Investigation into Zero-Crossing Techniques as a Viable Means of Speech Recognition.

Presented to the
School of Electronic Engineering,
Dublin City University

By
Keith J. Newsome BSc(Eng)

On the Date of
February 1996

For the Award of
MEng

Under the Supervision of
Dr. Eugene Coyle (Dublin Institute of Technology)
and
Dr. Ronan Scaife (Dublin City University)

Completed using Information Entirely Based on the Candidate's Own Research.

Total Number of Volumes Including this One: 1

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M.Eng is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my own work.

Signed: Keith Newsome **Date:** 1996-10-12

Candidate

I would like to dedicate this thesis to the memories of both my Grandmothers, Mrs. Mary Newsome and Mrs. Josephine Cantwell, who sadly passed away during its completion.

Acknowledgements

I wish to extend my kindest regards to both my supervisors, Dr. Eugene Coyle (DIT) and Dr. Ronan Scaife (DCU), for their generous help and advice throughout the duration of this project.

In addition, I would like to thank Mr. Willie Brien, Engineering Manager, Ogden Atlantic, for organising most generous funds in the sponsorship of this project.

Finally, I wish to thank the following people:

- Mr. Robert Lawlor for his Matlab and DSP advice.
- Mr. Damon Berry, Mr. Ciarán Young and all associated with the IC^2 for their charitable assistance.
- Mr. Tony Davitt and Mr. Steven Nuzum for their kind help in certain aspects of speech recognition.
- Mr. Carsten Clausnitzer, Mr. Maic Kröll and Miss Steffi Bobek for their shrewd hardware and software advice.
- Miss Muriel Devane for her support throughout the project.
- My family for their support and endurance throughout the duration of this project and the rest of my educating years.

Publications.

'Comparison of Zero Crossing Rate and Higher Order Crossing Techniques for Improving Discrimination In Speech Signals', Irish Digital Signal Processing & Control Colloquim (IDSPCC '95), Queen's University Belfast, June 1995.

List of Figures.

CHAPTER 4

Contents

Abstract

The idea behind this research is to demonstrate how a fundamental characteristic of speech (zero-crossing information) may be exploited in the development of a low cost, highly effective speech recognition system. The system is to be used to recognise a small vocabulary of isolated speech. Although intended to be speaker dependent, the system is also tested for speaker independence.

A brief description of how speech is produced and recognised by a human subject is first presented. Following this, some features of both voiced and unvoiced speech signals and their associated spectra are discussed in relation to zero-crossing information. Phonemes and their segmentation (using zero-crossing data or otherwise) are also examined. A brief discussion of stationarity and its effects on zero-crossings is then given. The choice of pre-processing filters is also mentioned.

Two methods of speech recognition implementing zero-crossing information are then discussed.

The first technique studied analyses the 'spacing' between zero-crossings, producing a signal whose amplitude is proportional to the distance between successive crossings. The possibility of this system, (termed *Sinusoidal Instantaneous Frequency Extractor (SIFE)* [14]), producing effective recognition parameters is examined.

A second analysis technique, called *Higher Order Crossing Analysis (HOC)* [25], is then introduced. This method extracts higher order zero-crossing information from the signal using various filtering techniques and uses this data to recognise the speech signal.

Modified versions of both methods were developed, tested and found to be more effective and adaptable than their predecessors.

A new parameter (*Columnised Higher Order Crossing (CHOC)*) was developed and found to be more effective than HOC. Dynamic Time Warping was then implemented to pattern match CHOC templates with CHOC test signals, enabling a percentage success rate for the CHOC system to be achieved (~90%).

Finally, a comparison of the two systems is then made and a discussion about their effectiveness is given.

