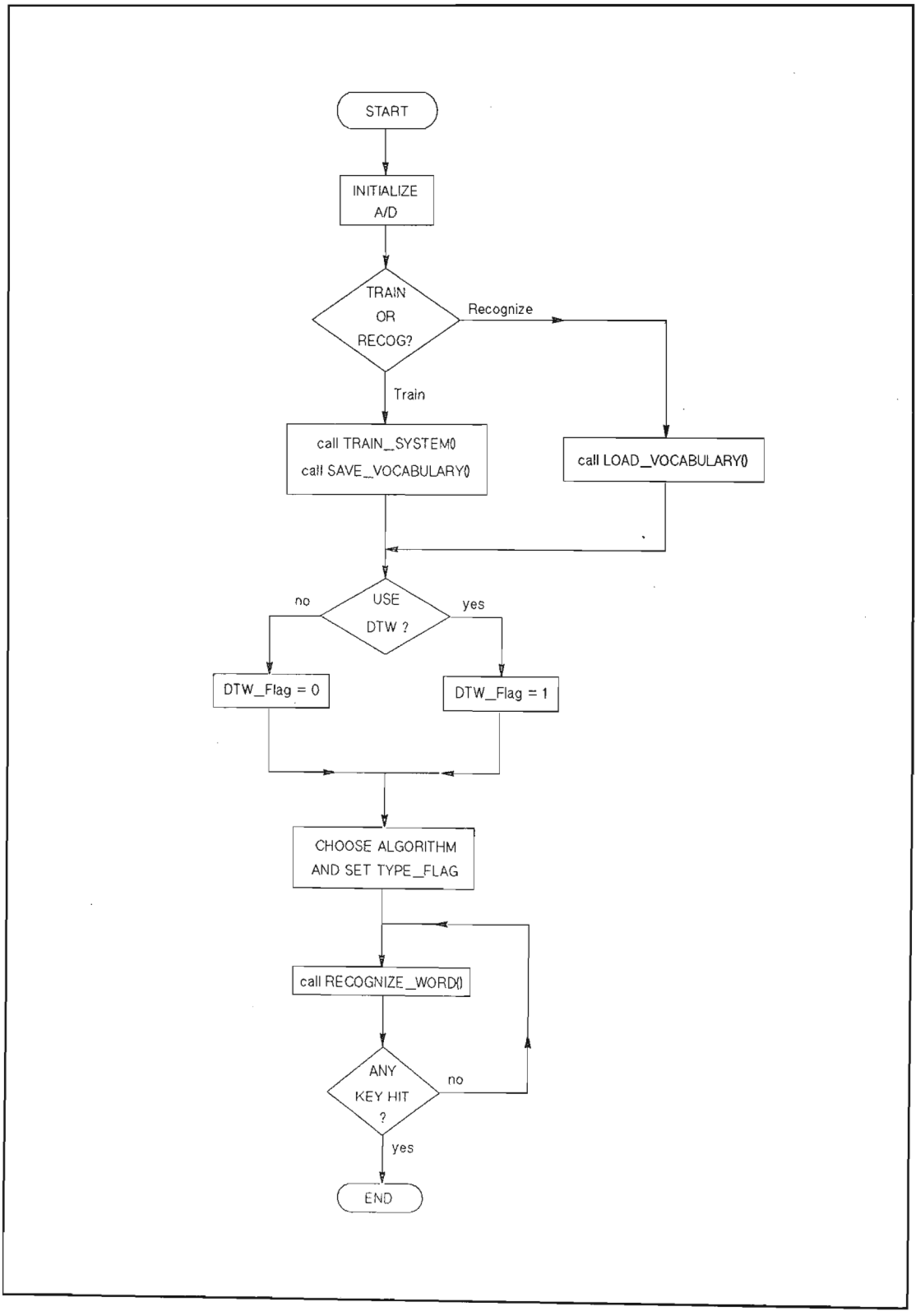



**Fig. 5.6 Flowchart of Main Program**

### 5.3.3 Procedure for Sampling a Word

#### 5.3.3.1 Body of Procedure

The Sample_Word() procedure is responsible for detecting the start of an utterance, sampling for a period of approximately one second and then locating the end of the word. Having done so it then calculates the parameters needed for the recognition process. A flowchart of this procedure is given in figure 5.8. The two blocks marked 'Look for Beginning of Word' and 'Locate end of Word' are described in section 5.3.3.2.

Having sampled the whole word and stored all the samples in consecutive locations of a buffer, the word is divided up into overlapping frames as illustrated by the example in Figure 5.7. The frame size used was determined by the maximum rate at which the speech waveform changes. This rate is in fact the rate at which the articulators change their position and is seldom greater than 100Hz [13]. Frame sizes ranging from 100 to 450 samples (10 to 45 ms) were used and for analysis purposes the waveform can be considered constant over such an interval.

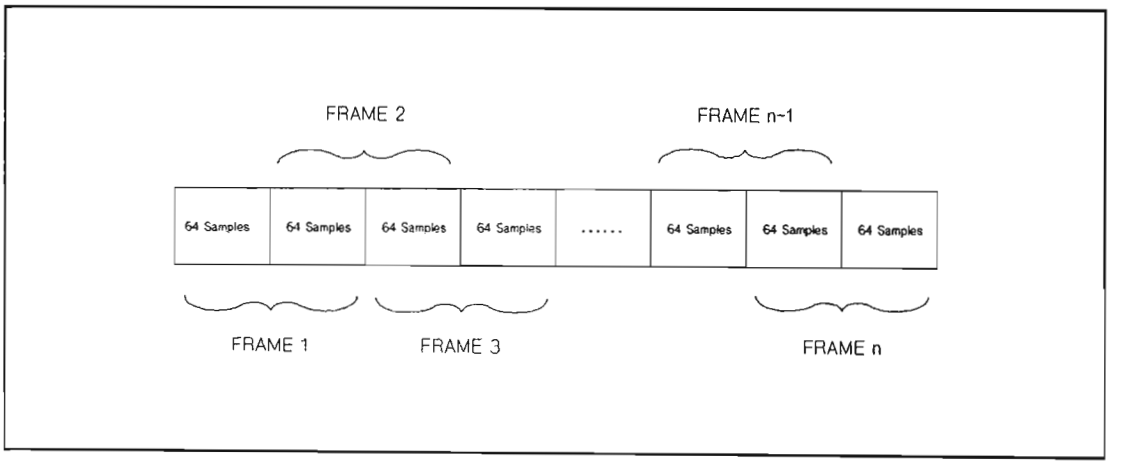



**Fig. 5.7 Overlapping Frame Structure used for word of n frames with 128 samples per frame**

Each frame in the word is pre-emphasized using the function $y(n) = x(n) - 0.95\ x(n-1)$ and a Hamming window is then placed over the frame. All or some of the following parameters are then calculated - energy content, zero-crossing counts, autocorrelation coefficients, normalized autocorrelation coefficients, LPC coefficients and Cepstral coefficients. If the system is in training mode then all of these parameters are calculated. When in recognition mode, however, only the parameters required by the recognition technique selected are calculated.

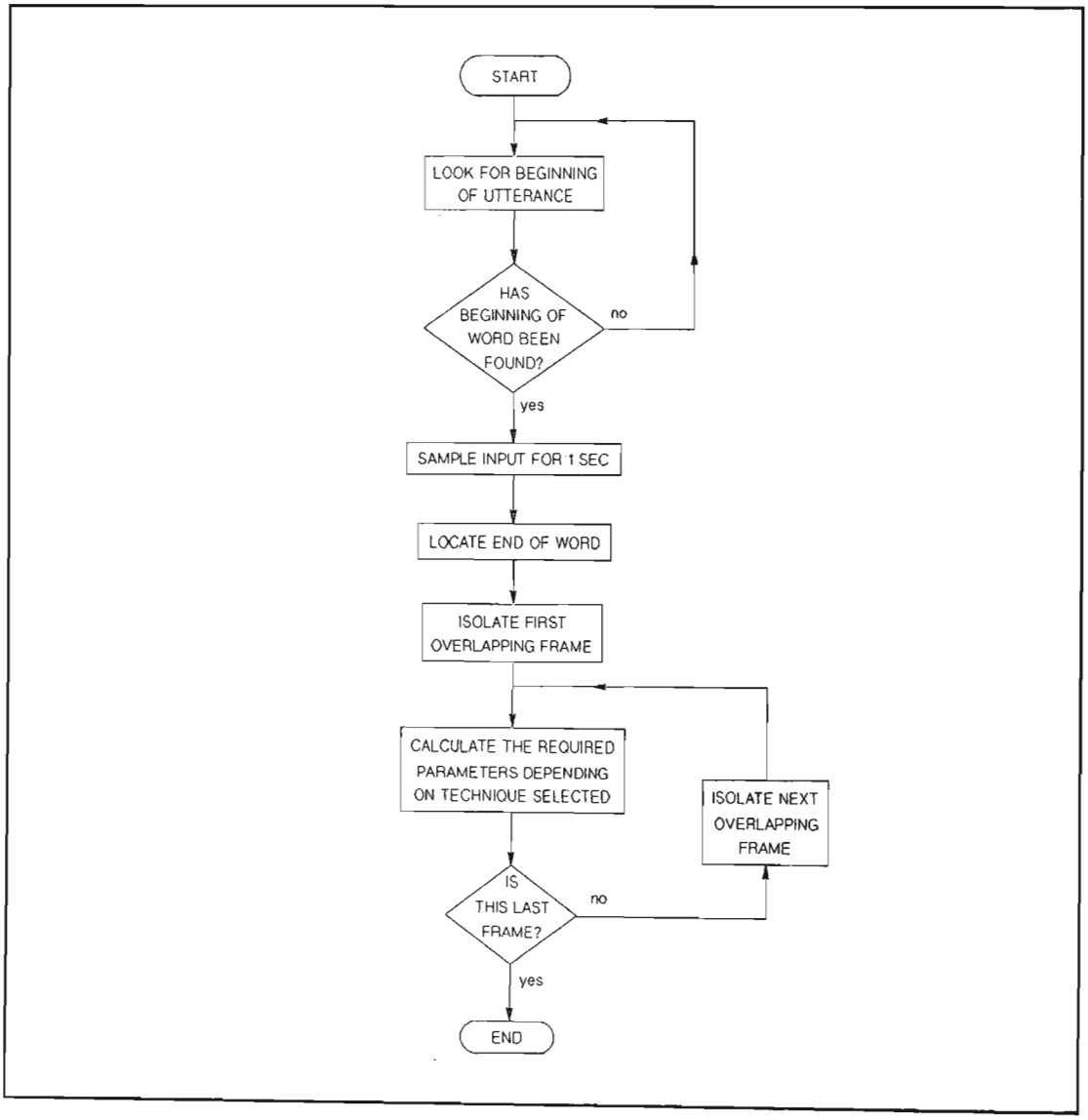



**Fig. 5.8 Flowchart of Sample_Word() Procedure**

#### 5.3.3.2 Design of an Algorithm to Locate the Endpoints

The algorithm designed to detect the beginning and ending points of a word was based on one proposed by Rabiner and Sambur [26] and incorporates some ideas of Lamel et al [25]. The algorithm uses the average energy (magnitude) and the number of zero-crossings in a frame to detect the two endpoints.

A set of three energy thresholds $E_U$, $E_L$, and $E_E$ are defined as illustrated in figure 5.9. The beginning of the word is provisionally chosen as that point, $N_1$, at which the energy exceeds a lower threshold $E_L$, provided that the energy does not again fall below $E_L$ before it exceeds an upper threshold $E_U$. It is reasonable to assume that the beginning point does not lie after $N_1$. It may, however, be located before $N_1$ if the word starts, for example, with a weak fricative.

The next step is therefore to move backwards from $N_1$ comparing the zero-crossing rate to a threshold, $ZC_{th}$, determined from the statistics of the zero-crossing rate for the background noise. Only the 25 frames preceding $N_1$ are considered and if the zero-crossing rate exceeds the threshold in three or more consecutive frames, the beginning point $N_1$ is moved back to the first point at which the zero-crossing threshold was exceeded. Otherwise $N_1$ is defined as the beginning of the utterance.

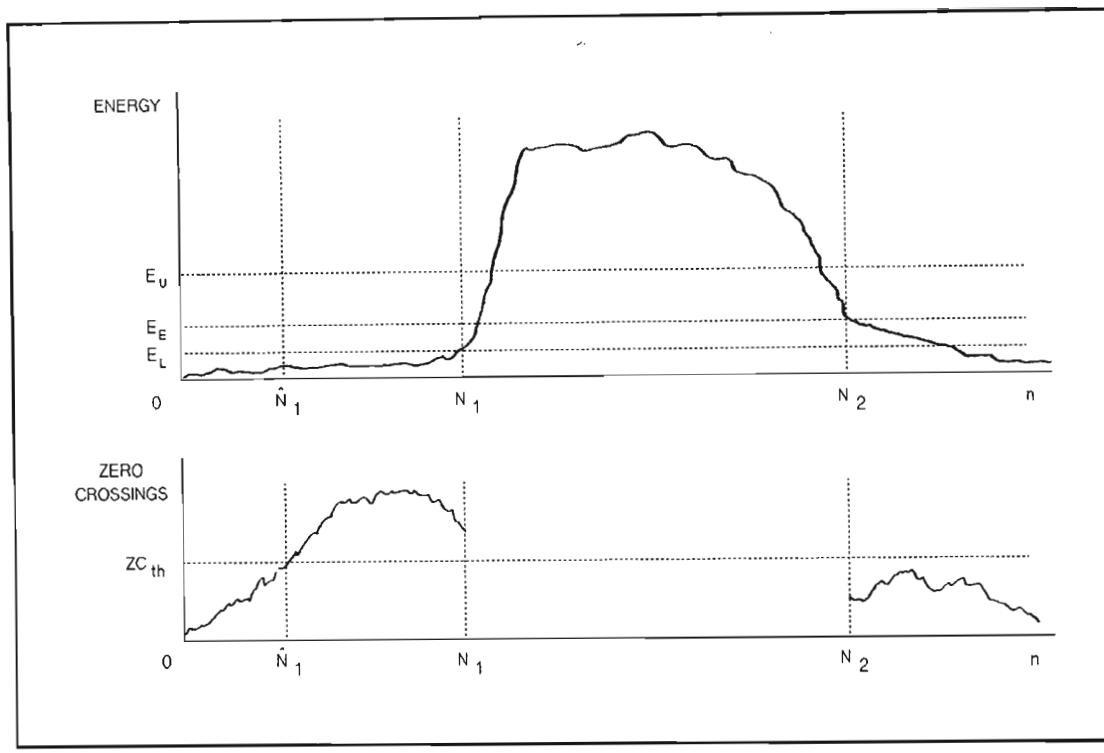



**Fig. 5.9 Typical example of energy and zero-crossing measurements for a word with a fricative beginning [26]**

To find the ending point $N_2$, a similar search is carried out backwards working from frame $N_1+N_{max}$ (where $N_{max}$ is the maximum duration that any word is likely to have - typically 0.7-1.0 seconds or 70-100 frames of 10 msec each). The reason for working backwards to find the endpoint, and not simply forward from $N_1$, is to avoid omitting part of a word. This is shown in figure 5.10 where it is obvious that $N_2'$ would have been chosen as the end of the utterance if a forward search from $N_1$ had been used to find the first point at which the energy fell below $E_E$. On the other hand, by searching backwards from frame $N_1+N_{max}$, the correct endpoint $N_2$ is located.

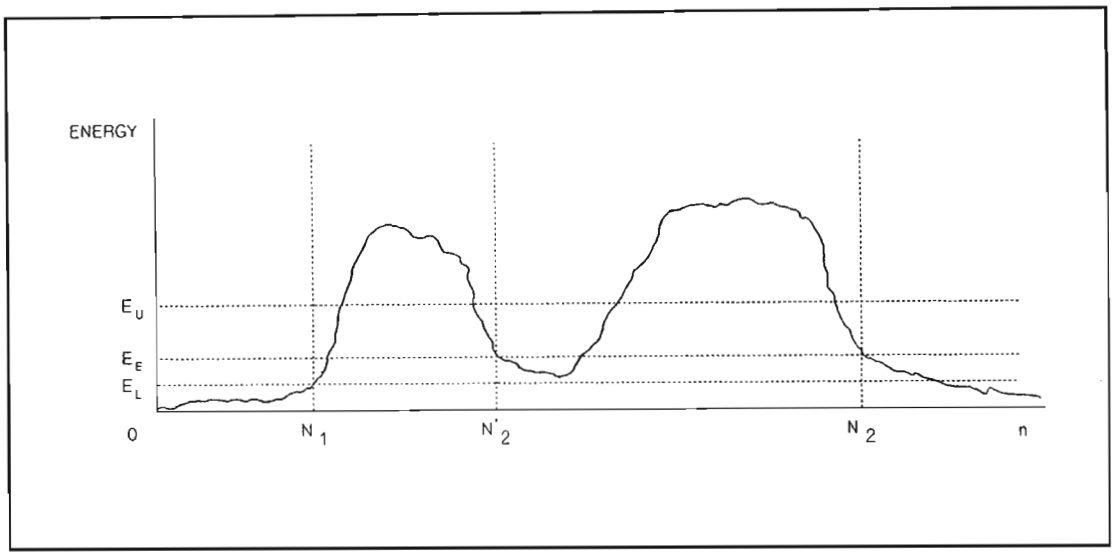



**Fig. 5.10 Typical example of energy plot of a word with a stop gap in the middle**

### 5.3.3.3 Determining the Thresholds

The energy thresholds $E_L$, $E_U$ and $E_E$, and the zero-crossing threshold $ZC_{th}$ were determined by trial and error, experimenting to see what thresholds gave the best results. The actual thresholds used were very dependent on the actual level of the background noise. The energy thresholds were set to the following values, where $E_{background}$ was the average energy level measured due to background noise:

$$E_L = 25 * E_{background}$$
$$E_U = 100 * E_{background}$$
$$E_E = 40 * E_{background}$$

The zero-crossing threshold was set to detect signals from about 1.3 kHz upwards.

#### 5.3.3.4 **Flowchart of Endpoint Detection Algorithm**

Figure 5.11 shows the algorithm used for estimating the beginning point of an utterance. The algorithm continually samples sections (frames) of the input, calculating the number of zero-crossings and the total energy in each frame.

The algorithm waits for the energy of the input signal to exceed the lower threshold $E_L$ at which time the frame number is stored as the preliminary starting point. It then waits for the upper threshold $E_U$ to be exceeded without the energy having dropped below $E_L$. Once the energy has risen above $E_U$, the zero-crossing rates in the 25 frames preceding the preliminary starting point (which, incidentally, corresponds to $N_1$ in figure 5.9) are checked to see whether the zero-crossing threshold was exceeded three or more times. If so then the start of the word is moved to that frame at which the threshold was first exceeded.

Although from the opposite end of the word, the algorithm used for detecting the end of an utterance is essentially very similar to the process of determining the beginning point as described above. Figure 5.12 shows a flowchart of this algorithm.