Abbreviations

ADM	Accumulated Distance Matrix
β	Delay Operator
BPF	Band Pass Filter
Cat X	Category X
CHOC	Columnised Higher Order Crossing
C_V	Coefficient of Variation
∇	Differencing Operator
$\mathcal{D}_A$	Accumulated Distance
DFT	Discrete Fourier Transform
D_j	Number of Zero-Crossings after j Differencing Stages
D_{jav}	Average Number of Zero-Crossings in a phoneme after j Differencing Stages
$\bar{D}_{min}$	Average of Minimum Accumulated Distances
DSP	Digital Signal Processing
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
f_N	Nyquist Frequency
f_S	Sampling Frequency
Gr X	Group X Vocabulary Set
HOC	Higher Order Crossing
HPF	High Pass Filter
IPA	International Phonetic Alphabet
${}_jD$	Number of Zero-Crossings after j Summation Stages
KNN	K Nearest Neighbour
LDM	Local Distance Matrix
LPF	Low Pass Filter
.mat	Matlab File Format
MSE	Mean Square Energy
NN	Nearest Neighbour
$\check{S}$	Summation Operator
SIFE	Sinusoidal Instantaneous Frequency Extractor
SIFE'	Inverse SIFE Data (ie. 1/SIFE)
SIFESIM	SIFE Simulator
S/N	Signal to Noise Ratio
STFT	Short-Time Fourier Transform
TFR	Time-Frequency Representation
UV	Unvoiced Speech
V	Voiced Speech
Var	Variance
VUD	Voiced-Unvoiced Decision
.wav	Windows Sound File Format
ZC	Zero-Crossing
ZCR	Zero-Crossing Rate
ZOH	Zero Order Hold

Introduction

Current research in the field of speech recognition is advancing at a great rate with new heights in improved success rates being reached as a result of implementing the latest state of the art technologies. A particular challenge to researchers is to maintain a balance in gaining an understanding of and utilising mathematical processing techniques while moving in a direction which will positively contribute to the subject. It can often happen that more simplistic analysis techniques are in fact too useful to be discarded. Hence, this thesis was written to demonstrate how a fundamental characteristic of speech signals (i.e. zero-crossing data) may be exploited to recognise speech patterns.

The fact that it is simple and inexpensive to implement zero-crossing analysis in both hardware and software makes this an attractive analysis technique. However, the success rates achieved from zero-crossing techniques throughout the years have not been as attractive and so research in this field has become sparse. Nevertheless, research performed by Licklider and Niederjohn et al. [12] found high intelligibility (>90%) in the zero-crossing information of speech signals. For this reason, (as well as the cost and ease of implementation), zero-crossing analysis cannot yet be discarded as a speech recognition technique.

This thesis shows how suitable zero-crossing information may be used in the recognition of small vocabularies (less than 15 words) of isolated speech (i.e. not continuous speech or phrases). In particular, two zero-crossing techniques are studied, (*SIFE* and *HOC*), with the two resulting systems being tested and modified in an attempt to develop a reliable speech recognition system. A new system based on HOC theory (termed *CHOC*) is then developed and tested resulting in percentage success rate in the order of 90%.

The CHOC system was tested for noise tolerance, resistance to non-linear phase changes and intonation changes. It was found to be tolerant to non-linear phase changes, although a threshold signal to noise ratio of 10dB was established to ensure effective operation. The CHOC system was found to be relatively tolerant to intonation changes depending on the vocabulary size. The extension of this system to cater for continuous speech was examined and a comparison of CHOC data to that of a spectrogram was made.

Chapter 1

1. Theory of Speech Signals.

This chapter begins by discussing the human brain as a speech recognition system and questions whether or not there will ever be an electronic equivalent developed. A background to zero-crossing analysis in speech and other fields is then presented. This is followed by a brief description of how voiced and unvoiced speech is produced, how each affects the zero-crossing content of the signal and the resulting differences in the corresponding spectra. Then follows an introduction to the concept of phonemes, their characteristic properties and how a speech signal may be segmented into its individual phonemes. A section is presented describing the quasi-stationary nature of speech signals and how this may affect zero-crossing analysis. Finally, the necessary pre-processing of speech signals is mentioned.

1.1 The Human Brain - The Ultimate Speech Recognition System?

It would be quite an insult to the human species to discuss the field of speech recognition without referring to the most powerful of all recognition systems: the human brain. This marvellous 'microprocessor' can:

(i) Detect a wide range of sounds via the ear and localise the position of the source [49];

(ii) Calculate basic acoustic parameters such as frequency, timbre and intensity [49];

(iii) Understand speech signals and carry out responses to them;

(iv) Decide which sounds to listen to and which to ignore (*selection*) [72]

The brain performs all processing in real-time, allowing a constant up-date of information. It allows efficient transfer of data to other parts of the body and is totally compatible with all other systems in the body [26]. The brain is also reasonably tolerant to a range of environments.