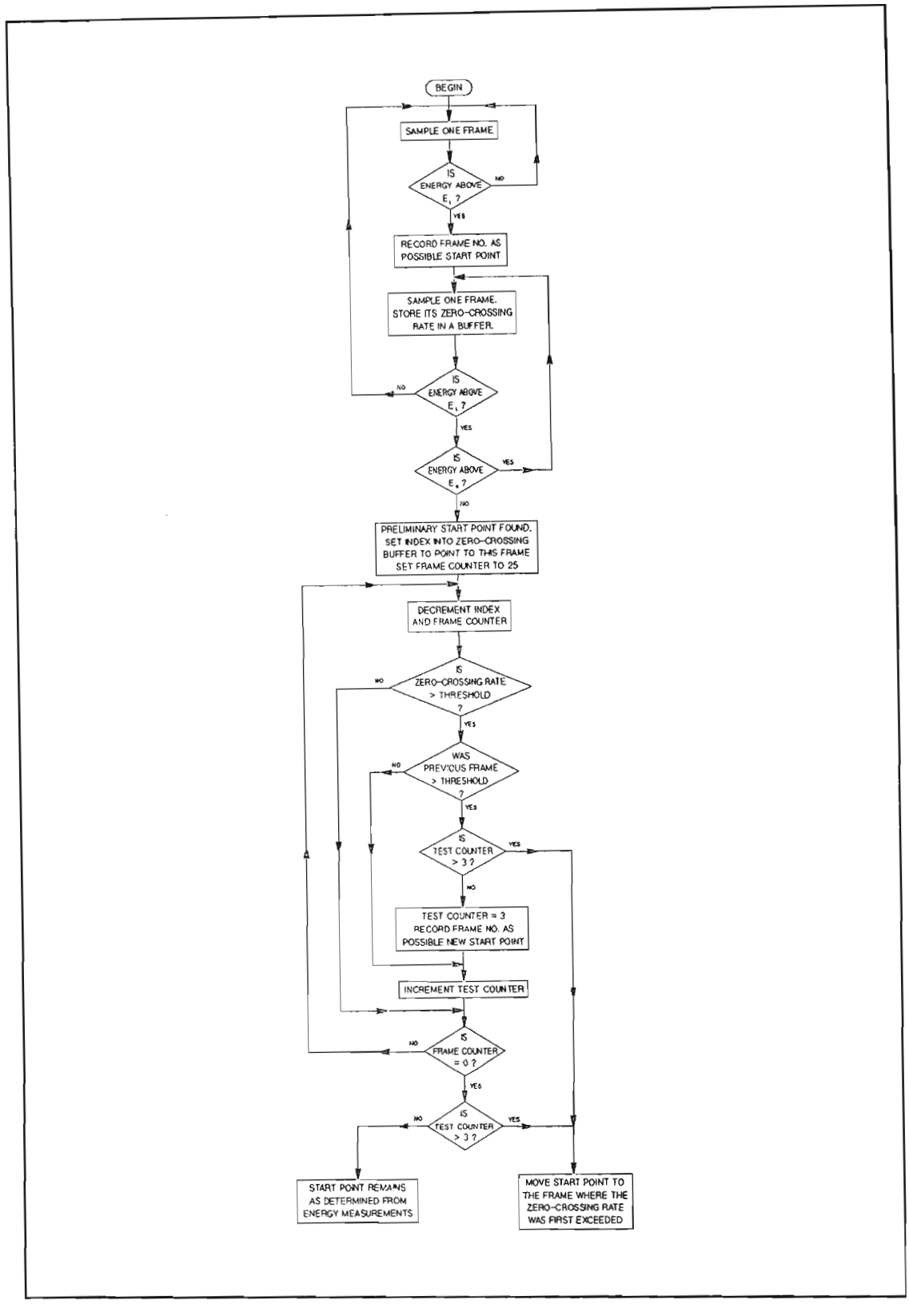



**Fig. 5.11 Flowchart of Algorithm Used for Detecting the Beginning of an Utterance.**

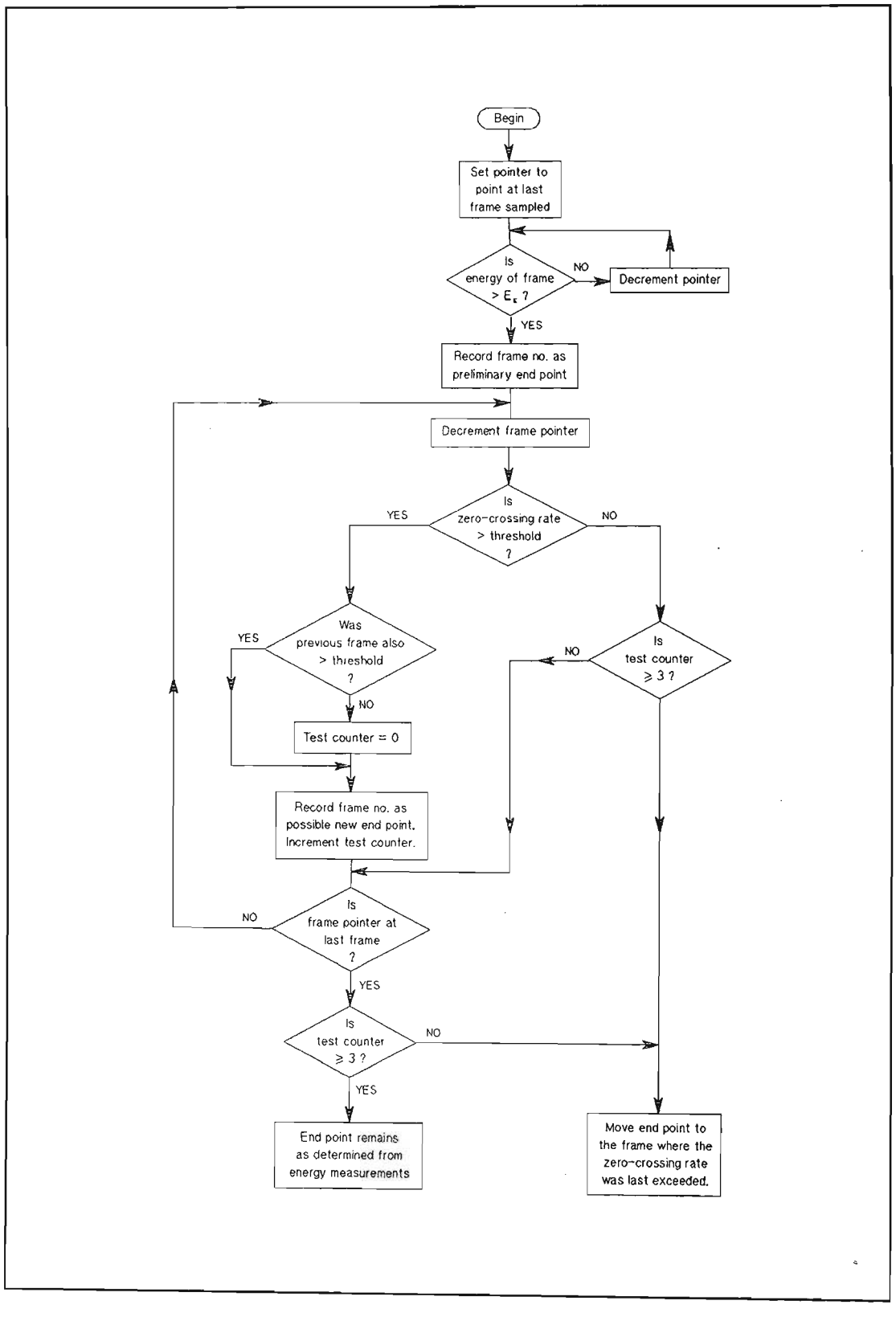



**Fig. 5.12 Flowchart of Algorithm Used for Detecting the End of an Utterance.**

### 5.3.4 System Training Procedure

Figure 5.13 shows a flowchart of the Train_System() procedure that is used to train the Big Ears speech recognition system. A short message explaining how to train the system is displayed, after which the user is requested to enter the number of words he wishes to include in the current vocabulary. Memory is then allocated to hold the parameters used to describe each of these words.

The user is repeatedly requested to type in and say a word until all the words have been entered. After each word has been spoken, the user is given the option of repeating that word should he not be satisfied with the way in which it was said. The spoken word is captured by making a call to the Sample_Word() procedure and the parameters calculated by this procedure are stored in memory for later use.

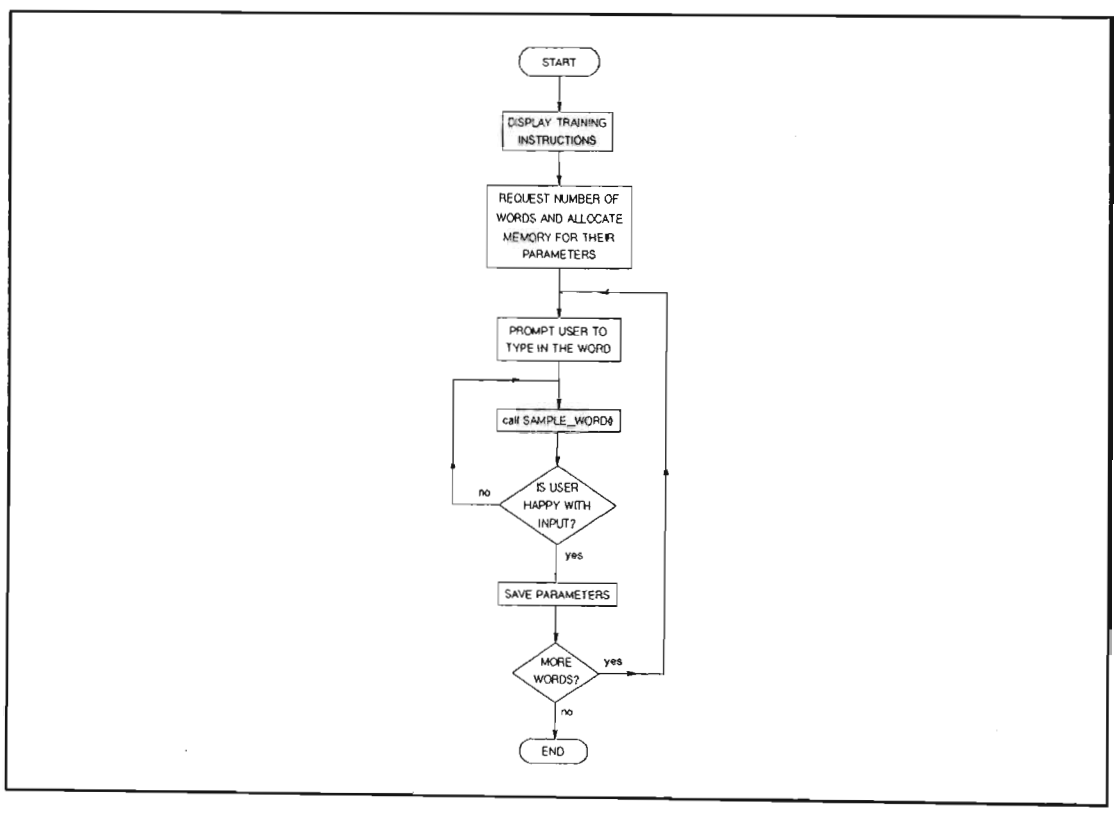



**Fig. 5.13 Flowchart of Train_System() Procedure**

### 5.3.5 Word Recognition Procedure

#### 5.3.5.1 Body of Procedure

The Recognize_Word() procedure is illustrated in Figure 5.14 by a flowchart. The word to be recognized is captured by means of a call to the Sample_Word() procedure. This word is then compared to each word contained in the vocabulary. If the dynamic time warping option has not been selected then a linear frame-for-frame comparison is done, the frame distance calculations being determined by the Frame_Distance() procedure. This procedure is passed the recognition technique currently in use and does the appropriate distance calculation.

If, however, dynamic time warping is to be used then the frame comparison is not a linear process (ie. frame **n** of the unknown word is not necessarily compared with frame **n** of the reference word) and the DTW algorithm - discussed in the following section - is responsible for calling the Frame_Distance() procedure.

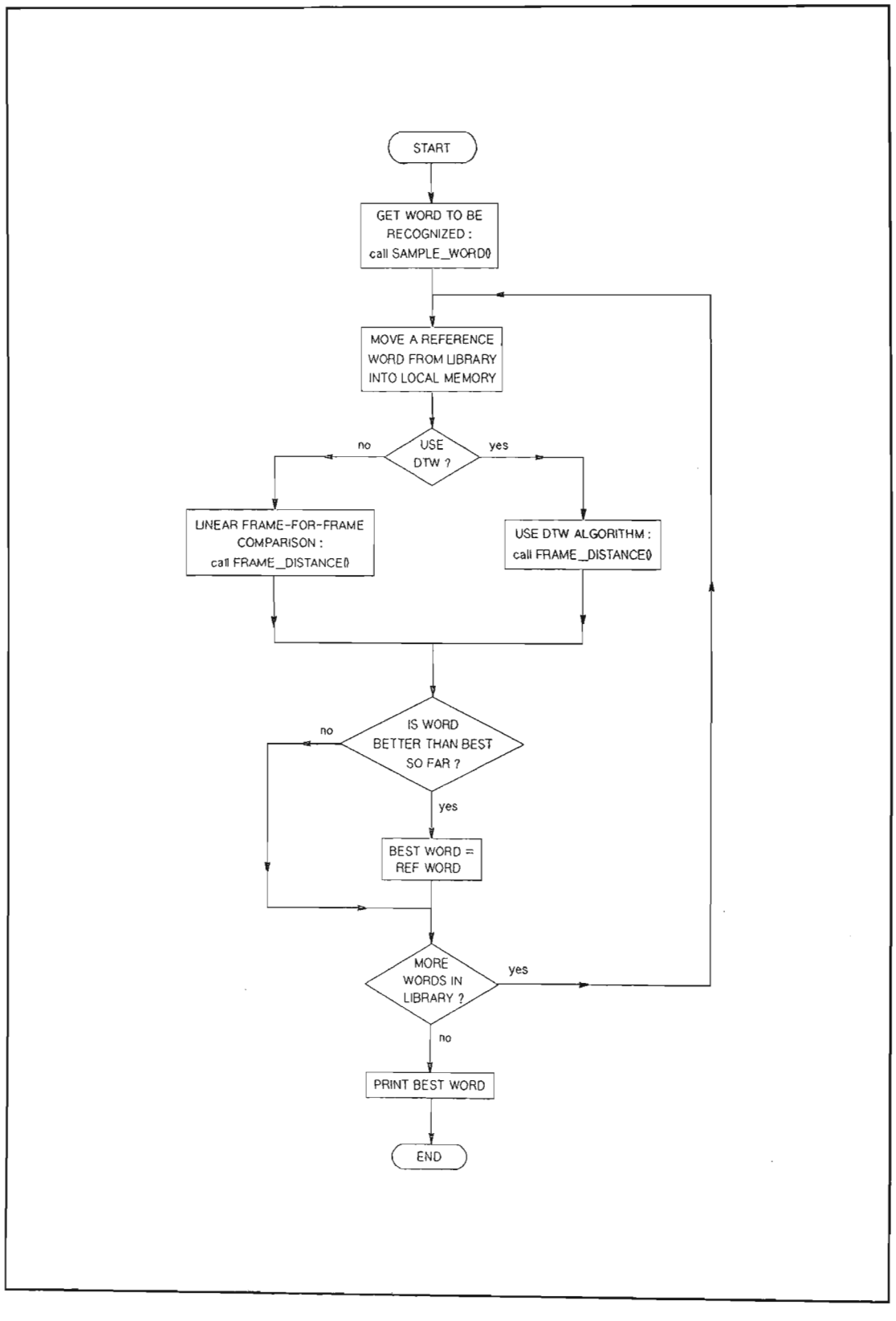



**Fig. 5.14 Flowchart of Recognize_Word() Procedure**

#### 5.3.5.2 **The OGS Algorithm for DTW**

The theory of the ordered graph searching (OGS) approach to DTW was explained in section 4.14.3 and a flow diagram of the algorithm used is given in figure 5.15. Referring to figure 4.17, any path from start node s to intermediate node i is completely characterized by a nodal state, which includes:

1) the node coordinates (n,m)
2) the cost g(i) from the start node to node i
3) the estimated cost $\hat{h}(i)$ from node i to end
4) a pointer to the previous node on the path

By using the above nodal state information, an 'open' list of potential paths is made where each entry in the list is the nodal state of the **last** node on the path. The last node of the path having the lowest estimated total cost $\hat{f}(i)$ is placed at the top of the list. The algorithm tries to find the minimum path by removing the top node from the open list and, after saving it's nodal state on a 'closed' list, finding all permissible successor nodes. New path cost estimates are computed for each of these successor nodes and they are sorted into the open list. The process continues until node t is found.

Initially, start node is the only node on the open list and the closed list is empty. The open list is always sorted so that the node having the least estimated warp path heads the list. This node at the top of the list is then expanded into successor nodes. Provided that the generated node lies within the global constraints, $\hat{f}(i) = g(i) + \hat{h}(i)$ is then computed and using this value the node is inserted in the open list at the appropriate locations. When the terminal node is found, the g(i) already calculated for this node is then equal to the path cost g(t) = f(t).

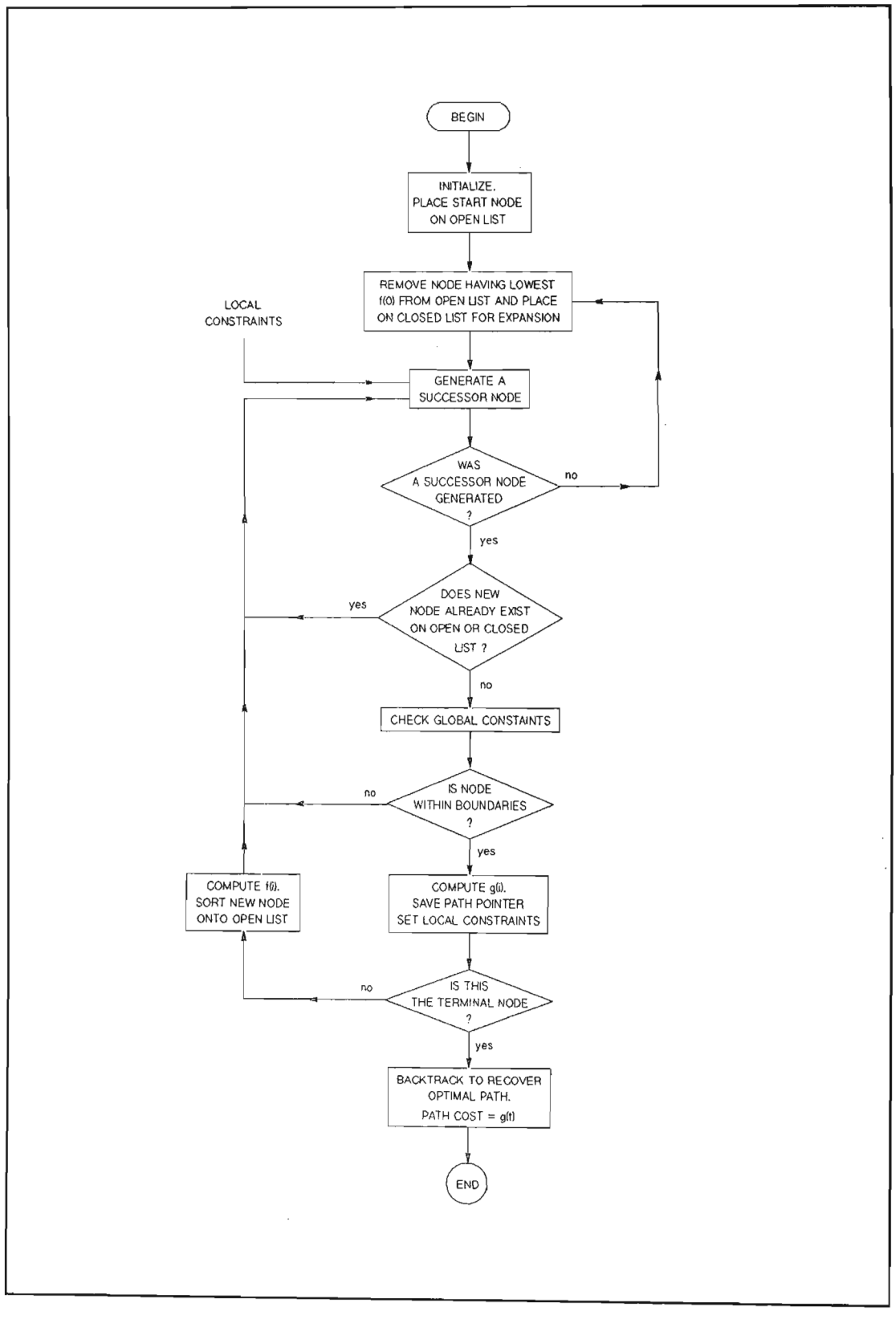



**Fig. 5.15 Flowchart of the OGS Method of Dynamic Programming**

### 5.3.6 Comparison of Word Templates

In order to compare two words, each with their own sets of feature vectors (eg. LPC coefficients, cepstral coefficients, energy, etc.), it is necessary to be able to compare two frames and obtain some quantitative measure of similarity. The Frame_Distance() procedure is called by the Recognize_Word() procedure and is passed two frame numbers, one of the reference word and one of the unknown word. It returns the distance between these two frames, using one of four distance measures to do the calculation. The distance measure used is obviously determined by the recognition technique that has been selected. The distance measures associated with the four techniques are discussed below.

#### 5.3.6.1 Zero-Crossing/Energy Measurement

The number of zero-crossings in a frame gives an indication of the frequency content of the speech signal over that time interval. Possibly the simplest method of comparing two words is to sum the differences in zero-crossings in each of their frames. This method was taken one step further and an energy measurement was incorporated into the frame distance calculation. Since the total energy of a frame is numerically far greater than the number of zero-crossings, the energy difference cannot simply be added to the zero-crossing difference. Furthermore, the frequency information is of more interest than the energy content. Instead of including an energy difference, the following distance measure was devised in which a weighted energy ratio is used.

$$d(n,m) = ZC_t(n) - ZC_r(m) + \alpha[E_t(n)/E_r(m)] \quad \text{if } E_t(n) > E_r(m) \qquad \ldots(5.1a)$$

$$d(n,m) = ZC_t(n) - ZC_r(m) + \alpha[E_r(m)/E_t(n)] \quad \text{if } E_t(n) \leq E_r(m) \qquad \ldots(5.1b)$$

$ZC_t(n)$ and $E_t(n)$ are the zero-crossing and energy values for the $n^{th}$ frame of the test input word, while $ZC_r(m)$ and $E_r(m)$ correspond to the $m^{th}$ frame of a reference word. $\alpha$ is a constant used to set the weighting of the energy contribution to the distance measure.