Several theories have been proposed over the years as to how exactly the human brain, working in conjunction with the auditory system, recognises different sounds. Early physiology assumed the ear (both inner and outer) was solely responsible for sound perception and recognition. However, further studies in human anatomy discovered the vital role played by the brain. Research into how the ear actually produces the electrical signal and the structure of this signal resulted in what are widely known as the *Classical Theories* [71]: The Place Theory after Helmholtz, frequency theories, The Resonance Theory and The Volley Theory. Although some of this research dates from as far back as the 16th century, the theories are still respected today with auditory protagonists dividing into three classes [48]:

(i) Those who believe the Place Theory;

(ii) Those who believe Temporal Theory;

(iii) The eclectic group who believe both theories are significant.

The Place Theory suggests that shifts in the place of maximum excitation in the cochlea† occur for different sounds. The Temporal Theory claims information is stored in the form of time intervals between neural firings. Evidence of the validity of both theories is described by Pickles [48] who concludes that to favour one over the other would be unjustified. His research tends to suggest that the Temporal Theory holds for speech signals. However, Pickles also claims that favouring the eclectic view may be a function of the quality of evidence available rather than of the actual operation of the auditory system.

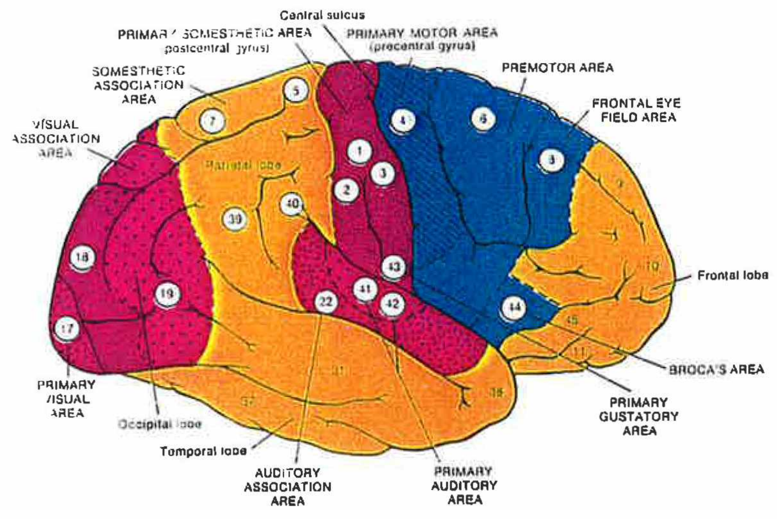


Figure 1.1: Functional areas of the brain.

Recent research [43] has proven that the ear, (up to now understood as being purely a receiver), also transmits a distinct audio signal for each detected sound.

† The section of the inner ear containing nerve fibres which vibrate transmitting signals to the brain via the auditory nerve.

Other pre-processing also occurs due to the outer ear and body and head resonance [49]. Perhaps this pre-processing should be considered when developing a speech recognition system as powerful as the human brain.

A detailed description of the areas in the human brain responsible for sound analysis is shown in Figure 1.1 [67]. The Primary Auditory Area (areas 41 and 42) is responsible for interpreting the fundamental characteristics of sound - pitch, rhythm, timbre, etc. and is located in the temporal lobe. Area 22 (the Auditory Association Area) is also located in the temporal lobe and determines if the sound detected is music, speech or noise, etc. Also known as Wernicke's Area, it is responsible for interpreting the meaning of speech by translating words and phrases into thoughts. These areas were discovered (as were many other areas) through accidental damage to the brain. Damage to the left side of the brain tended to result in a loss of comprehension of speech, but this rarely happened when the injury was inflicted to the right hand side. Pickles showed how humans have a greater chance of recognising speech when received through the right ear instead of the left (thus verifying the fact that the left half of the body is mapped to the right side of the brain and vice-versa). Taniguchi et al [62] and Price et al [50] describe the methods used to map the different parts of the brain resulting in a contour map similar to the one shown in Figure 1.2 [48] clearly showing the relationship between blood-flow in certain areas and hearing spoken words.

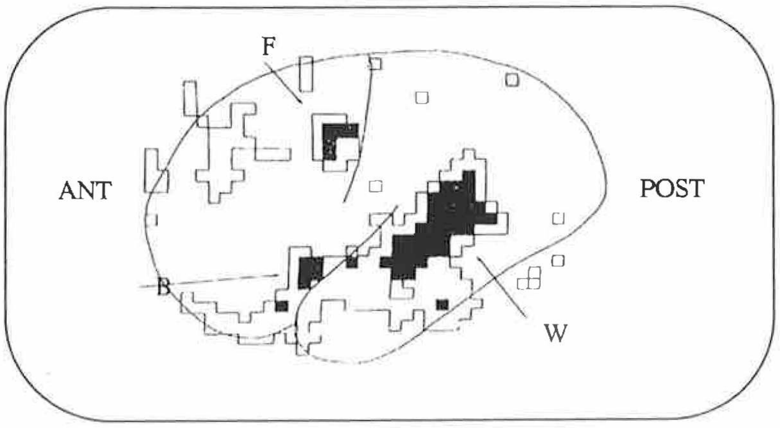


Figure 1.2: Mapping of the brain demonstrating the increase in blood flow while passively listening to speech [48]. Note: W=Wernicke's Area, B=Broca's Area and F=Frontal Eye Fields.

Petsche et al [47] show how EEG signals differs for when hearing speech and music and carrying out mental arithmetic. The EEG signals resulting from listening to music and solving simple mathematical problems proved to be significantly different to those resulting from interpreting speech. This strengthens the idea of a separate '*mode*' for dealing with speech signals.

Much research into the speech recognition function of the brain has yet to be carried out and as Pickles concludes, our understanding of how the brain actually processes speech is still in its infancy. However, it seems to make sense that in order to develop a speech recognition system as 'intelligent' as the brain, we must first understand how the most powerful system operates.

1.2 Background to Speech Recognition and Zero-Crossing Information.

Prior to the 1960's, most speech recognition systems consisted of an acoustic analyser, (producing some form of spectrum), followed by a pattern classifier [2]. These procedures proved uneconomic and generally resulted in low success rates. This may have contributed to the changes in approach during the 1960's. One particular technique which was studied in detail by many researchers is zero-crossing analysis, (probably due to its ease of implementation in hardware).

Over the past thirty years zero-crossing information has been used as a signal analysis technique for a broad range of applications. Gluskin (1991) [20] showed how zero-crossing data may be used to obtain a mathematical description of a system. Friedman (1994) [18] found it useful to estimate the frequency of a single sinusoid in white noise. Other implementations of zero-crossing data includes bar-code recognition [24], the recovery of missing speech packets [16], and signal reconstruction of unknown signals [68]. Meanwhile, the study into the use of zero-crossing data as a means of recognising speech signals has proven popular for research purposes [1,3,14,73].

The most basic method of achieving a zero-crossing rate (ZCR) plot is to slide a non-overlapping running rectangular window along the speech signal and to calculate the zero-crossing rate within that time frame. A plot of ZCR against time results. Various adaptive versions of this technique have been developed, one of which uses an overlapping window incorporating weighting factors, giving more weight to the most current data [25].

However, not all researchers have an optimistic view of zero-crossing techniques. Weng claims 'richer' information may be extracted from the signal using the *Windowed Fourier Phase (WFP)* information instead of zero-crossing data [70]. Meanwhile, Tong et al have shown how a '*filter bank approach*' results in greater signal information than a zero-crossing approach [66].

Nevertheless, as Basztura et al [7] showed a zero-crossing method to be most suited to small vocabulary systems (owing to its low cost and easy implementation) and as this is the main specification of the proposed system the pessimistic attitudes should be temporarily dismissed.

1.3 Speech Production.

When generating a speech sound, the lungs act as an air reservoir and bellows, forcing air between the vocal cords and causing them to vibrate, much like the double reed of an oboe (refer to Figure 1.3). The resulting sound is amplified as it resonates in the cavities of the chest, neck and head, and it is articulated, (shaping vowels and consonants), by the speaker's lips, teeth, tongue, and palate.