#### 5.3.6.2 **Autocorrelation/LPC Measurement**

The method used to compare a reference word with the test utterance for the second speech recognition technique involved mapping the autocorrelation coefficients of the test word onto the LPC coefficients of the reference word. The distance measure, consisting of a sum of products, is defined as

$$d(n,m) = \sum_{i=0}^{p} [r_{tn}(i) * a_{rm}(i)] \qquad \ldots(5.2)$$

where $r_{tn}(i)$ are the autocorrelation coefficients of the $n^{th}$ frame of the test word and $a_{rm}(i)$ are the LPC coefficients of the $m^{th}$ frame of the reference word.

#### 5.3.6.3 A Squared Euclidean Distance for Cepstral Coefficients

The distance measure used to compare two sets of cepstral coefficients was a squared Euclidean distance. In other words the distance between the $n^{th}$ frame of the test word and the $m^{th}$ frame of a reference word is given by

$$d(n,m) = \sum_{i=1}^{p} [c_{tn}(i) - c_{rm}(i)]^2 \qquad ...(5.3)$$

where p is the order and $c_{tn}(i)$ and $c_{rm}(i)$ are the cepstral coefficients of the $n^{th}$ and $m^{th}$ frames of the test and reference words respectively.

#### 5.3.6.4 Ikatura's LPC Distance Measure

The final speech recognition technique that was implemented made use of the widely used distance measure proposed by Ikatura (often known as the log likelihood ratio). This distance measure, defined earlier by equation 4.36 and discussed in section 4.11.5, is repeated here

$$d = \log_e \frac{A_r \, \mathbf{R}_t \, A_r^T}{A_t \, \mathbf{R}_t \, A_t^T} \qquad ...(5.4)$$

where $A_r$ and $A_t$ are the vectors of LPC coefficients describing the reference and unknown test words respectively and $\mathbf{R}_t$ is the autocorrelation matrix of the test word. The distance defined by equation 5.4 was calculated using the computationally efficient form defined in equation 4.40.

## 5.4 Test Vocabularies

Use was made of four test vocabularies in the comparisons of the relative performances of the speech recognition techniques. The first was a ten word vocabulary consisting of the digits zero to nine. The second and third vocabularies consisted of 30 and 60 computer terms respectively and the fourth vocabulary comprised of 120 airline booking terms taken from a vocabulary used by Wilpon et al [28].

| | | | |
|---|---|---|---|
| INSERT | IF | BEGIN | NUMBER |
| DELETE | ELSE | END | ADD |
| REPLACE | DO | ADDRESS | SUBTRACT |
| WORD | WHILE | OVERFLOW | MULTIPLY |
| READ | GOTO | REGISTER | DIVIDE |
| WRITE | CALL | MEMORY | ABSOLUTE |
| SAVE | RETURN | POINTER | REMAINDER |
| LOAD | BREAK | | |

**Table 1 30-word Computer Vocabulary**

| | | | |
|---|---|---|---|
| NAME | BINARY | BIT | DEFINE |
| PROGRAM | DECIMAL | BYTE | DECLARE |
| PROCEDURE | OCTAL | INPUT | INCLUDE |
| FUNCTION | HEX | OUTPUT | EXIT |
| SUBROUTINE | INTEGER | COMPARE | CONTINUE |
| STRUCTURE | REAL | LIST | COMPILE |
| STRING | CHARACTER | STORE | ASSEMBLE |
| ARRAY | CONSTANT | | |

**Table 2 Additional 30 words to give 60-word Computer Vocabulary**

| | | | |
|---|---|---|---|
| Monday | one | cash | a |
| Tuesday | two | credit | at |
| Wednesday | three | master | are |
| Thursday | four | diners | do |
| Friday | five | card | does |
| Saturday | six | club | I |
| Sunday | seven | pay | my |
| January | eight | code | by |
| February | nine | American | is |
| March | ten | express | in |
| April | eleven | fare | on |
| May | twelve | first | of |
| June | O'clock | class | oh |
| July | Boeing | seat | go |
| August | B.A.C. | reservation | the |
| September | D.C. | information | there |
| October | Boston | arrive | time |
| November | Chicago | arrival | want |
| December | Denver | depart | make |
| a.m. | Detroit | departure | take |
| p.m. | Douglas | flight | will |
| morning | Lockheed | leave | what |
| afternoon | Los Angeles | return | when |
| evening | Miami | stops | would |
| night | New York | nonstop | like |
| plane | Philadelphia | prefer | many |
| coach | Seattle | please | some |
| phone | Washington | repeat | from |
| number | area | meal | how |
| office | home | served | much |

**Table 3 120-word Airline Vocabulary [28]**

# Chapter 6

## COMPARISON OF RELATIVE PERFORMANCES

### 6.1 Introduction

The four speech recognition techniques implemented were based on the following:

1. Zero-Crossing/Energy Comparison
2. Autocorrelation/LPC Measurement
3. Cepstral Distance Measure
4. LPC Analysis (Ikatura's distance measure)

The results achieved using the first technique are discussed in section 6.2 and since it was not possible to use a large vocabulary (60 or 120 words), these results are not compared directly with the other methods.

The second technique, which used the autocorrelation coefficients of the test utterance and the LPC coefficients of the reference utterance (as defined in equation 5.2), was not found to give acceptable results. The best recognition rate that could be achieved using this technique was little more than 50% for a 30-word vocabulary. Although many attempts were made to improve on this performance, no success was forthcoming. Consequently, this technique was not pursued any further.

The last two techniques listed above both produced excellent results. Before the performances of these two techniques are discussed in section 6.4, the effects of a number of parameters are analyzed. These parameters include the frame length, the overlapping interval of adjacent frames, pre-emphasis, Hamming windows, filter order and dynamic time warping. Since the effect of these six parameters on the performance the last two speech recognition techniques was essentially the same, only one set of results (using LPC coefficients and Ikatura's distance measure) is given to illustrate their effect. These results may be found in section 6.3.

## 6.2 Zero-Crossing/Energy Technique

As pointed out in the previous section, this method of speech recognition cannot be expected to give a high level of accuracy for large vocabularies. It can, however, yield a recognition rate of close to 100% for a carefully chosen small vocabulary. The results that were achieved using this technique on a 10-digit vocabulary are given and some of the factors influencing it's performance are discussed.

### 6.2.1 Band-Pass Filter Specifications

Without any filtering other than that described in section 5.2.3, the recognition rate achieved by this technique was 87.7%. By isolating a narrower band (±800Hz) of the speech signal and using only this for the recognition process, the recognition rate was improved somewhat. This follows from the fact that the number of zero-crossings in a specific frequency band gives a good indication of the predominant frequency within that band. On the other hand, a zero-crossing count for a speech signal with a bandwidth of 3.2kHz gives a rather crude indication since a speech signal is comprised of a number of frequency components.

If the bandpass filter was positioned so that it's low-end cut-off frequency was above 1.7kHz, the performance was found to begin to deteriorate. This was due mainly to end-point detection problems since the 'f' and 's' fricatives at the start of words could not be detected as a result of the filtering. The best performance (92.6%) was achieved when analyzing the frequency content of the speech signal lying between 1.2kHz and 2.0kHz.

### 6.2.2 Effect of the Frame Length

The analysis frames used for this technique were non-overlapping and the effect of the length of this analysis interval is illustrated in figure 6.1. Frame lengths of between 10ms and 25ms all yielded good results whereas for lengths outside of this range, the recognition rate suffered.

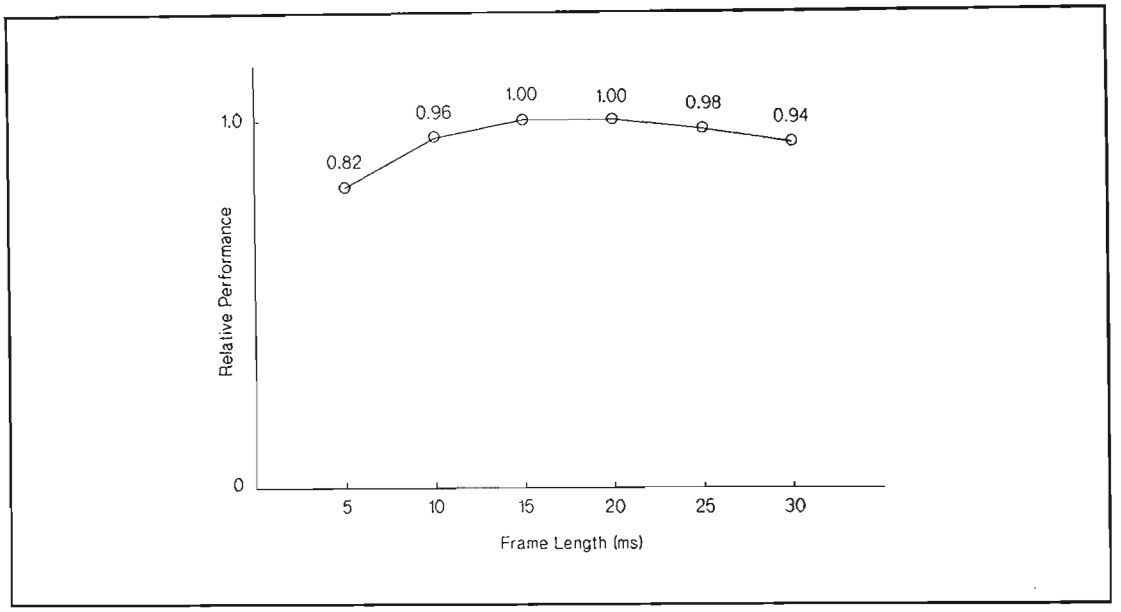



**Fig. 6.1 Effect of Frame Length**

### 6.2.3 Energy Weighting of Distance Measure

The effect due to the inclusion of an energy component into the distance measure used for this technique (equation 5.1) was somewhat disappointing. It did not significantly improve the accuracy of the zero-crossing technique and no value of $\alpha$, the energy weighting factor, could be found such that the performance showed a consistent improvement for all words. It may be concluded from this that the energy information in the speech signal is of little value when implementing a zero-crossing analysis.

### 6.2.4 **Effect of Dynamic Time Warping**

By matching the frames of the unknown word to those of a reference word in a non-linear fashion (in order to take into account the variation in speed with which a word is spoken), the performance of any speech recognition system can be improved. This simple technique was no exception although the improvement was marginal. This limited improvement can be attributed to the fact that practically all the digits from zero to nine are very short and, with the exception of 'seven', monosyllabic. The possibility of saying these words with much time variation is therefore significantly reduced. The best recognition rate of 91.4% obtained using a linear comparison of frames was increased to 92.6% as a result of the dynamic time warping algorithm.

It should be noted that the results given for the zero-crossing technique are those achieved from a set of fifty reference templates - one for each of the ten digits in the test vocabulary.

## 6.3 Parameters to Optimize Performance

The performance of a speech recognition system using one of the last two techniques implemented may be optimized by the following six parameters:

1. Frame length
2. Adjacent-frame overlap interval
3. Pre-emphasis of the speech signal
4. Hamming window
5. Filter order
6. Dynamic Time Warping

As mentioned earlier in this chapter, the effect of the above parameters was essentially the same for the last two speech recognition techniques (ie. the cepstral analysis method and LPC analysis method). A discussion of the effects of each of these parameters on the performance of the speech recognition techniques follows.

One method of evaluating the effects of these parameters would have been to observe the recognition rate. In many cases, however, due to the high recognition rate, the change in this rate was very small. A better method was to compare the distance between the unknown word and the best matching reference word with the distance between the unknown word and the second best reference word. For example, if with a frame length of 45ms the distance between the unknown word and the best matching word is 5.8 (using Ikatura's distance measure) and the distance for the second best match is 43.2, then there is little possibility of this word being incorrectly recognized in future utterances. If now for a frame length of 15ms these distances are 24.3 and 26.1 respectively, then there is a much higher possibility of an error in future utterances of this word.

In this case one could conclude that a frame length of 45ms will give a higher degree of accuracy than a frame length of 15ms.

The above method of comparison was quantified by taking the ratio of the best distance to the second best distance. In the above example this then gives values of 5.55 and 1.07 for frame lengths of 45ms and 15ms respectively. Not only was this method of comparison found to be consistent with the increase or decrease in incorrectly recognized words, it also gave a numerical quantity that changed more noticeably than did the recognition rate.

The results given below correspond to tests using the 60-word vocabulary. The library of reference words for each set of measurements consisted of two templates for each of the 60 words. Each word was then spoken a total of five times and the best and second best distances noted. The ratios were summed to give an average and the normalized results are illustrated graphically.

### 6.3.1 **Frame Length**

The size of the frames was varied from 75 to 900 samples (7.5ms to 90ms for a 10kHz sampling rate). The results are illustrated in Figure 6.2 where it can be seen that a frame length of 45ms was found to yield the best results. For frame lengths of less than 30ms in duration, the distance scores became rather inconsistent and the recognition rate deteriorated.

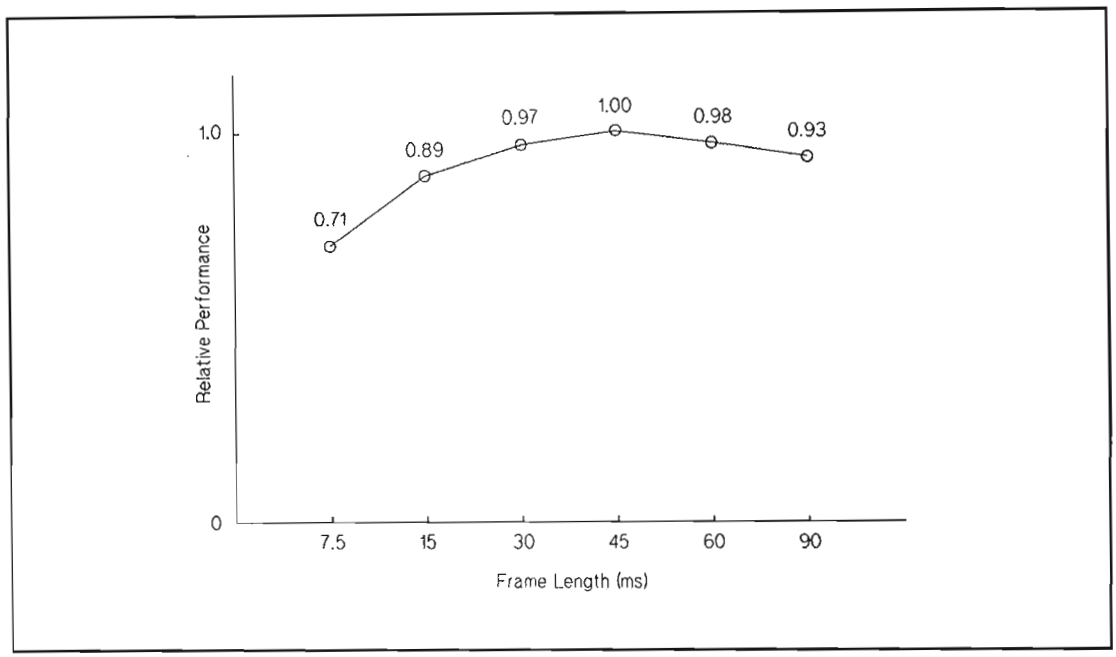



**Fig. 6.2 Effect of Frame Length**

It is interesting to note that the optimum frame length for these two methods is considerably longer than for the zero-crossing method. One reason for this is the fact that for best performance, the duration of the Hamming window used is required to be very long compared to the impulse response of the windowed signal [17].

### 6.3.2 **Adjacent-frame Overlap Interval**

For a frame length of 45ms, the effect of spacing consecutive frames 15ms, 30ms and 45ms apart was observed. Clearly when the spacing is less than 45ms there is an overlap between adjacent frames. Such an overlap provides smoothing between sets of feature coefficients (be they LPC, cepstral or autocorrelation coefficients) and the effects can be seen in Figure 6.3.

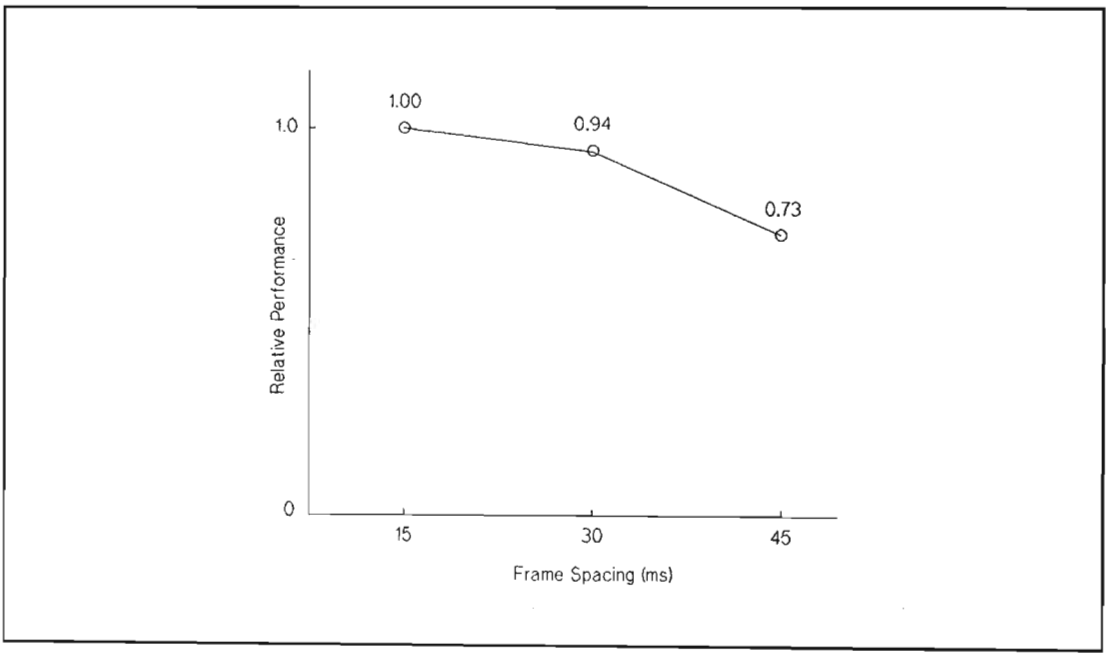



**Fig. 6.3 Effect of Overlapping Frames**

### 6.3.3 **Pre-emphasis and Hamming Windows**

The effects of pre-emphasis and a Hamming window are discussed jointly since the main reason for pre-emphasizing the speech signal is to reduce the dynamic range of the windowed signal. It should be noted that where tests were done without a Hamming window, use was made of a rectangular window function and that where the speech signal was pre-emphasized, use was made of a first-order system with a transfer function $H(z) = 1 - 0.95z^{-1}$.

The use of a Hamming window significantly improved the performance of the LPC and Cepstral methods and pre-emphasizing the speech signal made a slight further improvement to the recognition rate. The relative performance with and without pre-emphasis and a Hamming window is illustrated in figure 6.4.

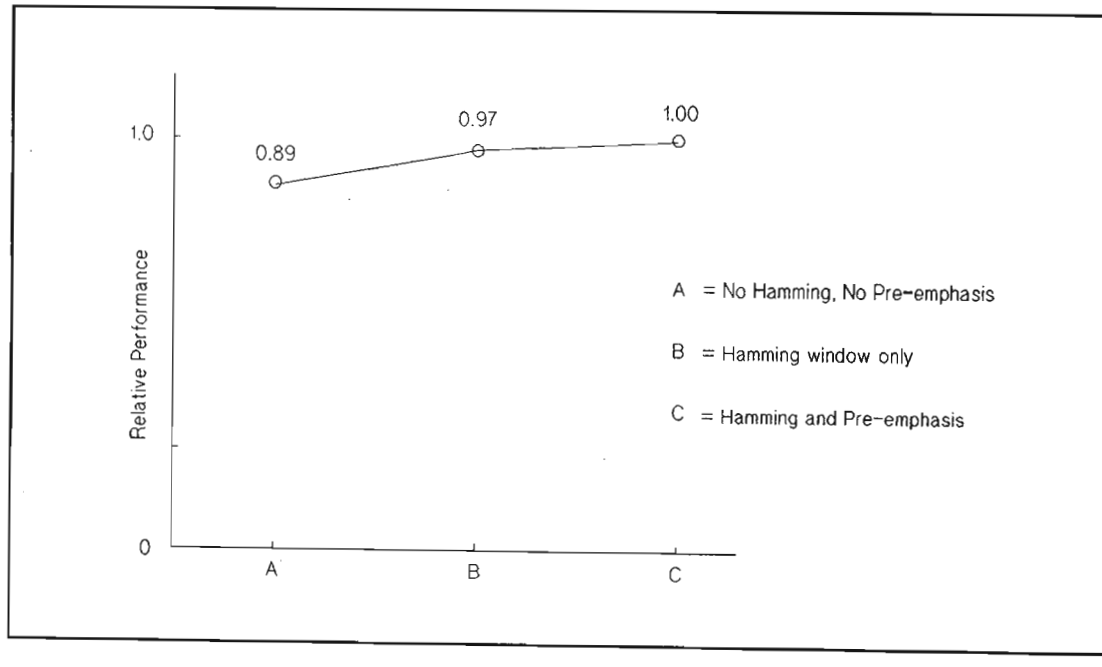



**Fig. 6.4 Effects of a Hamming Window and Pre-emphasis**

### 6.3.4 Filter Order

As expected, the recognition rate improved with increasing filter order (**p**). Bearing in mind that as the order increases, so does the computational load as well as the memory required to store the coefficients, it is necessary to draw the line at some point. The tests to evaluate the effect of **p** were performed on the 120-word vocabulary and the results are given in figure 6.5 as percentages.