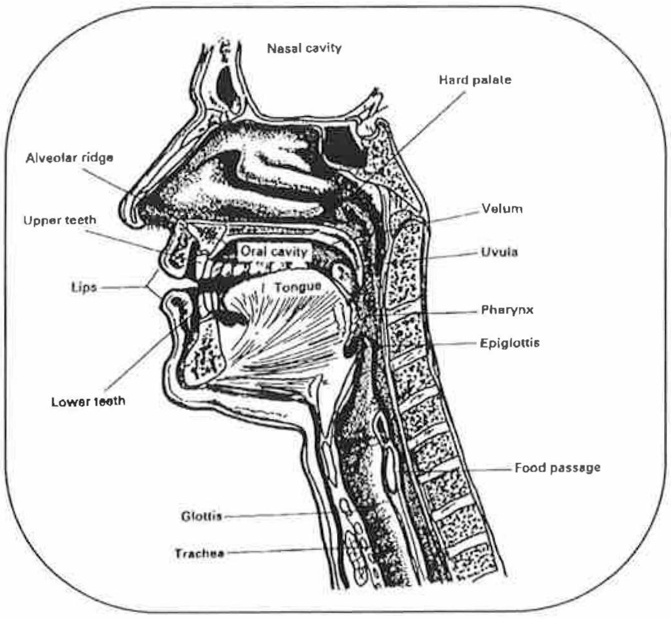


Figure 1.3: The Human Vocal Organs.

1.3.1 Voiced and Unvoiced Speech.

Voiced speech is produced by forcing air through the glottis while vibrating the vocal cords (refer to Figure 1.3). It is usually a quasi-periodic signal in the time domain and results in a spectrum of a clearly defined fundamental and harmonics at multiples of this frequency also known as the pitch frequency. Voiced speech tends to carry greater signal energy than unvoiced speech. For an adult male subject the pitch frequency may range from 50Hz to about 250Hz; for an adult female it may be as high as 500Hz.

If the log of the Discrete Fourier Transform (DFT) of a windowed section of voiced speech is calculated, a '*log-magnitude spectrum*' results, in which a 'slowly varying component' (due to vocal tract transmission) and a 'rapidly varying periodic component' (due to pitch) are very apparent (refer to Figure 1.4). For the average vocal tract there are three to five formants† below 5kHz, the first three being the most important for speech synthesis and recognition and usually lying below 3kHz [46].

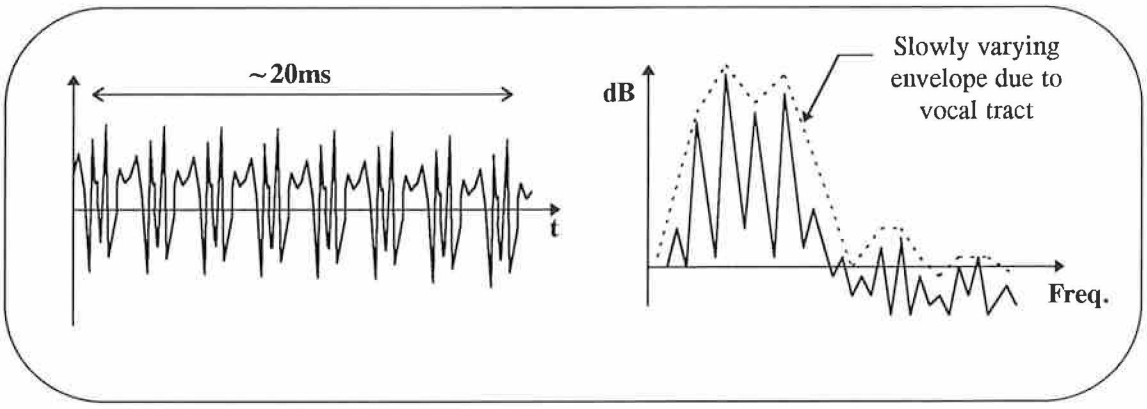


Figure 1.4: Section of voiced speech with its corresponding Log-Magnitude Spectrum showing a slowly varying component due to the pharynx and vocal tract and a rapidly changing component due to the pitch.

Unvoiced speech is produced when no vibrations take place in the vocal cords. Some constriction may be present at different stages of the airway (e.g. Teeth, lips, tongue etc.), which gives each sound its characteristic spectral shape. The resulting speech signal is non-periodic and random-like in the time domain and consists of a broad band spectrum (refer to Figure 1.5).

† Formants are resonance frequencies present in the vocal tract and are dependent on the tract length.

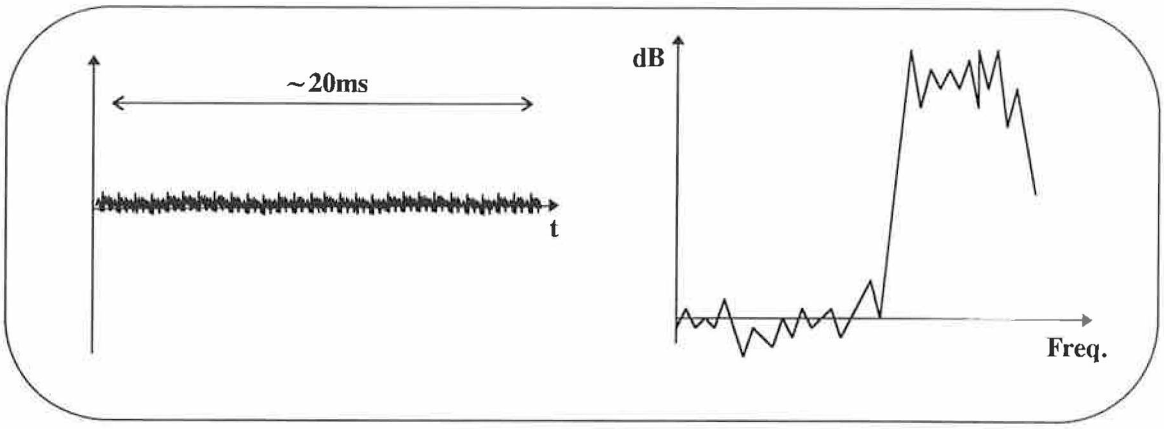


Figure 1.5: Section of unvoiced speech with its corresponding Log-Magnitude Spectrum.

1.3.2 Zero-Crossing Content of Voiced and Unvoiced Speech.

It has been proven that most of the energy in speech signals lies below ~ 4.5kHz [37]. The energy in voiced speech is mainly below ~ 3kHz and that of unvoiced speech is located at higher frequencies. High frequencies result in a high zero-crossing rate (ZCR) and low frequencies in a low zero-crossing rate. This would suggest that the assumption of classifying speech as voiced if a low ZCR is detected and as unvoiced if a high ZCR is present is valid.

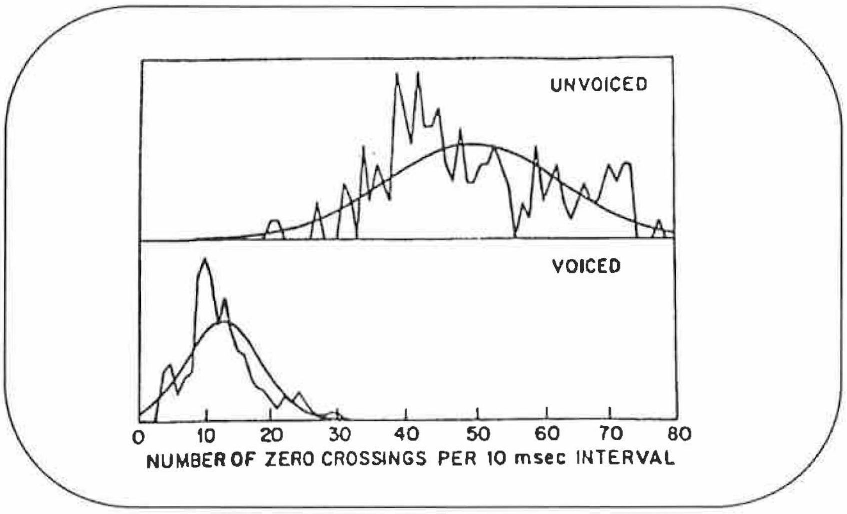


Figure 1.6: Distribution of zero-crossings for voiced and unvoiced speech (after Rabiner and Schafer).

Rabiner & Schafer demonstrated how the distributions of the ZCR for voiced and unvoiced speech are fitted quite well by a Gaussian curve (see Figure 1.6) [51] with mean short-time average ZCRs as follows[†]:

[†] These values are based on 10msec intervals.

Unvoiced	4900 Crossings per second.
Voiced	1400 Crossings per second.