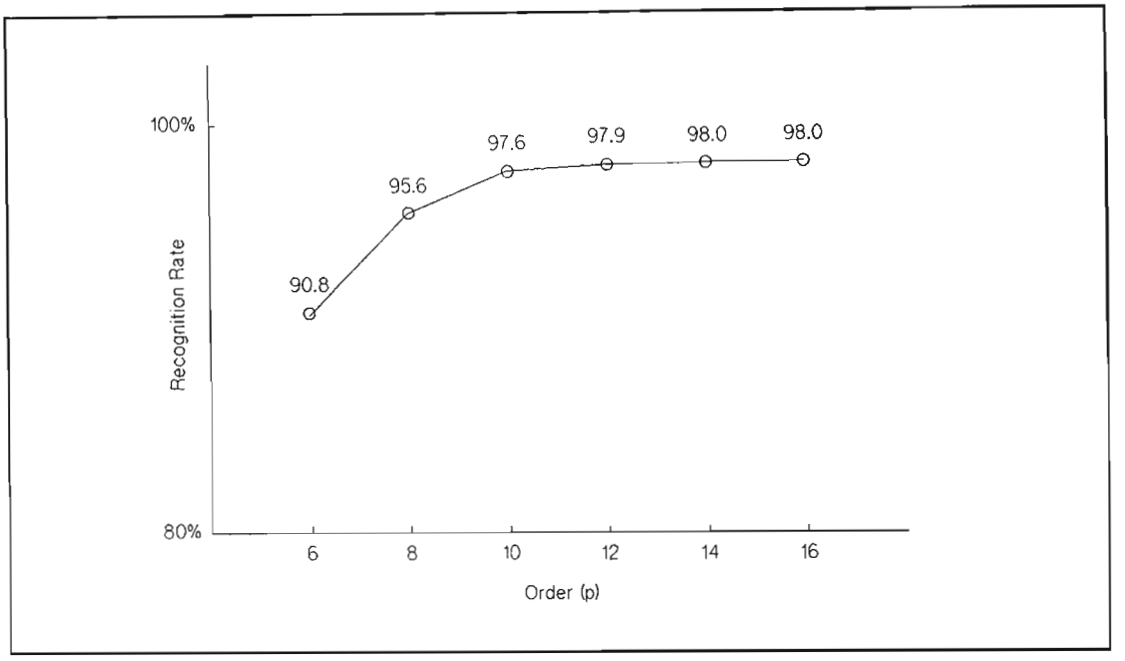



**Fig. 6.5 Effect of Filter Order (p)**

From the above results it is evident that the marginal improvement in performance for p>10 does not warrant the extra time for computation nor the additional memory required (unless, of course, these constraints do not exist). For all measurements during this study, an order of 10 was used for the coefficient calculations.

### 6.3.5 **Dynamic Time Warping**

The recognition rate achieved when using a dynamic time warping algorithm to match the unknown word to the reference word was notably better than that when performing a linear time-alignment of words. The improvement in the word recognition rate for the 120-word vocabulary was approximately 3.8%. In addition, the distance score for the best word was reduced, on average, to about 0.8 of it's original value (ie. the distance without DTW).

## 6.4 Performance Comparisons of Speech Recognition Techniques

The performances of the LPC analysis method and the cepstral analysis method (hereafter referred to as the LPC method and Cepstral method) were examined using the 30, 60 and 120-word vocabularies described in section 5.4. Unfortunately, it was not possible to use a larger vocabulary due to the limitations imposed by the computer used. The tests were carried out by repeating each word in the vocabulary a total of 10-15 times and noting the number of incorrectly recognized utterances.

### 6.4.1 Performance using a 30-word Vocabulary

Figure 6.6 gives the results that were achieved for the LPC and Cepstral methods using the 30-word vocabulary. Initially there was only **one** template for each word contained in the library. In other words, each word to be included in the library was spoken only once during the training procedure and this utterance was used to build the reference template.

The recognition rate achieved by the LPC method of speech recognition for the 30-word vocabulary when using one template per word was 93.6%. It was observed, however, that almost 5% of the 6.4% error rate was made up of the same three words, namely 'if', 'break' and 'subtract'.

By including an additional three templates for each of these three words, the recognition rate was increased to 98.2%. In this case, the 1.8% error rate consisted of random words and could therefore be attributed to variations in word pronunciations.

In order to allow slightly more variation in the manner in which a word was spoken, the number of templates was increased further by including an additional two templates for every word in the vocabulary. The library of reference words now consisted of three templates for each word except the three initial problem words, for which there were now six templates. The desired effect was obtained and the recognition rate increased to an impressive **99.1%** (4 errors out of a total of 450 utterances - 15 repetitions of each word).

The results for the Cepstral method on the above vocabulary were only fractionally lower than those obtained by the LPC method. At the three stages in the development of the above vocabulary, the recognition rates yielded by the Cepstral method were 91.8% with one template per word, 98.0% with additional templates for problem words, and **98.7%** for the final vocabulary (6 errors in 450 utterances).

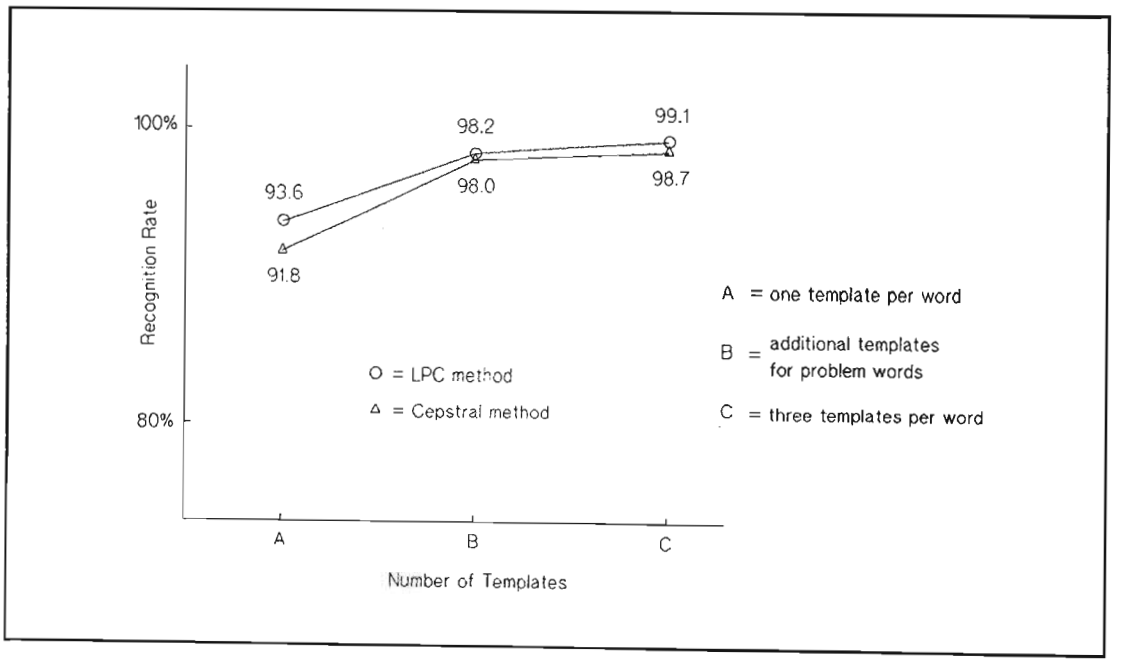



**Fig. 6.6 Performance Comparison using a 30-word Computer Vocabulary**

### 6.4.2 **Performance using a 60-word Vocabulary**

The tests that were performed on the 60-word computer vocabulary were similar to those carried out on the 30-word vocabulary.

After building an initial library consisting of one reference template per word, the performance of each technique (ie. LPC method and Cepstral method) was studied. In view of there being a limitation on the maximum number of reference templates allowed (128 per vocabulary), it was not possible to include additional templates for every word in the vocabulary and then still more for the problem words. The performance when using this size vocabulary was optimized by adding extra templates for only those words that were incorrectly recognized or words that showed a considerable probability of being incorrectly recognized (based on their distance scores). Additional templates were obviously not included for words that were always correctly recognized.

The final library of reference templates was made up as follows: six words were each described by five templates, seven words used three templates each, thirty words each had two templates and the remaining seventeen words were described by only one template. Using this data base, the recognition rate achieved for the Cepstral method was **97.8%** and for the LPC method it was **98.7%**. These figures represent 13 errors and 8 errors respectively out of a total of 600 utterances - 10 repetitions of each word.

The performance improvement as a result of optimizing the set of reference templates as described above was very substantial - the recognition rates for both methods when using only one reference template per word were slightly above 90%.

### 6.4.2 **Performance using a 120-word Vocabulary**

The full potential of the LPC and Cepstral methods was not reached when using the vocabulary of 120 airline terms. This was again due to the 128-template limitation which in this case allowed only eight words to be described by two reference templates.

This problem was lessened, however, by the fact that, as was the case with the first two vocabularies, it was only a few words that contributed towards the majority of the errors. Having constructed a library consisting of one reference template per word, an examination was made of the results achieved and the eight words generating the most errors were isolated. The recognition rate for the initial set of reference templates was 89.3% for the LPC method and 87.0% for the Cepstral method. Furthermore, it was found that these eight words collectively contributed towards approximately 78% of all the errors.

By including an additional template for each of the eight words selected, the recognition rates improved to **94.6%** and **91.7%** for the LPC and Cepstral methods respectively. Based on these improvements and on the results achieved when using the smaller vocabularies, it is the firm belief of the author that if a library consisting of 3-5 reference templates per word had been used, recognition rates very much closer to 100% would have been achieved.

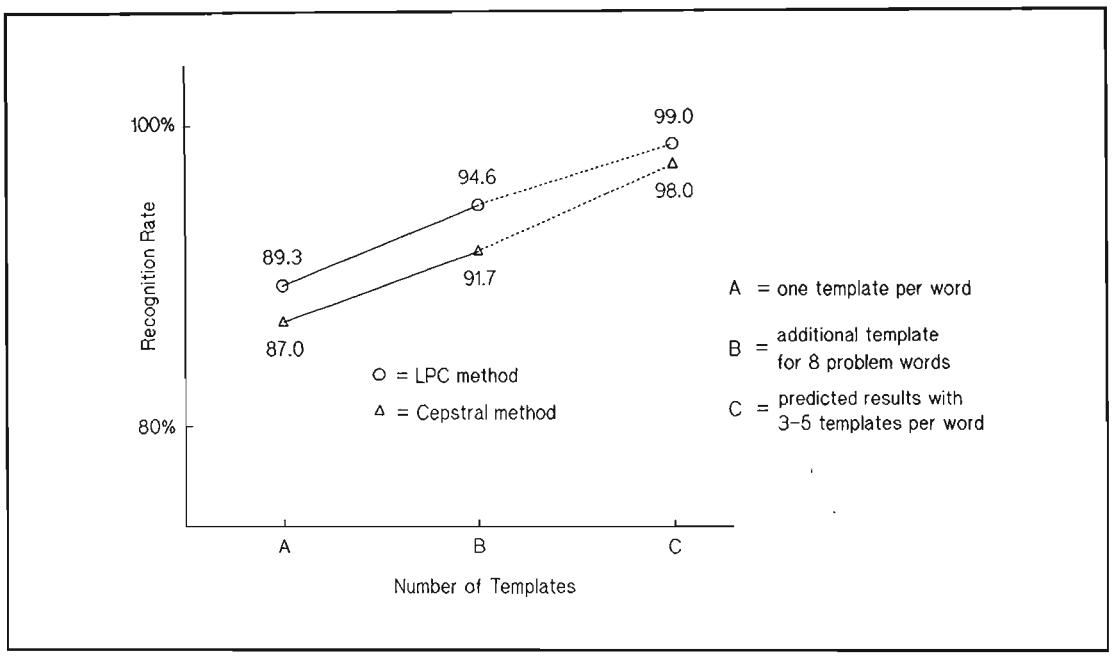



**Fig. 6.7 Performance Comparison using a 120-word Airline Vocabulary**

## 6.5 Comparison of Recognition Times

It has already been mentioned that the zero-crossing technique provided real time performance. For the remaining three techniques, the time taken to recognize a word was made up of two components. The first being the time taken to determine the set of parameters (coefficients) for the unknown spoken word, and the second being the time required to compare these parameters to those belonging to all the words in the current vocabulary. It should be clear that the former time period is approximately equal to the time per word required to train the system.

Table 6.1 details these recognition times (average values for a vocabulary of 120 words on a 12MHz PC-AT computer).

| Recognition Technique | Time to Compute Parameters | Time to Compare Word to Templates | Total Time |
|---|---|---|---|
| Zero-Cross | - | - | - |
| Auto-Corr | 14s | 9s | 23s |
| Cepstral | 17s | 10s | 27s |
| LPC | 16s | 16s | 32s |

**Table 6.1 Comparison of Word Recognition Times**

From the above results it can be seen that Ikatura's distance measure (for LPC coefficients) is the most computationally severe method of word comparison. It should be noted that the time of 17s to determine the cepstral coefficients could possibly be reduced since they were, for the purposes of this study, determined indirectly from the LPC coefficients.

## 6.6 Further Discussion of Results

It has been shown that of the four speech recognition techniques implemented, the one that yielded the best performance was based on an LPC analysis and made use of a distance measure (log likelihood ratio) proposed by Ikatura. This was not too surprising since it is one of the most popular methods currently being used in speech recognition systems.

A point worthy of mentioning is that the increase in error rate was not proportional to the vocabulary size. It is especially encouraging to compare the recognition rate, when using only one template per word, of 89.3% for the 120-word vocabulary with the 93.6% obtained when using the 30-word vocabulary.

# Chapter 7

## CONCLUSION

The subject of speech recognition was studied and most of the speech recognition techniques used to date were investigated. Four speech recognition techniques for isolated-word speaker-dependent applications were implemented and the effects of various parameters influencing their performance were studied experimentally. The performances of the four techniques were also compared.

One of the major problems encountered during the course of this work was the lack of practical literature on the implementation of a speech recognition system. Although there are numerous books that deal with the broad concepts of speech recognition and it's associated problems as well as a large number of papers dealing with specific topics, techniques and the results attained, it was found that very seldom was there much information pertaining to **how** exactly these results had been achieved.

It is therefore hoped that this thesis provides, in one book, a fairly comprehensive description (without too much emphasis on the theory - which may be found in the references given) of the various techniques that may be used in implementing a speech recognition system. In

addition, it was the goal of the author to provide anyone interested in further research on this topic with enough information to readily continue where this work ends. It is for this reason that the software that was developed to implement the speech recognition system capable of achieving the high recognition accuracies described has been included in an appendix.

The simplest technique, one that used zero-crossings and energy in it's recognition process, yielded a recognition rate of only 92.6% on a 10-digit vocabulary. A better performance was not expected and the reason for implementing this technique was to demonstrate that a speech recognition system with a very limited vocabulary can be implemented in real time on an IBM PC. The performance of such a system can be improved considerably by choosing a vocabulary of 10 acoustically very different words as opposed to 10 digits. Such a vocabulary could find use in the control of machinery where only a few commands are required.

The technique that yielded the highest recognition accuracy was one that used LPC coefficients, with Ikatura's distance measure and a filter order of 10. The frames were of 45ms duration and spaced 15ms apart, the speech signal was pre-emphasized and a Hamming window was placed over each frame. The test utterance was matched onto each reference word using a dynamic time warping algorithm. The recognition rate achieved was 98.7% for a 60-word vocabulary. This performance could have been improved even further by the use of additional templates for each word in the vocabulary.

Although, after many years of effort, the goal of a high performance speech recognition system for general use still eludes us, substantial progress has been made. As technology continues to move forward at its formidable pace, so will progress in area of speech recognition. However, performance equal to that of a human will, if ever achieved, remain out of reach for some years to come.

# Appendix A

**TABLE OF PHONETIC SYMBOLS**

The phonetic symbols for the following consonants are the same as their letters in the conventional alphabet: b, d, f, g, h, k, l, m, n, p, r, s, t, v, w, and z. The following table lists the phonemes according to the International Phonetic Alphabet as well as an orthographic representation [24].

| PHONETIC SYMBOLS | ORTHOGRAPHIC REPRESENTATION | PRONOUNCIATION |
|---|---|---|
| /I/ | /I/ | kit |
| /ɛ/ | /E/ | dress |
| /æ/ | /ae/ | trap |
| /ɔ/ | /o/ | cloth |
| /ʌ/ | /A/ | strut |
| /ʊ/ | /U/ | foot |
| /ə/ | /e/ | attain |
| /j/ | /j/ | yes |
| /ŋ/ | /ng/ | bang |
| /i:/ | /i:/ | fleece |
| /a:/ | /a:/ | bath |
| /ɔ:/ | /o:/ | north |
| /ʉ:/ | /u:/ | goose |
| /ɜ:/ | /E:/ | fern |
| /eI/ | /eI/ | pray |
| /aI/ | /aI/ | buy |
| /ɔI/ | /oI/ | choice |
| /aʊ/ | /aU/ | mouth |
| /ʉə/ | /eU/ | loan |
| /Iə/ | /Ie/ | near |
| /ɛə/ | /Ee/ | bear |
| /ʊə/ | /Ue/ | cure |
| /ʃ/ | /sh/ | dish |
| /ɵ/ | /th/ | path |
| /ʒ/ | /zh/ | rouge |
| /tʃ/ | /tsh/ | catch |
| /dʒ/ | /dzh/ | bridge |

**Table of Phonetic Symbols**

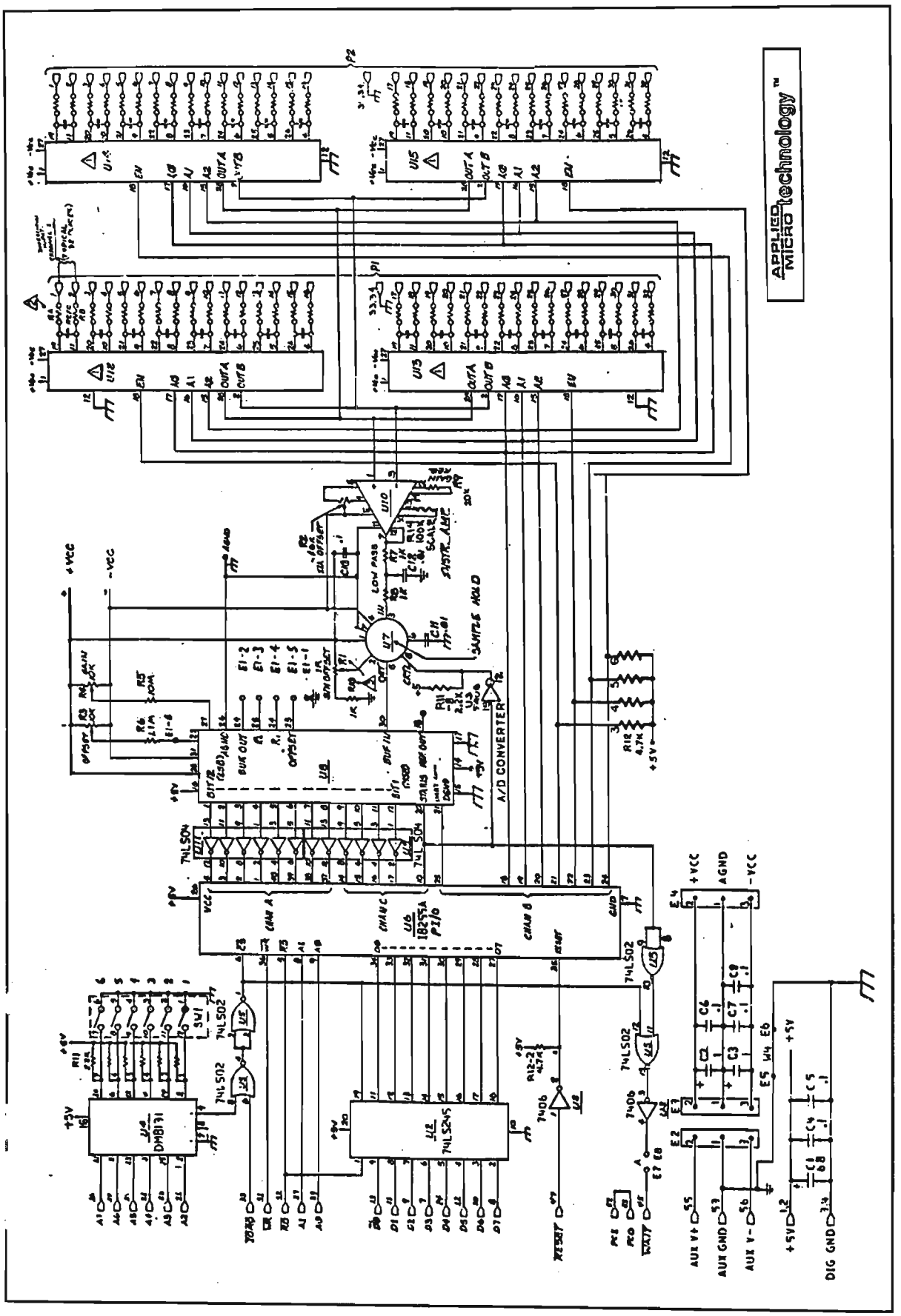
# Appendix B

## ST4303 A/D CONVERTER & INTERFACE CARD

### ST4303 Specifications

The ST4303 is a 32-channel, 12-bit differential input analogue to digital converter made by Applied Micro Technology Inc. for the STD BUS environment.

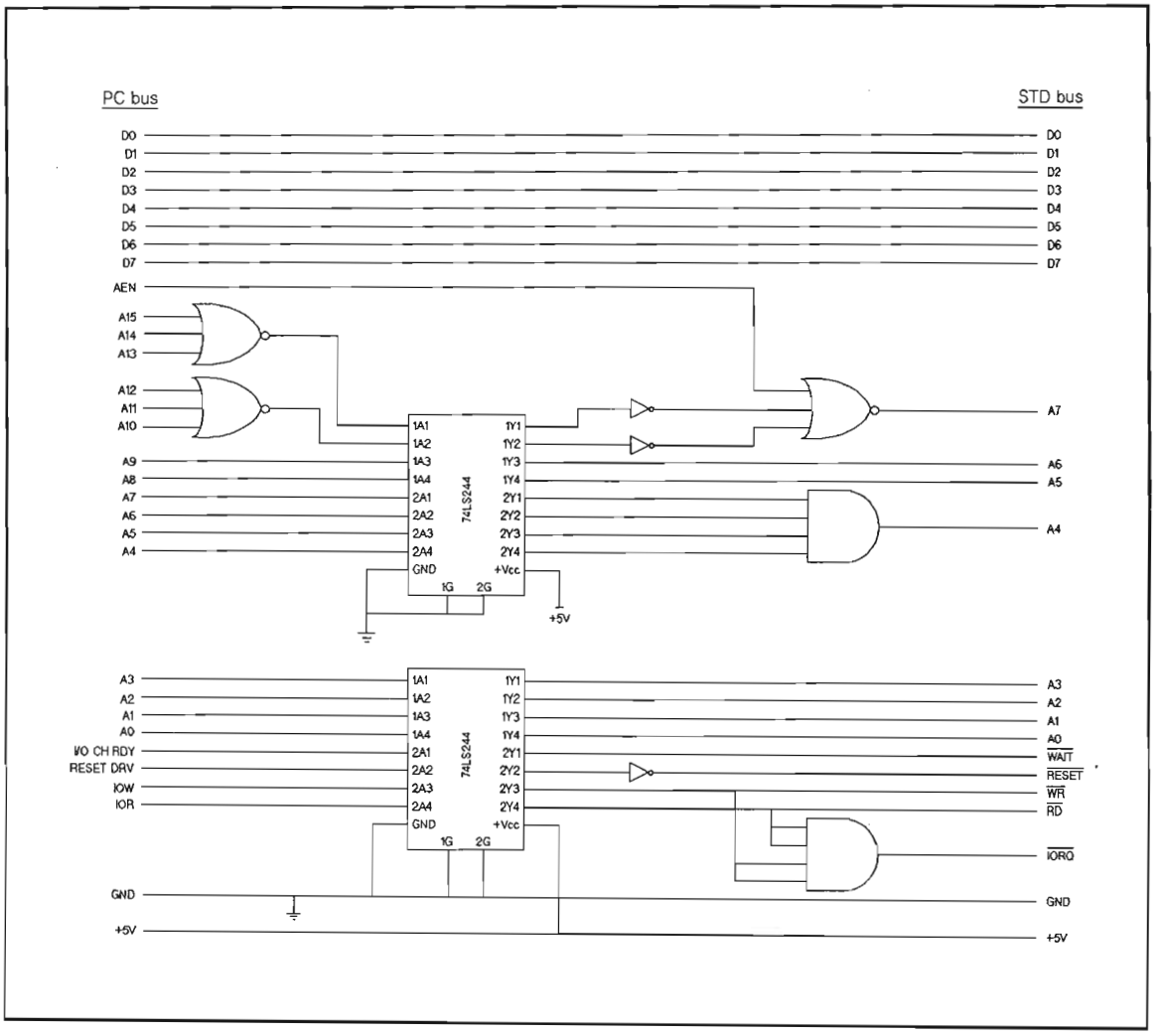
| | |
|---|---|
| INPUT CONNECTOR TYPE | 17 pin .10" x .10"<br>(mates with Ansley 609 3400) |
| A/D CONVERTER TYPE | Burr Brown ADC84KG-12 |
| CMRR | 90dB |
| SETTLING TIME | < 50 $\mu$s |
| CONVERSION TIME | < 20 $\mu$s |
| INPUT IMPEDENCE | 10 G$\Omega$ |
| VOLTAGE REQUIREMENTS | +15V 80 mA<br>-15V 80 mA<br>+5V 100 mA |
| I/O ADDRESSING | Any four consecutive addresses between 00h and FFh |
| CARD DIMENSIONS | 4.5" x 6.5" |
| CARD FORMAT | STD bus |




**Circuit Diagram of ST4303 A/D Board**

## ST4303 to IBM PC Interface Card

The figure below shows the interface circuit that was designed to connect the ST4303 A/D board to an IBM PC. It should be noted that this inteface card is not a general STD bus to PC bus interface since only those signals used by the ST4303 were decoded.

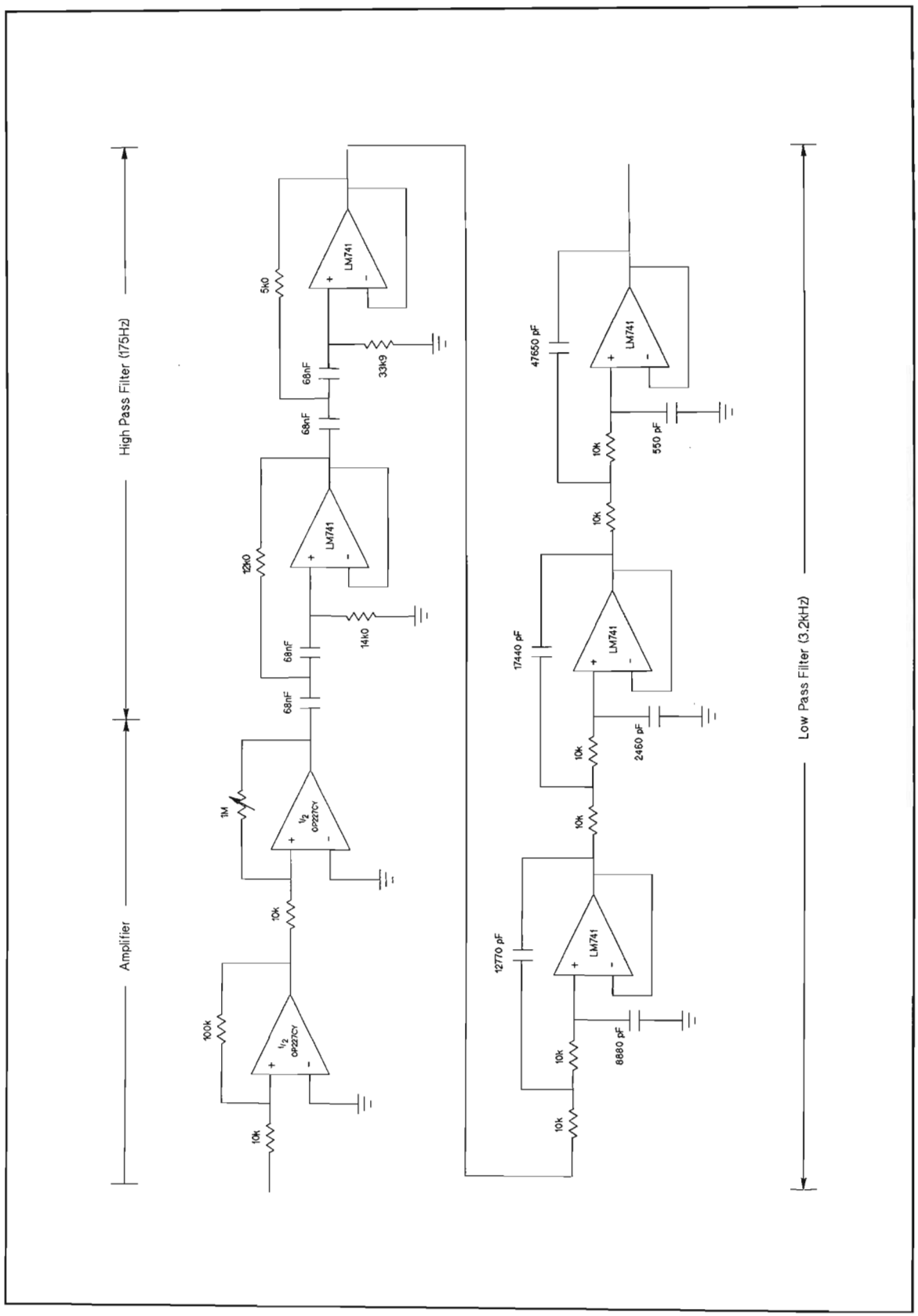



**Circuit for ST4303 to IBM PC Interface**

# Appendix C

**INPUT FILTER AND AMPLIFIER**

The diagram overleaf illustrates the circuit for the amplifier and two filters described in section 5.2.3.




**Circuit for Input Filters and Amplifier**

# Appendix D

**SOFTWARE LISTINGS**

The listings given in this appendix provide the reader with the minimum software required to implement the speech recognition system that was developed for this study. The parameters such as frame length, filter order, endpoint detection thresholds, etc. are set to values that were found to provide the best performance when using Ikatura's distance measure for LPC coefficients. It should be noted that the given program represents only a single system and the user is not given the capability of dynamically changing any of these parameters.

Use is made of 1Mb of expanded memory (EMS), giving a maximum vocabulary size of 128 words. For implementation on a PC without EMS, the program may be modified to use only the memory below 640k, although the maximum vocabulary size (without using disk storage) will be limited to about 60 words.

The first listing is of the header file (BIG_MESS.H) containing all the screen messages. The listing for the main program (BIG_EARS.C) begins on page 139.

```
/*****************************************************************************/
*                                                                           */
*                                BIG_MESS.H                                 */
*                               ------------                                */
*                                                                           */
*            Header file to be included in the BIG_EARS program.            */
*              Contains all the windowed messages and prompts.              */
*                                                                           */
*                                                                           */
*                             by R C PITCHERS                               */
*                                                                           */
/*****************************************************************************/

define   black           0          /* Foreground & Background Colours */
define   blue            1
define   green           2
define   cyan            3
define   red             4
define   magenta         5
define   brown           6
define   light_grey      7

define   dark_grey       8          /* Foreground Colours Only         */
define   light_blue      9
define   light_green     10
define   light_cyan      11
define   light_red       12
define   light_magenta   13
define   yellow          14
define   white           15
define   blink           128

include  "conio.h"


oid  Welcome_Message (void)

  clrscr();
  window (8,4, 72,21);
  textbackground (light_grey);
  clrscr();
  window (10,5, 70,20);
  textcolor (yellow);
  textbackground (red);
  clrscr();

  gotoxy(7,5); cprintf("                    Welcome to the                 ")
  gotoxy(7,6); cprintf("        BIG EARS Speech Recognition System.        ")
  gotoxy(7,8); cprintf("   Please speak clearly and one word at a time.    ")
  textcolor (white);
  gotoxy(7,11);cprintf("      Do you want to Train the system or           ")
  gotoxy(7,12);cprintf("      Load an existing vocabulary (T/L)?           ")
```

```
oid  Training_Message (void)

  window (8,4, 72,21);
  textbackground (red);
  clrscr();
  window (10,5, 70,20);
  textcolor (white);
  textbackground (cyan);
  clrscr();
  gotoxy(7,6); cprintf("In order to train this speech recognition system,")
  gotoxy(7,7); cprintf("each word that is to be included in  the  chosen ")
  gotoxy(7,8); cprintf("vocabulary, must be entered through the keyboard ")
  gotoxy(7,9); cprintf("and then said when prompted.                      ")
  textcolor (yellow);
  gotoxy(7,12);cprintf("                          Press any key to continue...")
  getch();


oid Setup_Main_Screen (void)

  int  n;

  window (1,1, 80,5);
  textcolor (yellow);
  textbackground (red);
  clrscr();
  gotoxy (2, 1);  cprintf ("╔═══════════════════════════════════════");
  gotoxy (40,1);  cprintf ("═══════════════════════════════════════╗");
  gotoxy (2, 5);  cprintf ("╚═══════════════════════════════════════");
  gotoxy (40,5);  cprintf ("═══════════════════════════════════════╝");
  for (n=2 ; n <= 4 ; n++) {
     gotoxy (2, n);  cprintf ("║");
     gotoxy (79,n);  cprintf ("║");
  }
  textcolor (white);
  gotoxy (10, 3); cprintf ("  BIG EARS Speech Recognition System");
  textcolor (yellow);
  cprintf ("          by R C Pitchers");
  window  (1,6, 80,24);
  textbackground (cyan);
  textcolor      (yellow);
  clrscr();
  gotoxy (2,  1);  cprintf ("╔═══════════════════════════════════════");
  gotoxy (40, 1);  cprintf ("═══════════════════════════════════════╗");
  gotoxy (2, 19);  cprintf ("╚═══════════════════════════════════════");
  gotoxy (40,19);  cprintf ("═══════════════════════════════════════╝");
  for (n=2 ; n <= 18 ; n++) {
     gotoxy (2, n);  cprintf ("║");
     gotoxy (79,n);  cprintf ("║");
  }
  window (3,7, 78,23);                          /* Main working window   */
  textbackground (cyan);                        /* 17 lines x 76 columns */
  textcolor (white);
  clrscr();
```

```
oid Clear_Main_Screen (void)

  window (3,7, 78,23);                          /* Main working window   */
  textbackground (cyan);                        /* 17 lines x 76 columns */
  textcolor (white);
  clrscr();


oid  Save_Vocab_Message (void)

  window (14,13, 67,17);
  textcolor (white);
  textbackground (red);
  clrscr();
  gotoxy(2,1);cprintf("
  gotoxy(2,2);cprintf("
  gotoxy(2,3);cprintf("
  gotoxy(2,4);cprintf("
  gotoxy(2,5);cprintf("


oid  Which_Algorithm_Message (void)

  window (15,10, 65,19);
  textcolor (white);
  textbackground (red);
  clrscr();
  gotoxy(2,1); cprintf("                                                ")
  gotoxy(2,2); cprintf("                                                ")
  gotoxy(2,3); cprintf("                                                ")
  gotoxy(2,4); cprintf("                                                ")
  gotoxy(2,5); cprintf("                                                ")
  gotoxy(2,6); cprintf("                                                ")
  gotoxy(2,7); cprintf("                                                ")
  gotoxy(2,8); cprintf("                                                ")
  gotoxy(2,9); cprintf("                                                ")
  gotoxy(2,10);cprintf("                                                ")


oid  Ready_To_Recognize_Message (void)

  window (14,12, 67,18);
  textcolor (white);
  textbackground (red);
  clrscr();
  gotoxy(2,1);cprintf("
  gotoxy(2,2);cprintf("
  gotoxy(2,3);cprintf("
  gotoxy(2,4);cprintf("
  gotoxy(2,5);cprintf("
  gotoxy(2,6);cprintf("
  gotoxy(2,7);cprintf("
```

Do you want to save this vocabulary (Y/N) ?

Which algorithm would you like to use?

1 = LPC Coeffs (Ikatura's Measure)
2 = Cepstral Coefficents
3 = Autocorrelation + LPC Coeffs
4 = Zero-Crossing + Energy Difference