As can be seen in Figure 1.6, an overlap exists between the curves which would suggest a simple voiced-unvoiced decision (VUD) on this parameter alone may prove insufficient.

It should be noted that the first formant, (being the most prominent spectral component), has the strongest influence on the ZCR in the case of voiced speech.

1.3.3 An Interpretation of the Spectra of Speech Signals.

The Fast Fourier Transform (FFT) has proven to be very useful in signal processing over the years. However, due to its computational expense, it is not always a preferred tool in speech recognition. Nevertheless, the FFT is still very helpful in acquiring a greater understanding of how speech signals are structured and what phenomena may influence these structures.

Figure 1.7 [60] shows 32msec of the phoneme /i/ as in '*bee*', (sampled at 16kHz), uttered by male and female subjects. Below each plot is the corresponding amplitude spectrum. The more powerful lower frequency components yield a clearly defined envelope in the time domain signal, while the weaker higher frequency components result in finer signal detail. Hence, it is clear that the male utterance, whose energy is concentrated in the lower frequencies, has a slowly varying envelope with little fine detail. However, the female utterance, consisting of weaker low frequency components and more significant higher frequencies, contains a less smooth, faster changing envelope. Understanding this can help in the analysis of the zero-crossing content of a signal.

To study the spectrum of a complete utterance would be of little use because nearly all frequencies would be present. As Lovel [33] claims: "Simple Fourier analysis of modulated signals (e.g. speech) gives no indication of changes in signal character

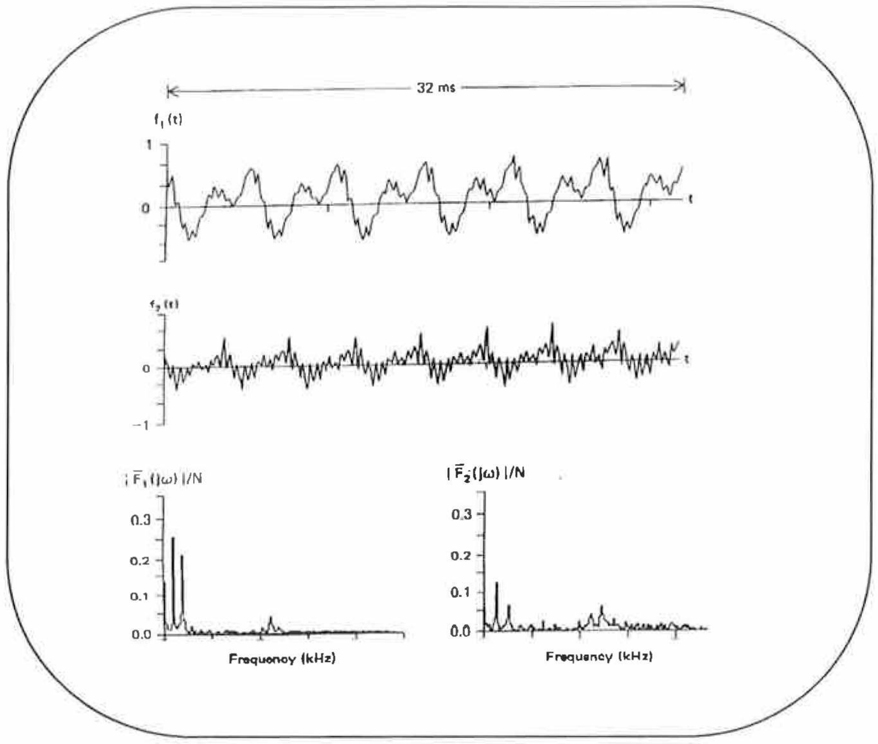


Figure 1.7: Speech signals of the phoneme /i/ as in '*Bee*' as uttered by male (f_1) and female (f_2) subjects and their corresponding spectra.

within the observation window." To overcome this problem various windowed versions of the FFT have been developed. An example of these is the Short-Time Fourier Transform (STFT) or spectrogram [19] as described in [30], which describes how the frequency changes in time (refer to Figure 1.8). Such a plot is referred to as a *Time-Frequency Representation (TFR)*.