Computer now ready to recognize these words

Press any key to continue...

```
  getch();
  window (3,7, 78,23);                          /* Main working window   */
  textbackground (cyan);                        /* 17 lines x 76 columns */
  textcolor (white);
  clrscr();



oid Was_Input_OK_Message (void)

  window (49, 8, 76, 14);
  textcolor (black);
  textbackground (light_grey);
  clrscr();
  gotoxy (2, 1);  cprintf ("╔══════════════════════════╗");
  gotoxy (2, 2);  cprintf ("║   Are you happy with     ║");
  gotoxy (2, 3);  cprintf ("║  that input, or would    ║");
  gotoxy (2, 4);  cprintf ("║   like to repeat it?     ║");
  gotoxy (2, 5);  cprintf ("║                          ║");
  gotoxy (2, 6);  cprintf ("║    Enter R to repeat     ║");
  gotoxy (2, 7);  cprintf ("╚══════════════════════════╝");
  textcolor (blue);
  gotoxy (7, 6);  cprintf ("Enter R to repeat");



oid Clear_Was_Input_OK_Message (void)

  textcolor (white);
  textbackground (cyan);
  clrscr();
  window (3,7, 78,23);



oid Speak_When_Ready_Message (void)

  window          (57, 19, 77, 23);
  textcolor       (white);
  textbackground (magenta);
  clrscr();
  gotoxy (2, 1);  cprintf ("╔═══════════════╗");
  gotoxy (2, 2);  cprintf ("║   Speak when  ║");
  gotoxy (2, 3);  cprintf ("║    you are    ║");
  gotoxy (2, 4);  cprintf ("║     Ready     ║");
  gotoxy (2, 5);  cprintf ("╚═══════════════╝");



oid Processing_Input_Message (void)

  window          (56, 18, 77, 23);
  textcolor       (white);
  textbackground (blue);
  clrscr();
```

```
  gotoxy (2, 1);  cprintf ("╔═══════════════════════╗");
  gotoxy (2, 2);  cprintf ("║                       ║");
  gotoxy (2, 3);  cprintf ("║     Processing        ║");
  gotoxy (2, 4);  cprintf ("║     Input Word        ║");
  gotoxy (2, 5);  cprintf ("║                       ║");
  gotoxy (2, 6);  cprintf ("╚═══════════════════════╝");


oid Comparing_Input_Message (void)

  window          (51, 17, 77, 23);
  textcolor       (yellow);
  textbackground (blue);
  clrscr();
  gotoxy (2, 1);  cprintf ("╔═══════════════════════╗");
  gotoxy (2, 2);  cprintf ("║                       ║");
  gotoxy (2, 3);  cprintf ("║   Comparing Input     ║");
  gotoxy (2, 4);  cprintf ("║  Word to Library of   ║");
  gotoxy (2, 5);  cprintf ("║   Reference Words     ║");
  gotoxy (2, 6);  cprintf ("║                       ║");
  gotoxy (2, 7);  cprintf ("╚═══════════════════════╝");


oid Working_Message (int Background_Color)

  window          (57, 19, 77, 23);
  textcolor       (white);
  textbackground (Background_Color);
  clrscr();
  gotoxy (2, 1);  cprintf ("╔═══════════════════╗");
  gotoxy (2, 2);  cprintf ("║                   ║");
  gotoxy (2, 3);  cprintf ("║   Working...      ║");
  gotoxy (2, 4);  cprintf ("║                   ║");
  gotoxy (2, 5);  cprintf ("╚═══════════════════╝");


oid Clear_Little_Message (void)

  textbackground (cyan);
  textcolor (white);
  clrscr();
  window (3,7, 78,23);


oid Reset_Screen (void)

  textcolor (white);
  textbackground (black);
  window (1,1, 80,25);
  clrscr();
```

```
/*****************************************************************************/
*                                                                           */
*                                BIG_EARS.C                                 */
*                               ------------                                */
*                                                                           */
*          Speech recognition system implementing four different            */
*                        recognition techniques.                            */
*                                                                           */
*         Use is made of LPC coefficients, Cepstral coefficients,           */
*         Autocorrelation coefficients, energy and zerocrossings.           */
*                                                                           */
*                                                                           */
*                             by R C PITCHERS                               */
*                                                                           */
/*****************************************************************************/



#define  FRAMESIZE    450                  /* No. samples in one full frame   */
#define  OVERLAP      3                    /* Adjacent frame overlap          */
#define  SFRAMESIZE   FRAMESIZE/OVERLAP    /* Small frame size to find end pts*/
#define  ORDER        10                   /* Filter order for LPC coeficients*/
#define  E_LOWER      SFRAMESIZE*1000L     /* Lower energy threshold level    */
#define  E_UPPER      SFRAMESIZE*20000L    /* Upper energy threshold level    */
#define  ZCTHRESH     SFRAMESIZE/5         /* Zero crossing threshold level   */
#define  ZCNUMTHR     3    /* No. times ZCTHRESH must be exceeded             */
#define  CONSECTHR    4    /* No. times to consecutively exceed ZCTHRESH      */
#define  BEGFRAMES    10   /* No. frames to check prior to initial beg pnt */
#define  ENDFRAMES    70   /* No. frames to sample after initial begin pnt */
#define  MAXWORDS     128                    /* Max no. of words in library   */
#define  MAXFRAMES    BEGFRAMES+ENDFRAMES /* Max no. of small frames per wd*/
#define  BEGSAMPLES   BEGFRAMES*SFRAMESIZE
#define  ENDSAMPLES   ENDFRAMES*SFRAMESIZE
#define  MAXSAMPLES   MAXFRAMES*SFRAMESIZE
#define  BUFFERSIZE   (MAXFRAMES-OVERLAP+1)*(ORDER+1)
#define  PI           3.1415926536

#define  REG0         0x200           /* Register contains least sig. byte  */
#define  REG1         0x201           /* Selects channel + MSB starts conv. */
#define  REG2         0x202           /* Holds 4 MSBs of data + EOC bit     */
#define  REG3         0x203           /* Initialize by sending 99h to REG3  */

#define  INIT         0x99            /* Initialize A/D board               */
#define  START        0x88            /* Select channel + start command     */
#define  RESET        0x08            /* Reset start conversion bit         */
```

```
;----------------------------------------*/
*  Define prototypes for function calls.  */
*----------------------------------------*/

include  "dos.h"
include  "math.h"
include  "float.h"
include  "stdlib.h"
include  "stdio.h"
include  "conio.h"
include  "fcntl.h"
include  "io.h"
include  "alloc.h"

include  <BIG_MESS.H>


oid    Delay              (int Del);
oid    Sample_Word        (int Mode);
oid    Train_System       (void);
oid    Recognize_Word     (void);
oid    Modify_Vocabulary  (void);
oid    Save_Vocabulary    (void);
oid    Load_Vocabulary    (void);
ouble  Frame_Distance     (int UnknownFrame, int RefFrame);
oid    Allocate_Library_Memory (void);

nt          Alloc_EMS  (int NumPages);
oid         Free_EMS   (int Handle);
ouble far   *Map_EMS   (int Handle, int WordNum);


*----------------------------*/
*  Declare global variables.  */
*----------------------------*/

louble Hamming_Lookup[FRAMESIZE]; /* Lookup table used for Hamming Window */
.nt    NumFrames[MAXWORDS];       /* No. of frames in each word in libary */
:har   WordNames[MAXWORDS][20];   /* Arrays each holding the word name    */
.nt    WordTotal;                 /* Total number of words in library     */
.nt    Length;                    /* Number of short frames in input word */
.nt    Num_Frames;                /* No. overlapping frames in input word */
.nt    Zero;                      /* True A/D offset for zero volts.      */
.nt    ZC_Zero;                   /* Adjusted offset used to find ZCs     */

.nt    EMS_Handle;                     /* Handle used for EMS allocation */
.nt    Distance_Type;                  /* Current distance type selected */
.nt    Start_Frame, End_Frame;         /* 2 frames marking word end pnts */

.nt    DTW_Flag;                       /* Flags used to indicate whether */
.nt    Hamming_Flag;                   /* or not to use DTW, a Hamming   */
.nt    Preemphasis_Flag;               /* window and pre-emphasis.       */
```

```
.nt     zcross;                              /* Zero crossings of one frame    */
.nt     ZC_Buffer[MAXFRAMES];                /* Zero-crossings of input word   */
.nt     Ref_ZC_Buffer[MAXFRAMES];            /* Temp buffer for one ref word   */
.nt     Temp_ZC_Buffer[BEGFRAMES];           /* Temp buffer to find begin pnt. */
.nt     far *ZC_Library_Addr[MAXWORDS];      /* Ptrs to library of z-crossings */
.nt     far *ZC_Library_Ptr;

.nsigned long  energy;                       /* Total energy content of 1 frame*/
.nsigned long  E_Buffer[MAXFRAMES];          /* Energy of input word           */
.nsigned long  Ref_E_Buffer[MAXFRAMES];      /* Temp buffer for one ref word   */
.nsigned long  Temp_E_Buffer[BEGFRAMES];     /* Temp buffer to find begin pnt. */
.nsigned long  far *E_Library_Addr[MAXWORDS]; /* Ptrs to energy library.       */
.nsigned long  far *E_Library_Ptr;

.louble Ref_Coeff[MAXFRAMES-OVERLAP+1][ORDER+1]; /* Temp buffer for 1 ref word
.louble LPC_Coeff[MAXFRAMES-OVERLAP+1][ORDER+1]; /* LPC coefficients of inp wd
.louble Cepstral [MAXFRAMES-OVERLAP+1][ORDER+1]; /* Cepstral coeffs  of inp wd
.louble Auto_Corr[MAXFRAMES-OVERLAP+1][ORDER+1]; /* Auto-corr coeffs of inp wd

.louble far *Log_Power_Addr[MAXWORDS];       /* Ptrs to library of Log powers  */
.louble far *Log_Power_Ptr;                  /* used for Ikatura's measure only*/
.louble Log_Power[MAXFRAMES-OVERLAP+1];      /* Log powers for one word.       */

.louble far *Coeff_Ptr;
.nt     far *Sample_Ptr;
.nt     far *Sample_Buffer_Start_Addr1;
.nt     far *Sample_Buffer_Start_Addr2;
.nt     far *True_Start_Address;

.nt     far *Temp_Sample_Ptr;
.nt     far *Temp_Sample_Buffer_Start_Addr;

.nt     Global_Int_Dummy;
.nt     Global_Int_Dummy_Buffer[MAXFRAMES];

.nsigned long  Global_Long_Dummy;
.nsigned long  Global_Long_Dummy_Buffer[MAXFRAMES];
```

```
*****************************************************************/
*                    Main program starts here.                  */
*****************************************************************/

oid  main (void)


  int     c, n, p, k, i, j, w;
  int     data;
  long    Zero_Sum;


 /*-------------------------------------------------------------*/
 /*  Allocate memory space in the far heap for the buffer to hold the    */
 /*  samples of one word. Also for the temporary buffer holding samples  */
 /*  until the preliminary starting point is found.                      */
 /*-------------------------------------------------------------*/

  Sample_Buffer_Start_Addr1 = farcalloc (MAXSAMPLES, sizeof(int));
  if (Sample_Buffer_Start_Addr1 == NULL) {
     printf("\n\nNot enough memory in far heap for sample buffer");
     getch(); return;
  }
  Sample_Buffer_Start_Addr2 = Sample_Buffer_Start_Addr1 + BEGSAMPLES;

  Temp_Sample_Buffer_Start_Addr=farcalloc (BEGSAMPLES, sizeof(int));
  if (Temp_Sample_Buffer_Start_Addr == NULL) {
     printf("\n\nNot enough memory in far heap for temporary sample buffer"
     getch(); return;
  }


 /*---------------------------*/
 /*  Initialize the A/D Board  */
 /*---------------------------*/

  outportb (REG3, INIT);
  outportb (REG1, START);
  outportb (REG1, RESET);


 /*-------------------------------------------------------------*/
 /* Setup a lookup table used for doing the Hamming window calculation  */
 /* The contents are                                                    */
 /*                                                                     */
 /*          0.54  -  0.46 * cos (n * 2 * pi / (FRAMESIZE-1))           */
 /*                                                                     */
 /* Remember that a window is placed over n frames of SFRAMESIZE samples */
 /* where n = OVERLAP                                                   */
 /*-------------------------------------------------------------*/

  for (n=0 ; n < FRAMESIZE ; n++)
     Hamming_Lookup[n] = 0.54 - 0.46 * cos (n * 2 * PI / (FRAMESIZE-1));
```

```
/*-------------------------------------------------------------------*/
/*  Determine the digital value of zero by averaging 10000 samples.  */
/*  Store this value, to be used as A/D offset, in a variable Zero.  */
/*  Set ZC_Zero to be 10 units higher.                               */
/*-------------------------------------------------------------------*/

 Zero_Sum = 0;
 for (n=0 ; n < 10000 ; n++) {
    outportb (REG1, START);              /* Select channel + start comand */
    outportb (REG1, RESET);              /* Reset start conversion bit    */
    inportb (REG2);                      /* Get EOC bit (bit 7 of REG2)   */
    while ((_AL = _AL & 0x80) != 0)      /* Wait until conv. complete     */
       inportb (REG2);
    inportb (REG2);                      /* Get most significant 4 bits   */
    _AH = _AL;                           /* Move them into AH             */
    inportb (REG0);                      /* Get lower 8 bits into AL      */
    data = _AX & 0x0FFF;                 /* Set top 4 bits of word to zero*/
    Zero_Sum = Zero_Sum + data;
    Delay (25);                          /* Delay to set sample frequency */
 }
 Zero_Sum = Zero_Sum / 10000;
 Zero     = Zero_Sum;
 ZC_Zero  = Zero + 9;


/*-------------------------------------------------------------------*/
/*  Display an introductory message and ask whether the user wants   */
/*  to train the system or load an existing vocabulary.              */
/*-------------------------------------------------------------------*/

 Welcome_Message();

 GetInput1:
 if ((c=getch())=='T' || c=='t') {
    Train_System();                      /* Call training procedure       */
    Clear_Main_Screen();
    Save_Vocab_Message();                /* Do you want to save this vocab */
    if ((c=getch())=='Y' || c=='y')      /* If so, call saving procedure  */
       Save_Vocabulary();
 }
 else if (c=='L' || c=='l') {
    Setup_Main_Screen();
    Load_Vocabulary();
    Clear_Main_Screen();
    gotoxy (10, 5);  cprintf ("Do you want to modify this vocabulary? ");
    if ((c=getch())=='Y' || c=='y') {
       Modify_Vocabulary();
       Clear_Main_Screen();
       Save_Vocab_Message();
       if ((c=getch())=='Y' || c=='y')
          Save_Vocabulary();
    }
 }
 else                                    /* If input not T or L then  */
    goto GetInput1;                      /* get another input char    */
```

```
Clear_Main_Screen();
GetInput2:
gotoxy (5, 3);
cprintf (" Do you want to use Dynamic Time Warping (Y/N) ?");
if ((c=getch())=='Y' || c=='y')
   DTW_Flag = 1;
else if (c=='N' || c=='n')
   DTW_Flag = 0;
else
   goto GetInput2;
GetInput3:
Clear_Main_Screen();
Which_Algorithm_Message();
if ((c=getch())=='1')
   Distance_Type = 1;
else if (c=='2')
   Distance_Type = 2;
else if (c=='3')
   Distance_Type = 3;
else if (c=='4')
   Distance_Type = 4;
else
   goto GetInput3;
Clear_Main_Screen();

/*-------------------------------------------------------------*/
/*  If using Cepstral coefficients then convert all the LPC    */
/*  coefficients in the library to Cepstral coefficients.      */
/*-------------------------------------------------------------*/

 if (Distance_Type == 2) {
    Working_Message (light_grey);
    for (w=0 ; w < WordTotal ; w++) {
       Coeff_Ptr = Map_EMS (EMS_Handle, w);      /* Transfer one word of */
       for (n=0 ; n < NumFrames[w] ; n++) {      /* LPC coeffs from lib  */
           for (p=0 ; p <= ORDER ; p++)          /* to current data seg. */
              LPC_Coeff[n][p] = *Coeff_Ptr++;
       }
       for (n=0 ; n < NumFrames[w] ; n++) {      /* Convert LPC coeffs   */
          Cepstral[n][1] = LPC_Coeff[n][1];      /* to Cepstral coeffs.  */
          for (p=2 ; p <= ORDER ; p++) {
              Cepstral[n][p] = LPC_Coeff[n][p] * p;
              for (k=1 ; k < p ; k++)
                  Cepstral[n][p] -= LPC_Coeff[n][k] * Cepstral[n][p-k];
          }
          for (p=2 ; p <= ORDER ; p++)
              Cepstral[n][p] = Cepstral[n][p] / p;
       }
       Coeff_Ptr = Map_EMS (EMS_Handle, w);      /* Store Cepstral coefs */
       for (n=0 ; n < NumFrames[w] ; n++) {      /* in library.          */
          for (p=0 ; p <= ORDER ; p++)
             *Coeff_Ptr++ = Cepstral[n][p];
       }
    }
    Clear_Main_Screen();
 }
```

```
/*-------------------------------------------------------------------*/
/*  If using Ikatura's measure then convert all the LPC coefficients  */
/*  to the modified coefficients and calculate the log power constant. */
/*-------------------------------------------------------------------*/

 if (Distance_Type == 1) {

    Working_Message (light_grey);
    for (w=0 ; w < WordTotal ; w++) {

       Coeff_Ptr = Map_EMS (EMS_Handle, w);      /* Transfer one word of */
       for (n=0 ; n < NumFrames[w] ; n++) {      /* LPC coeffs from lib  */
           for (p=0 ; p <= ORDER ; p++)          /* to current data seg. */
              LPC_Coeff[n][p] = *Coeff_Ptr++;
       }
       for (n=0 ; n < NumFrames[w] ; n++) {      /* Convert LPC coeffs   */
                                                 /* to modified coeffs.  */
          Log_Power[n] = 0;
          for (i=0 ; i <= ORDER ; i++)
              Log_Power[n] += LPC_Coeff[n][i] * LPC_Coeff[n][i];

          Ref_Coeff[n][0] = 1;
          for (i=1 ; i <= ORDER ; i++) {
              Ref_Coeff[n][i] = 0;
              for (j=0 ; j <= ORDER-i ; j++)
                  Ref_Coeff[n][i] += LPC_Coeff[n][j] * LPC_Coeff[n][j+i];
              Ref_Coeff[n][i] = 2 * Ref_Coeff[n][i] / Log_Power[n];
          }
          Log_Power[n] = log (Log_Power[n]);

       }

       Coeff_Ptr = Map_EMS (EMS_Handle, w);      /* Store modified coefs */
       Log_Power_Ptr = Log_Power_Addr[w];        /* and log power in lib */
       for (n=0 ; n < NumFrames[w] ; n++) {
          *Log_Power_Ptr++ = Log_Power[n];
          for (p=0 ; p <= ORDER ; p++)
             *Coeff_Ptr++ = Ref_Coeff[n][p];
       }
    }
    Clear_Main_Screen();

 }
```

```
/*-------------------------------------------------------*/
/*  Go into an infinite loop, recognizing spoken words.  */
/*  Hit any key to quit.                                 */
/*-------------------------------------------------------*/

 Ready_To_Recognize_Message();
 while (1) {

    if (!kbhit())
       Recognize_Word();
    else {
       getch();
       Clear_Main_Screen();
       gotoxy (10,6); cprintf ("Do you want to quit? ");
       if ((c=getch())=='Y' || c=='y') {
          for (w=WordTotal-1 ; w >= 0 ; w--)
             farfree (Log_Power_Addr[w]);
          for (w=WordTotal-1 ; w >= 0 ; w--)
             farfree (E_Library_Addr[w]);
          for (w=WordTotal-1 ; w >= 0 ; w--)
             farfree (ZC_Library_Addr[w]);
          farfree  (Temp_Sample_Buffer_Start_Addr);
          farfree  (Sample_Buffer_Start_Addr1);
          Free_EMS (EMS_Handle);
          Reset_Screen();
          return;
       }
    }
 }
```

```
'*************************************************************************/
'*  Procedure to sample one word (after determining the endpoints).     */
'*  Calculate Autocorrelation coefficients, LPC coeffs and Cepstral     */
'*  coeffs - depending on the value of MODE passed to the procedure.    */
'*  If called by training procedure, then Mode = 0.                     */
'*  If called by recognition procedure, then Mode = Distance_Type.      */
'*************************************************************************/

/oid  Sample_Word (int Mode)

{
   int      c, n, m, f, k, p, M;   /* Temporary counter variables          */
   int      err = 0;               /* Flag to check for singualar matrices */
   int      P_Start;
   int      ZC_Count;              /* Count no. times ZCTHRESH is exceeded */
   double   WSamps[FRAMESIZE];     /* Full-size frame of windowed samples  */
   double   Refl_Coeff[ORDER+1];   /* Reflection coefficients of one frame */
   double   Sum, Residual, Temp;

   int      Local_Dummy;
   int      data, value;
   int      Sign, PSign;
   long     long_value;

   Repeat:
   zcross = 0;                             /* Initialize z-crossing counter */
   energy = 0;                             /* Initialize energy to zero     */

   for (n=0 ; n < MAXFRAMES ; n++) {
      ZC_Buffer[n] = 0;
      E_Buffer [n] = 0;
   }
   for (n=0 ; n < BEGFRAMES ; n++) {
      Temp_ZC_Buffer[n] = 0;
      Temp_E_Buffer [n] = 0;
   }

   Speak_When_Ready_Message();
   ZC_Count = 0;

  Temp_Sample_Ptr = Temp_Sample_Buffer_Start_Addr;
  for (f=0, m=0, n=0 ;    ; m++) {

    if (m == SFRAMESIZE) {                 /* If at the end of a frame,     */
                                           /* then save the zero-crossing   */
       Temp_ZC_Buffer[f] = zcross;         /* count and energy content of   */
       Temp_E_Buffer[f]  = energy;         /* the frame.                    */

       if (zcross >= ZCTHRESH) {           /* If no. of zero crossings is > */
          ZC_Count++;                      /* ZCTHRESH then increment count */
          if (ZC_Count >= CONSECTHR) {     /* If ZC_Count > consecutive     */
             P_Start = f;                  /* threshold, then stop.         */
             break;
          }
       }
```

```
      else                                    /* Else if zcross not > threshold*/
         ZC_Count = 0;                        /* then reset ZC_Count           */

      if (energy >= E_UPPER) {                /* If energy > upper threshold   */
         P_Start = f;                         /* then stop.                    */
         break;
      }

      zcross = 0;
      energy = 0;
      m=0;
      f++;
      if (f == BEGFRAMES) {
          Temp_Sample_Ptr = Temp_Sample_Buffer_Start_Addr;
          f = 0;
      }
   }

   else {
      Global_Int_Dummy_Buffer[f]  = zcross;/* This dreadful piece of    */
      Global_Long_Dummy_Buffer[f] = energy;/* code attempts to keep the */
      Global_Int_Dummy  = 0;               /* sample frequency constant */
      Global_Long_Dummy = 0;               /* at the frame boundaries.  */
      Local_Dummy = 0;                     /* ie. So that there is no   */
      Local_Dummy++;                       /* period without sampling   */
   }

   outportb (REG1, START);                 /* Select channel+start comand*/
   outportb (REG1, RESET);                 /* Reset start conversion bit */
   inportb (REG2);                         /* Get EOC bit (bit 7 of REG2 */
   while ((_AL = _AL & 0x80) != 0)         /* Wait until conv. complete  */
     inportb (REG2);

   inportb (REG2);                         /* Get most significant 4 bits*/
   _AH = _AL;                              /* Move them into AH          */
   inportb (REG0);                         /* Get lower 8 bits into AL   */
   data  = _AX & 0x0FFF;                   /* Set top 4 bits of word to 0*/
   value = data - Zero;                    /* Subtract the A/D zero      */
   *Temp_Sample_Ptr++ = value;             /* Save the sampled value     */
   if (data < ZC_Zero)
     Sign = 0;                             /* Sign = 0 if value negative */
   else
     Sign = 1;                             /* Sign = 1 if value positive */
   if (n == 0)
     n++;                                /* If first sample, dont compare */
   else if (Sign != PSign)               /* If sign not equal to previous */
     zcross++;                           /* sign, then increment zcross   */
   PSign = Sign;                         /* Previous sign = present sign  */
   long_value = value;                   /* Convert value to long integer */
   energy += long_value * long_value;    /* energy = energy + value*value */

   Delay (25);                           /* Short delay to set sample freq*/

}
```

```
/*-------------------------------------------------------------------*/
/* Preliminary start of word has been found. Now sample for +- 0.8 secs */
/*-------------------------------------------------------------------*/

 Sample_Ptr = Sample_Buffer_Start_Addr2;

 for (f=BEGFRAMES, m=0, n=0 ; n < ENDSAMPLES ; n++, m++) {

   if (m == SFRAMESIZE) {                  /* If at the end of a frame,   */
      ZC_Buffer[f] = zcross;               /* then save the zero-crossing */
      E_Buffer[f]  = energy;               /* count and energy content of */
      zcross = 0;                          /* the frame.                  */
      energy = 0;
      m=0;
      f++;
   }

   else {
      Global_Int_Dummy_Buffer[f]  = zcross;
      Global_Long_Dummy_Buffer[f] = energy;
      Global_Int_Dummy  = 0;
      Global_Long_Dummy = 0;
      Local_Dummy = 0;
      Local_Dummy++;
   }

   outportb (REG1, START);                 /* Select channel + start comand */
   outportb (REG1, RESET);                 /* Reset start conversion bit    */
   inportb (REG2);                         /* Get EOC bit (bit 7 of REG2    */
   while ((_AL = _AL & 0x80) != 0)         /* Wait until conv. complete     */
     inportb (REG2);

   inportb (REG2);                         /* Get most significant 4 bits   */
   _AH = _AL;                              /* Move them into AH             */
   inportb (REG0);                         /* Get lower 8 bits into AL      */
   data  = _AX & 0x0FFF;                   /* Set top 4 bits of word to zero*/
   value = data - Zero;                    /* Subtract the A/D zero         */
   *Sample_Ptr++ = value;                  /* Save the sampled value        */
   if (data < ZC_Zero)
     Sign = 0;                             /* Sign = 0 if value negative    */
   else
     Sign = 1;                             /* Sign = 1 if value positive    */
   if (Sign != PSign)                      /* If sign not equal to previous */
     zcross++;                             /* sign, then increment zcross   */
   PSign = Sign;                           /* Previous sign = present sign  */
   long_value = value;                     /* Convert value to long integer */
   energy += long_value * long_value;      /* energy = energy + value*value */

   Delay (25);

 }

 Clear_Little_Message();
```

```
/*-------------------------------------------------------------------*/
/* Find end of word by searching backwards until energy < threshold.  */
/* Then search forward from this point to see if zero-crossing rate   */
/* is above the threshold value. Save the end frame number.           */
/*-------------------------------------------------------------------*/

 f = MAXFRAMES-1;
 while (E_Buffer[f]  < E_LOWER  && f > BEGFRAMES)
    f--;
 while (ZC_Buffer[f] > ZCTHRESH && f < MAXFRAMES)
    f++;
 End_Frame = f-1;

 if (End_Frame < BEGFRAMES + ENDFRAMES/20)  /* Anything this short must */
    goto Repeat;                            /* be a mouth noise.        */


/*-------------------------------------------------------------------*/
/*  The preliminary beginning of the word was where the energy exceeded */
/*  the upper threshold value or the zero-crossing count consecutively  */
/*  exceeded the threshold the required number of times. 0.2 seconds of */
/*  samples prior to this point are stored in a circular buffer. Trans- */
/*  fer these samples to the main buffer and then find where the energy */
/*  first exceeded the lower threshold and where the zero-crossing rate */
/*  was first greater than its threshold. Update the starting point     */
/*  accordingly.                                                        */
/*-------------------------------------------------------------------*/

 Sample_Ptr = Sample_Buffer_Start_Addr2;
 Temp_Sample_Ptr = Temp_Sample_Buffer_Start_Addr + (P_Start+1)*SFRAMESIZE;

 for (n=0 ; n < BEGSAMPLES ; n++) {
     Sample_Ptr--;
     Temp_Sample_Ptr--;
     *Sample_Ptr = *Temp_Sample_Ptr;
     if (Temp_Sample_Ptr == Temp_Sample_Buffer_Start_Addr)
         Temp_Sample_Ptr += BEGSAMPLES;
 }

 for (n=BEGFRAMES-1, f=P_Start ; n >= 0 ; n--, f--) {
     ZC_Buffer[n] = Temp_ZC_Buffer[f];
     E_Buffer [n] = Temp_E_Buffer [f];
     if (f==0)
         f = BEGFRAMES;
 }

 Start_Frame = BEGFRAMES;

 f = Start_Frame-1;
 while ((E_Buffer[f] >= E_LOWER)  &  (f >= 0))
    Start_Frame = f--;
 while ((ZC_Buffer[f] >= ZCTHRESH)  &  (f >= 0))
    Start_Frame = f--;
```

```
Length = End_Frame - Start_Frame + 1;
Num_Frames = Length - OVERLAP + 1;
True_Start_Address = Sample_Buffer_Start_Addr1 + Start_Frame*SFRAMESIZE;

textcolor (light_grey);
gotoxy (58, 9); cprintf ("  Start = %d  ", Start_Frame);
gotoxy (58,10); cprintf ("  End   = %d  ", End_Frame);
textcolor (white);


/*-------------------------------------------------------------*/
/*  If in training mode, ask the user if he is happy with the  */
/*  way he said the word. If not then grab it again.           */
/*-------------------------------------------------------------*/

 if (Mode == 0) {
    Was_Input_OK_Message();
    c = getch();
    Clear_Was_Input_OK_Message();
    if (c == 'R' || c == 'r')
       goto Repeat;
 }

 Processing_Input_Message ();
 for (n=0 ; n < Length ; n++) {
    ZC_Buffer[n] = ZC_Buffer[n+P_Start];
    E_Buffer [n] = E_Buffer [n+P_Start];
 }


 if (Mode <= 3)
 for (p=0 ; p < Num_Frames ; p++) {

   /*-----------------------------------------------*/
   /*  Create the next frame of FRAMESIZE samples.  */
   /*  These are also converted to float variables. */
   /*-----------------------------------------------*/

    Sample_Ptr = True_Start_Address + p*SFRAMESIZE;
    for (n=0 ; n < FRAMESIZE ; n++)
        WSamps[n] = *Sample_Ptr++;

   /*-------------------------------------------------------------*/
   /*  Preemphasize this frame using  y(n) = x(n) - 0.95 * x(n-1) */
   /*-------------------------------------------------------------*/

    if (Preemphasis_Flag == 1) {
       for (n = FRAMESIZE-1 ; n > 0 ; n--)
           WSamps[n] = WSamps[n] - 0.95 * WSamps[n-1];
       WSamps[0] = 0.1 * WSamps[0];
    }
```

```
/*-------------------------------------------------------*/
/*  Place a Hamming window over this frame of samples.   */
/*  This is done using a look-up table.                  */
/*-------------------------------------------------------*/

 if (Hamming_Flag == 1) {
    for (n=0 ; n < FRAMESIZE ; n++)
        WSamps[n] *= Hamming_Lookup[n];
 }

/*-------------------------------------------------*/
/*  Calculate the Autocorrelation coefficients.  */
/*-------------------------------------------------*/

 for (k=0 ; k <= ORDER ; k++) {
    Auto_Corr[p][k] = 0;
    for (n=0 ; n < (FRAMESIZE-k) ; n++)
        Auto_Corr[p][k] += WSamps[n] * WSamps[n+k];
 }

/*-----------------------------------*/
/*  Calculate the LPC coefficients.  */
/*-----------------------------------*/

 if (Mode == 0 || Mode == 1 || Mode == 2) {

    Refl_Coeff[0] = -Auto_Corr[p][1] / Auto_Corr[p][0];
    LPC_Coeff[p][0] = 1;
    LPC_Coeff[p][1] = Refl_Coeff[0];
    Residual = Auto_Corr[p][0] + Auto_Corr[p][1] * Refl_Coeff[0];

    for (n=1 ; n < ORDER ; n++) {
       Sum = 0;
       for (k=0 ; k <= n ; k++)
          Sum += Auto_Corr[p][n-k+1] * LPC_Coeff[p][k];
       Refl_Coeff[n] = -Sum/Residual;
       M = (n+1)/2;
       for (k=1 ; k <= M ; k++) {
          Temp = LPC_Coeff[p][k] + LPC_Coeff[p][n-k+1] * Refl_Coeff[n];
          LPC_Coeff[p][n-k+1] += LPC_Coeff[p][k] * Refl_Coeff[n];
          LPC_Coeff[p][k] = Temp;
       }
       LPC_Coeff[p][n+1] = Refl_Coeff[n];
       Residual += Refl_Coeff[n] * Sum;
       if (Residual <= 0)
          err++;
    }

    if (err > 0) {
       textcolor (red);
       gotoxy (10,16);
       cprintf("WARNING: %d singular matrices in LPC calculation.",err)
       textcolor (white);
       getch();
    }
 }
```

```
  /*----------------------------------------*/
  /*  Calculate the Cepstral coefficients.  */
  /*----------------------------------------*/

   if (Mode == 2) {
      Cepstral[p][1] = LPC_Coeff[p][1];
      for (n=2 ; n <= ORDER ; n++) {
          Cepstral[p][n] = LPC_Coeff[p][n] * n;
          for (k=1 ; k < n ; k++)
              Cepstral[p][n] -= LPC_Coeff[p][k] * Cepstral[p][n-k];
      }
      for (n=2 ; n <= ORDER ; n++)
          Cepstral[p][n] = Cepstral[p][n] / n;
   }

}

Clear_Little_Message();
```

```
/*************************************************/
/*   Procedure to train the recognition system.  */
/*************************************************/

void Train_System (void)

{
   int    c, n, p, w;
   int    line;

   Training_Message();
   Setup_Main_Screen();
   Get_Number_Words:
   gotoxy (10, 5);  cprintf ("How many words to be entered ?  ");
   textcolor (yellow);
   cscanf ("%d", &WordTotal);  getch();
   textcolor (white);

   if (WordTotal > MAXWORDS) {
      WordTotal = MAXWORDS;
      Clear_Main_Screen();
      gotoxy (10, 3);
      cprintf ("Easy now!!  The maximum number of words is %d", WordTotal);
      goto Get_Number_Words;
   }

   Allocate_Library_Memory();


  /*-------------------------------------------------------------*/
  /*  Ask whether or not the speech must be pre-emphasized and if a  */
  /*  Hamming window must be placed over each frame.             */
  /*-------------------------------------------------------------*/

   Clear_Main_Screen();
   GetInput1:
   gotoxy (5, 3);
   cprintf(" Do you want to place a Hamming Window over each frame (Y/N) ?")
   if ((c=getch())=='Y' || c=='y')
      Hamming_Flag = 1;
   else if (c=='N' || c=='n')
      Hamming_Flag = 0;
   else
      goto GetInput1;

   Clear_Main_Screen();
   GetInput2:
   gotoxy (5, 3);
   cprintf (" Do you want to pre-emphasize each frame (Y/N) ?");
   if ((c=getch())=='Y' || c=='y')
      Preemphasis_Flag = 1;
   else if (c=='N' || c=='n')
      Preemphasis_Flag = 0;
   else
      goto GetInput2;
```

```
/*-------------------------------------------------------------------------*/
/*  Sample the requested number of words and store their parameters.  */
/*-------------------------------------------------------------------------*/

 for (w=0 ; w < WordTotal ; w++) {

    line = w+2-(w/15)*15;
    if (line == 2)
       Clear_Main_Screen();
    gotoxy (5, line);
    cprintf ("Print word to be entered:  ");
    textcolor (yellow);
    cscanf ("%s", &WordNames[w][0]); getch();
    textcolor (white);
    Delay (32000);                              /* Delay to stop keyboard */
    Delay (32000);                              /* click being picked up  */
    Delay (32000);                              /* by the microphone.     */
    Delay (32000);                              /* Approx 250 millisecs   */

    Sample_Word (0);                            /* Get the word.          */
    NumFrames[w] = Num_Frames;                  /* Save the no. of frames */

    Coeff_Ptr = Map_EMS (EMS_Handle, w);        /* Store the LPC coeffs   */
    for (n=0 ; n < Num_Frames ; n++) {          /* in library.            */
       for (p=0 ; p <= ORDER ; p++)
          *Coeff_Ptr++ = LPC_Coeff[n][p];
    }
    ZC_Library_Ptr = ZC_Library_Addr[w];        /* Store zero-crossing    */
    for (n=0 ; n < Length ; n++)                /* counts in library.     */
       *ZC_Library_Ptr++ = ZC_Buffer[n];
    E_Library_Ptr = E_Library_Addr[w];          /* Store frame energy     */
    for (n=0 ; n < Length ; n++)                /* contents in library.   */
       *E_Library_Ptr++ = E_Buffer[n];

 }
```

```
/*****************************************************************/
/*   Procedure to add words to or modify words in the vocabulary.   */
/*****************************************************************/

void Modify_Vocabulary (void)


   int    c, n, p, w;
   int    WordNum;


   Modify_Another:
   Clear_Main_Screen();
   gotoxy (10,5); cprintf("Modify a word, Add a word or Nothing? (M/A/N) ");

   if ((c=getch())=='M' || c=='m') {
      Get_Word_Number:
      gotoxy (10,7); cprintf("What is the number of the word to modify?  ");
      textcolor (yellow);
      cscanf ("%d", &WordNum);  getch();
      textcolor (white);
      if (WordNum > WordTotal)
         goto Get_Word_Number;
      gotoxy (10, 9);
      cprintf ("Is this the word \"%s\" ? ", WordNames[WordNum-1]);
      if ((c=getch())!='Y' && c!='y') {
         Clear_Main_Screen();
         goto Get_Word_Number;
      }
      w = WordNum-1;
   }

   else if (c=='A' || c=='a') {
      if (WordTotal >= MAXWORDS) {
         gotoxy (10, 7); cprintf("Library FULL");
         getch();
         return;
      }
      else {
         w = WordTotal;
         WordTotal++;
      }
   }

   else if (c=='N' || c=='n')
      return;

   else
      goto Modify_Another;

   gotoxy (10, 12);
   cprintf ("Print word to be entered:  ");
   textcolor (yellow);
   cscanf ("%s", &WordNames[w][0]); getch();
   textcolor (white);
```

```
Delay (32000);                          /* Delay to stop keyboard */
Delay (32000);                          /* click being picked up  */
Delay (32000);                          /* by the microphone.     */
Delay (32000);                          /* Approx 250 millisecs   */

Sample_Word (0);                        /* Get the word.          */
NumFrames[w] = Num_Frames;              /* Save the no. of frames */

Coeff_Ptr = Map_EMS (EMS_Handle, w);    /* Store the LPC coeffs   */
for (n=0 ; n < Num_Frames ; n++) {      /* in library.            */
   for (p=0 ; p <= ORDER ; p++)
      *Coeff_Ptr++ = LPC_Coeff[n][p];
}

ZC_Library_Ptr = ZC_Library_Addr[w];    /* Store zero-crossing    */
for (n=0 ; n < Length ; n++)            /* counts in library.     */
   *ZC_Library_Ptr++ = ZC_Buffer[n];

E_Library_Ptr = E_Library_Addr[w];      /* Store frame energy     */
for (n=0 ; n < Length ; n++)            /* contents in library.   */
   *E_Library_Ptr++ = E_Buffer[n];

goto Modify_Another;
```

```
'*********************************************/
'*   Procedure to recognize one spoken word.  */
'*********************************************/

void Recognize_Word (void)

   int     n, m, p, w, Temp;
   int     Nmax, Mmax;
   int     S[MAXWORDS];
   double  Distance [MAXWORDS];
   double  LocalDist[3];
   double  TotalDist;
   int     Closed[MAXFRAMES-1];
   int     Open[3];

  /*----------------------------------*/
  /*  Get the word to be recognized.  */
  /*----------------------------------*/

   Sample_Word (Distance_Type);
   Comparing_Input_Message ();

   Nmax = Num_Frames;
   for (n=0 ; n < 10 ; n++)
      Distance[n] = 30000;

  /*-------------------------------------------------*/
  /*  Compare input word to every word in library.  */
  /*-------------------------------------------------*/

   for (w=0 ; w < WordTotal ; w++) {

      Mmax = NumFrames[w];

     /*-------------------------------------------------------------------------*/
     /* Dont compare input word to reference words of very different size */
     /*-------------------------------------------------------------------------*/

      if (Mmax < (Nmax*3)/5 || Mmax > (Nmax*7)/5) {
          Distance[w] = 1000;
          continue;
      }

     /*-------------------------------------------------------------------------*/
     /* Move the coefficients of one word from library (far heap and EMS) */
     /* into local memory.                                                      */
     /*-------------------------------------------------------------------------*/

      if (Distance_Type <= 3) {
         Coeff_Ptr = Map_EMS (EMS_Handle, w);
         for (n=0 ; n < NumFrames[w] ; n++) {
             for (p=0 ; p <= ORDER ; p++)
                Ref_Coeff[n][p] = *Coeff_Ptr++;
         }
      }
```

```
 else if (Distance_Type == 4) {
    ZC_Library_Ptr = ZC_Library_Addr[w];
    for (n=0 ; n < MAXFRAMES ; n++)
       Ref_ZC_Buffer[n] = *ZC_Library_Ptr++;
    E_Library_Ptr = E_Library_Addr[w];
    for (n=0 ; n < MAXFRAMES ; n++)
       Ref_E_Buffer[n] = *E_Library_Ptr++;
 }

 if (Distance_Type == 1) {
    Log_Power_Ptr = Log_Power_Addr[w];
    for (n=0 ; n < NumFrames[w] ; n++)
        Log_Power[n] = *Log_Power_Ptr++;
 }

/*-------------------------------------------------*/
/*  If DTW_Flag = 0 then do linear time alignment.  */
/*-------------------------------------------------*/

 if (DTW_Flag == 0) {
    TotalDist = 0;
    for (n=0 ; n < Nmax ; n++) {
        m = (n-1) * (Mmax-1)/(Nmax-1) + 1;
        TotalDist += Frame_Distance (n, m);
    }
 }

/*-----------------------------------------------------------------*/
/*  If DTW_Flag = 1 then use OGS Dynamic Time Warping algorithm.  */
/*-----------------------------------------------------------------*/

 else {

    Closed[0] = 0;
    TotalDist = Frame_Distance (0, 0);

    LocalDist[0] = Frame_Distance (1, 0);
    LocalDist[1] = Frame_Distance (1, 1);

    if (LocalDist[0] < LocalDist[1]) {
       Closed[1] = 0;
       TotalDist += LocalDist[0];
    }
    else {
       Closed[1] = 1;
       TotalDist += LocalDist[1];
    }
```

```
for (n=2 ; n < Nmax ; n++) {

   Open[0] = Closed[n-1];
   Open[1] = Closed[n-1]+1;
   Open[2] = Closed[n-1]+2;

   if  ( Closed[n-1] != Closed[n-2]
      && Open[0] <= 2*n
      && Open[0] <= Mmax - (Nmax-n+1)/2
      && Open[0] <  Mmax
      && Open[0] >= n/2
      && Open[0] >  Mmax - 2*(Nmax-n)
      && Open[0] >= 0 )
      LocalDist[0] = Frame_Distance (n, Open[0]);
   else
      LocalDist[0] = 100;

   if  ( Open[1] <= 2*n
      && Open[1] <= Mmax - (Nmax-n+1)/2
      && Open[1] <  Mmax
      && Open[1] >= n/2
      && Open[1] >  Mmax - 2*(Nmax-n)
      && Open[1] >= 0 )
      LocalDist[1] = Frame_Distance (n, Open[1]);
   else
      LocalDist[1] = 100;

   if  ( Open[2] <= 2*n
      && Open[2] <= Mmax - (Nmax-n+1)/2
      && Open[2] <  Mmax
      && Open[2] >= n/2
      && Open[2] >  Mmax - 2*(Nmax-n)
      && Open[2] >= 0 )
      LocalDist[2] = Frame_Distance (n, Open[2]);
   else
      LocalDist[2] = 100;

   if (LocalDist[1] <= LocalDist[0]) {
      if (LocalDist[2] < LocalDist[1]) {
         TotalDist += LocalDist[2];
         Closed[n] = Open[2];
      }
      else {
         TotalDist += LocalDist[1];
         Closed[n] = Open[1];
      }
   }
   else if (LocalDist[2] < LocalDist[0]) {
      Closed[n] = Open[2];
      TotalDist += LocalDist[2];
   }
   else {
      Closed[n] = Open[0];
      TotalDist += LocalDist[0];
   }
}
```

```
    }
    Distance[w] = TotalDist;
 }

/*-----------------------------------------*/
/*  Sort the words and print the best five.  */
/*-----------------------------------------*/

 for (n=0 ; n < WordTotal ; n++)
     S[n] = n;

 for (n=0 ; n < WordTotal ; n++) {
    for (m=n+1 ; m < WordTotal ; m++) {
        if (Distance[S[n]] > Distance[S[m]]) {
           Temp = S[m];
           S[m] = S[n];
           S[n] = Temp;
        }
    }
 }

 Clear_Main_Screen();
 for (w=0 ; w < WordTotal  &&  w < 5 ; w++) {
     textcolor (light_grey);
     gotoxy  (27, w+12);
     cprintf ("%8.2f", Distance[S[w]]);
     textcolor (yellow);
     gotoxy  (9, w+12);
     cprintf ("%2d.", w+1);
     textcolor (white);
     gotoxy  (14, w+12);
     cprintf ("%s", WordNames[S[w]]);
 }

 window         (10, 9, 38, 13);
 textbackground (black);
 clrscr();
 gotoxy (2, 1);  cprintf ("╔═══ RECOGNIZED WORD ═══╗");
 gotoxy (2, 2);  cprintf ("║                       ║");
 gotoxy (2, 3);  cprintf ("║                       ║");
 gotoxy (2, 4);  cprintf ("║                       ║");
 gotoxy (2, 5);  cprintf ("╚═══════════════════════╝");
 textcolor (yellow);
 gotoxy (8, 3);  cprintf ("%s", WordNames[S[0]]);
 return;
```

```
**************************************************************************/
  Procedure to calculate the distance between one frame of the unknown  */
  spoken word and one frame of a reference word from the library.       */
  The distance measure used is user-selected.                           */
**************************************************************************/

uble  Frame_Distance (int UnknownFrame, int RefFrame)


 int     n, m, p;
 double  ZC_Dist;
 double  Energy_Dist;
 double  Residual1, Residual2;
 double  Dist, FrameDistance;


 n = UnknownFrame;
 m = RefFrame;

/*-------------------------------------------------*/
/*                                      Ar Rt Ar   */
/*  Ikatura's distance measure:   d = ----------   */
/*                                      At Rt At   */
/*-------------------------------------------------*/

 if (Distance_Type == 1) {

    Residual1 = 0;
    Residual2 = 0;
    for (p=0 ; p <= ORDER ; p++) {
        Residual1 += Ref_Coeff[m][p] * Auto_Corr[n][p];
        Residual2 += LPC_Coeff[n][p] * Auto_Corr[n][p];
    }

    if (Residual2 == 0)                 /* These two conditions can only */
        Residual2 = 0.000001;           /* occur if garbage is detected  */
    Dist = Residual1/Residual2;         /* as a word.                    */
    if (Dist <= 0)
        Dist = 0.000001 - Dist;

    return ( Log_Power[m] + log (Dist) );

 }
```

```
/*-------------------------------------------------------*/
/*                                                 2     */
/*    Cepstral Coefficients:   d = sum [ct(i) - cr(i)]   */
/*                                                       */
/*-------------------------------------------------------*/

 else if (Distance_Type == 2) {

    FrameDistance = 0;
    for (p=1 ; p <= ORDER ; p++) {
       Dist = Cepstral[n][p] - Ref_Coeff[m][p];
       FrameDistance += Dist * Dist;
    }
    return (FrameDistance);

 }

/*-------------------------------------------------------*/
/*                                                       */
/*    Distance measure:   d = log (AC(test), LPC(ref))   */
/*                                                       */
/*-------------------------------------------------------*/

 else if (Distance_Type == 3) {

    FrameDistance = 0;
    for (p=0 ; p <= ORDER ; p++)
       FrameDistance += Ref_Coeff[m][p] * Auto_Corr[n][p];
    if (FrameDistance < 0)
       FrameDistance = -FrameDistance;
    return (log(FrameDistance));

 }

/*---------------------------------------*/
/*                                       */
/*    Zero-Crossing / Energy difference  */
/*                                       */
/*---------------------------------------*/

 else if (Distance_Type == 4) {

    if (E_Buffer[n] > Ref_E_Buffer[m])
       Energy_Dist = E_Buffer[n] / Ref_E_Buffer[m];
    else
       Energy_Dist = Ref_E_Buffer[m] / E_Buffer[n];
    ZC_Dist = abs (Ref_ZC_Buffer[m] - ZC_Buffer[n]);
    return (ZC_Dist + 0.5*Energy_Dist);

 }

 return (NULL);
```

```
*************************************************************************/
  Procedure to allocate memory space in the far heap for the library of */
  zero-crossing counts and log power constants. Expanded memory (EMS)   */
  is allocated for the LPC coefficients. There are 64 logical pages of  */
  EMS, each 16k in size and the coefficients of two words are stored in */
  each page. The EMS pages are allocated here and may be accessed after */
  mapping them to a physical page (by calling the Map_EMS() procedure). */
  If Cepstral coeffs are used then the LPC coeffs are still stored here */
  (and later on disk) but they are converted to Cepstral coeffs just    */
  before recognition begins. Similarly, if Ikatura's measure is being   */
  used, the LPC coefficients in the library are modified.               */
*************************************************************************/

oid    Allocate_Library_Memory (void)


 int     w;


 EMS_Handle = Alloc_EMS (64);
 if (EMS_Handle == NULL) {
    Clear_Main_Screen();
    gotoxy (10, 8);
    cprintf("Error allocating EMS.");
    getch(); return;
 }

 for (w=0 ; w < MAXWORDS ; w++) {
    ZC_Library_Addr[w] = farcalloc (MAXFRAMES, sizeof(int));
    if (ZC_Library_Addr[w] == NULL) {
      Clear_Main_Screen();
      gotoxy (10, 8);
      cprintf("Not enough memory in far heap,%ul bytes free",farcoreleft()
      getch(); return;
    }
 }

 for (w=0 ; w < MAXWORDS ; w++) {
    E_Library_Addr[w] = farcalloc (MAXFRAMES, sizeof(int));
    if (E_Library_Addr[w] == NULL) {
      gotoxy (10, 8);
      cprintf("Not enough memory in far heap,%ul bytes free",farcoreleft()
      getch(); return;
    }
 }

 for (w=0 ; w < MAXWORDS ; w++) {
    Log_Power_Addr[w] = farcalloc (MAXFRAMES, sizeof(double));
    if (Log_Power_Addr[w] == NULL) {
      gotoxy (10, 8);
      cprintf("Not enough memory in far heap,%ul bytes free",farcoreleft()
      getch(); return;
    }
 }
```

```
*************************************************************************/
  Procedure to save a library of coefficients of the reference words   */
*************************************************************************/

id Save_Vocabulary (void)


 FILE     *File;
 int       c, n, w, Handle;
 int       ItemsWritten;
 char      FileName[12];
 double   *LPC_Buffer_Ptr;
 int      *ZC_Buffer_Ptr;
 unsigned long *E_Buffer_Ptr;

 Clear_Main_Screen();
 GetFileName:
 gotoxy (6,4); cprintf ("Enter file name to store vocabulary: ");
 cscanf ("%s", &FileName); getch();

 if (access(FileName, 0))
    _creat (FileName, 0);
 else {
    textcolor(yellow);
    gotoxy (6,6); cprintf ("File already exists.");
    gotoxy (6,7); cprintf ("Do you want to overwrite it? (Y/N) ");
    textcolor(white);
    if ((c=getch()) != 'Y' && c != 'y') {
       Clear_Main_Screen();
       goto GetFileName;
    }
 }

 Handle = _open  (FileName, O_RDWR);
 File   = fdopen (Handle, "w");
 if (File == NULL) {
    textcolor(yellow);
    gotoxy (6,4); cprintf ("Please try again.");
    textcolor(white);
    goto GetFileName;
 }

 Working_Message (light_grey);

 ItemsWritten  = fwrite(&WordTotal,       sizeof(WordTotal),       1,File)
 ItemsWritten += fwrite(&DTW_Flag,        sizeof(DTW_Flag),        1,File)
 ItemsWritten += fwrite(&Hamming_Flag,    sizeof(Hamming_Flag),    1,File)
 ItemsWritten += fwrite(&Preemphasis_Flag,sizeof(Preemphasis_Flag),1,File)

 ItemsWritten += fwrite (&WordNames, sizeof(WordNames[0]), WordTotal,File)
 ItemsWritten += fwrite (&NumFrames, sizeof(NumFrames[0]), WordTotal,File)

 if (ItemsWritten < WordTotal*2 + 4) {
    gotoxy (6,8);
    cprintf ("Error writing %s", FileName);
 }
```

```
for (w=0 ; w < WordTotal ; w++) {

   Coeff_Ptr = Map_EMS (EMS_Handle, w);          /* Transfer data to the */
   LPC_Buffer_Ptr = &LPC_Coeff[0][0];            /* current data segment */
   for (n=0 ; n < BUFFERSIZE ; n++)              /* before writing to    */
      *LPC_Buffer_Ptr++ = *Coeff_Ptr++;          /* the file.            */

   ItemsWritten = fwrite (&LPC_Coeff[0][0], sizeof(LPC_Coeff[0][0]),
                                                    BUFFERSIZE, File);
   if (ItemsWritten < BUFFERSIZE) {
      gotoxy (6,9);
      cprintf ("Error writing %s", FileName);
   }

}

for (w=0 ; w < WordTotal ; w++) {

   ZC_Library_Ptr = ZC_Library_Addr[w];          /* Transfer data to the */
   ZC_Buffer_Ptr  = &ZC_Buffer[0];               /* current data segment */
   for (n=0 ; n < MAXFRAMES ; n++)               /* before writing to    */
      *ZC_Buffer_Ptr++ = *ZC_Library_Ptr++;      /* the file.            */

   ItemsWritten=fwrite(&ZC_Buffer[0],sizeof(ZC_Buffer[0]),MAXFRAMES,File)
   if (ItemsWritten < MAXFRAMES) {
      gotoxy (6,10);
      cprintf ("Error writing %s", FileName);
   }

}

for (w=0 ; w < WordTotal ; w++) {

   E_Library_Ptr = E_Library_Addr[w];            /* Transfer data to the */
   E_Buffer_Ptr  = &E_Buffer[0];                 /* current data segment */
   for (n=0 ; n < MAXFRAMES ; n++)               /* before writing to    */
      *E_Buffer_Ptr++ = *E_Library_Ptr++;        /* the file.            */

   ItemsWritten=fwrite(&E_Buffer[0],sizeof(E_Buffer[0]),MAXFRAMES,File);
   if (ItemsWritten < MAXFRAMES) {
      gotoxy (6,10);
      cprintf ("Error writing %s", FileName);
   }

}

fclose (File);
```

```
**************************************************************/
  Procedure to load a previously saved library of coefficients  */
**************************************************************/

id Load_Vocabulary (void)


 FILE     *File;
 int       n, w, Handle;
 int       ItemsRead;
 char      FileName[12];
 double   *LPC_Buffer_Ptr;
 int      *ZC_Buffer_Ptr;
 unsigned long *E_Buffer_Ptr;


 Clear_Main_Screen();
 GetFileName:
 gotoxy (44,4);
 cprintf ("               ");
 gotoxy (6,4);
 cprintf ("Enter file name of vocabulary to load: ");
 cscanf ("%s", &FileName); getch();

 if (access(FileName, 0)) {
    gotoxy (6,2);
    textcolor (yellow+blink);
    cprintf ("File not found");
    textcolor (white);
    goto GetFileName;
 }

 Working_Message (light_grey);
 Handle = _open  (FileName, O_RDWR);
 File   = fdopen (Handle, "w");

 ItemsRead  = fread (&WordTotal,        sizeof(WordTotal),        1, File)
 ItemsRead += fread (&DTW_Flag,         sizeof(DTW_Flag),         1, File)
 ItemsRead += fread (&Hamming_Flag,     sizeof(Hamming_Flag),     1, File)
 ItemsRead += fread (&Preemphasis_Flag, sizeof(Preemphasis_Flag), 1, File)

 ItemsRead += fread (&WordNames, sizeof(WordNames[0]), WordTotal, File);
 ItemsRead += fread (&NumFrames, sizeof(NumFrames[0]), WordTotal, File);

 if (ItemsRead < WordTotal*2 + 4) {
    gotoxy (6,7);
    cprintf ("Error reading %s", FileName);
 }

 Allocate_Library_Memory();
```

```
for (w=0 ; w < WordTotal ; w++) {

   ItemsRead = fread (&LPC_Coeff[0][0], sizeof(LPC_Coeff[0][0]),
                                                BUFFERSIZE, File);
   if (ItemsRead < BUFFERSIZE) {
      gotoxy (6,10);
      cprintf ("Error reading %s", FileName);
   }

   Coeff_Ptr = Map_EMS (EMS_Handle, w);          /* Transfer data from */
   LPC_Buffer_Ptr = &LPC_Coeff[0][0];            /* local data segment */
   for (n=0 ; n < BUFFERSIZE ; n++)              /* to expanded memory */
      *Coeff_Ptr++ = *LPC_Buffer_Ptr++;

}

for (w=0 ; w < WordTotal ; w++) {

   ItemsRead = fread(&ZC_Buffer[0], sizeof(ZC_Buffer[0]), MAXFRAMES, File
   if (ItemsRead < MAXFRAMES) {
      gotoxy (6,10);
      cprintf ("Error reading %s", FileName);
   }

   ZC_Library_Ptr = ZC_Library_Addr[w];          /* Transfer data from */
   ZC_Buffer_Ptr  = &ZC_Buffer[0];               /* local data segment */
   for (n=0 ; n < MAXFRAMES ; n++)               /* to far heap.       */
      *ZC_Library_Ptr++ = *ZC_Buffer_Ptr++;

}

for (w=0 ; w < WordTotal ; w++) {

   ItemsRead = fread(&E_Buffer[0], sizeof(E_Buffer[0]), MAXFRAMES, File);
   if (ItemsRead < MAXFRAMES) {
      gotoxy (6,10);
      cprintf ("Error reading %s", FileName);
   }

   E_Library_Ptr = E_Library_Addr[w];            /* Transfer data from */
   E_Buffer_Ptr  = &E_Buffer[0];                 /* local data segment */
   for (n=0 ; n < MAXFRAMES ; n++)               /* to far heap.       */
      *E_Library_Ptr++ = *E_Buffer_Ptr++;

}

fclose (File);
```

```
*************************************************************/
  Procedure to give a delay of approximately 2 microseconds  */
*************************************************************/

id  Delay (int Del)

 int  n, d;

 for (n=0 ; n < Del ; n++) {
     d = 10;
     d = d*2;
 }



*************************************************************************/
*   Procedure to allocate a specified number of pages (NumPages) of     */
*   expanded memory and return an EMM handle for the pages allocated.   */
*   Each page is 16k in size. If the allocation is unsuccessful, then   */
*   NULL is returned.                                                   */
*************************************************************************/

nt Alloc_EMS (int NumPages)

 union  REGS reg;

 reg.h.ah = 67;
 reg.x.bx = NumPages;
 int86 (0x67, &reg, &reg);
 if (reg.h.ah == 0)
   return (reg.x.dx);
 else
   return (NULL);



*************************************************************/
*  Procedure to release all logical pages of expanded memory   */
*  associated with an active EMM handle.                       */
*************************************************************/

oid Free_EMS (int Handle)

 union  REGS reg;

 reg.h.ah = 69;
 reg.x.dx = Handle;
 int86 (0x67, &reg, &reg);
```

```
/*************************************************************************
  Procedure to map a logical EMS page (0-63) to page-0 of the EMM page  */
  frame. There are two words per 16k page; an even word starts at the   */
  beginning of a page and an odd word starts half-way. The start add-   */
  ress within page-0 is returned unless the mapping was unsuccessful in */
  which case NULL is returned.                                          */
*************************************************************************/

ouble far *Map_EMS (int Handle, int WordNum)

  union  REGS reg;

  reg.x.ax = 0;                                 /* Page-0 of EMM page frame  */
  reg.x.bx = WordNum/2;                         /* Logical EMS page number   */
  reg.x.dx = Handle;
  reg.h.ah = 68;
  int86 (0x67, &reg, &reg);
  if (reg.h.ah != 0)
     return (NULL);

  else if (WordNum%2 == 0)
     return (0xC8000000L);                      /* Start address of Page-0   */
  else                                          /* ie. C800:0000 (C8000h)    */
     return (0xCA000000L);                      /* Page-0 start address + 8k */
```

# References


[1] A S Poulton, "Microcomputer Speech Synthesis and Recognition", Sigma Technical Press, 1983

[2] K H Davis, R Biddulph and S Balashek, "Automatic Recognition of Spoken Digits", Journal of the Acoustic Society of America, Vol. 24, pp 637, 1952

[3] P Denes and M V Mathews, "Spoken Digit Recognition using Time-frequency Pattern-matching", Journal of the Acoustic Society of America, Vol. 32, pp 1450-1455, 1960

[4] Product Focus, "Voice Input/Output Systems and Devices", Electronic Engineering, pp 76-93, May 1982

[5] M K Brown and L R Rabiner, "An Adaptive, Ordered, Graph Search Technique for Dynamic Time Warping for Isolated Word Recognition", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-30, No. 4, pp 535-544, August 1982

[6] K C Pan, F K Soong and L R Rabiner, "A Vector-Quantization-Based Preprocessor for Speaker Independent Isolated Word Recognition.", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-33, No. 3, pp 546-560, June 1985

[7] M Elphick, "Speech Recognition", 'Speech Synthesis' edited by G Bristow, Granada Publishing Ltd., 1984

[8] J N Holmes, "Speech Synthesis and Recognition", Von Nostrand Reinold (UK), 1985

[9] G E Peterson and H L Barney, "Control methods used in a study of the vowels", Journal of the Acoustic Society of America, Vol. 24, No. 2, pp 175-184, March 1952

[10] H K Dunn, "Methods of Measuring Vowel Formant Bandwidths", Journal of the Acoustic Society of America, Vol. 33, pp 1737-1746, 1961

[11] L R Rabiner and R W Schafer, "Digital Processing of Speech Signals", Prentice/Hall, 1978

[12] B S Atal and S L Hanauer, "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave", Journal of the Acoustic Society of America, Vol. 50, No. 2 (Part 2), pp 637-655, August 1971

[13] F Fallside, "Frequency-domain Analysis of Speech", 'Computer Speech Processing' edited by F Fallside and W A Woods, Prentice/Hall, 1983

[14] L R Rabiner, "On the Use of Autocorrelation Analysis for Pitch Detection", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-25, No. 1, pp 24-33, February 1977

[15] J J Dubnowski, R W Schafer and L R Rabiner, "Real-Time Digital Hardware Pitch Detector", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-24, No. 1, pp 2-8, February 1976

[16] L R Rabiner, M J Cheng, A R Rosenberg and C A McGonegal, "A Comparative Performance Study of Several Pitch Detection Algorithms", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-24, No. 5, pp 399-418, October 1976

[17] I H Witten, "Principles of Computer Speech", Academic Press Inc., 1982

[18] L R Rabiner, S E Levinson, A E Rosenberg & J G Wilpon, "Speaker-Independent Recognition of Isolated Words Using Clustering Techniques", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-27, No. 4, pp 336-349, August 1979

[19] B S Atal, "Linear Predictive Coding of Speech", 'Computer Speech Processing' edited by F Fallside and W A Woods, Prentice/Hall, 1983

[20] J Makhoul, "Spectral Linear Prediction: Properties and Applications", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-23, No. 3, pp 283-296, June 1975

[21] J Makhoul, "Linear Prediction: A Tutorial View", Proceedings of the IEEE, Vol. 63, pp 561-580, April 1975

[22] F I Itakura, "Minimum Prediction Residual Principle Applied to Speech Recognition", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-23, No. 1, pp 67-72, Feb 1975

[23] Y Tohkura, "A weighted Cepstral Distance Measure for Speech Recognition", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-35, No. 10, October 1987

[25] L F Lamel, L R Rabiner, A E Rosenberg and J G Wilpon, "An Improved Endpoint Detector for Isolated Word Recognition", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-29, No. 4, pp 777-785, August 1981

[26] L R Rabiner and M R Sambur, "An Algorithm for Determining the Endpoints of Isolated Utterances", Bell System Technical Journal, Vol. 54, No. 2, pp 297-315, February 1975

[27] J M Tribolet and L R Rabiner, "Statistical Properties of an LPC Distance Measure", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-27, No. 5, pp 550-558, October 1979

[28] J G Wilpon, L R Rabiner and A Bergh, "Speaker-Independent Isolated Word Recognition using 129-word Airline Vocabulary", Journal of the Acoustic Society of America, Vol. 72, pp 390-396, August 1982

[29] L R Rabiner and J G Wilpon, "Speaker-Independent Isolated Word Recognition for a Moderate Size Vocabulary", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-27, No. 6, pp 583-587, December 1979

[30] E R Teja and G W Gonella, "Voice Technology", Reston Publishing Co., 1983

[31] L R Rabiner, M M Sondhi and S E Levinson, "Note on the Properties of a Vector Quantizer for LPC Coefficients", Bell System Technical Journal, Vol. 62, No. 8, pp 2603-2616, October 1983

[32] L R Rabiner, S E Levinson and M M Sondhi, "On the Application of Vector Quantization and Hidden Markov Models to Speaker Independent, Isolated Word Recognition", Bell System Technical Journal, Vol. 62, No. 4, pp 1075-1105, April 1983