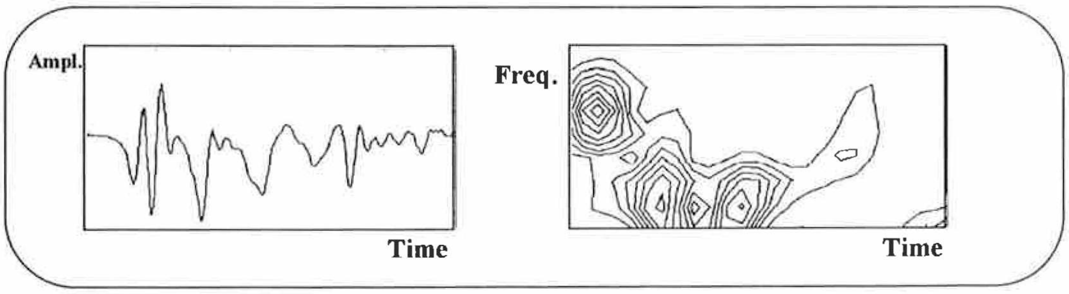


Figure 1.8: Speech signal of the phoneme /k/ as in '*Can*' and its corresponding spectrogram.

Other TFRs have been developed to reduce the problems of time and frequency resolution trade-offs [30] - Wavelet Transform, Wigner Distribution and Smoothed Wigner Distribution. Lawlor et al [30] found that in voiced speech the formant frequency values change slowly and so the spectrogram proves sufficient to describe the signal. However, fricative and plosive speech was found to change more rapidly and so a higher time-frequency resolution was necessary to track these changes. A

Gaussian Kernel Smoothed Wigner Distribution Function is suggested, owing to the suitability of a Gaussian kernel to the formant structure of unvoiced speech. This idea is reiterated by Lovel who claims that the Fourier Transform is only of use when the instantaneous frequency of the signal changes slowly with respect to the carrier signal [33].

1.4 Phonemes.

Definition: A phoneme is the basic sound unit in speech.

The English language comprises approximately forty phonemes - all words may be constructed from these alone. Although helpful in describing how a word is pronounced, phonemes are not perfect. Each phoneme may not always sound the same depending on the preceding and succeeding phoneme uttered. Allophones are different versions of the same phoneme. Dialect and the length of the vocal tract may result in the same phonemes being uttered differently.

The table in Figure 1.9 describes the set of phonemes implemented by the International Phonetic Alphabet (IPA). This table also gives examples of where the phonemes are used and the different categories into which they may be classified.

Each sound may be produced by shaping the vocal tract, changing the articulatory gesture (the position of the tongue, teeth, lips, etc.) or a combination of both. Vowels may be classed as purely voiced speech and are produced by vibrating the vocal cords and constricting the vocal tract with the tongue at the front, middle or back. Consonants may be classed as purely unvoiced (/ʃ/,/s/,etc.), partially voiced (/v/,/z/,etc.) or plosive (/p/,/b/,/t/,etc.).

1.4.1 Characteristic Properties of Phonemes [46].

- Vowels may be identified by the first three formant frequencies (usually low) extracted from the centre of the time domain signal and generally comprise a low ZCR.
- Diphthongs may be characterised by the formant frequencies of the initial and final vowel targets as well as the rate of change of formant trajectories† and contain a slightly greater amount of finer signal detail.

† That is, how the formant frequencies are changing with time.

Vowels	**Diphthongs**	**Fricatives**
Front		**Voiceless**
/i/ feet	/ɛi/ s*ay*	/s/ *s*it
/I/ d*i*d	/ai/ s*igh*	/ʃ/ *sh*ip
/ɛ/ red	/əʊ/ l*ow*	/f/ *f*at
/æ/ m*a*t	/ɑʊ/ b*ough*	/θ/ *th*in
	/iɜ/ d*eer*	/h/ *h*at
	/u/ d*oer*	
	/ɔi/ t*oy*	
	/ɛə/ d*are*	
Middle	**Affricates**	**Voiced**
/ɜ/ h*ear*d	/dʒ/ *j*ug	/v/ *v*an
/ʌ/ c*u*t	/tʃ/ *ch*um	/z/ *z*oo
/ə/ th*e*		/ð/ *th*is
		/ʒ/ azure
Back	**Nasals**	**Plosives**
		Voiced
/a/ c*ar*d	/m/ *m*an	/b/ *b*ad
/ø/ cod	/n/ *n*ow	/d/ *d*in
/ɔ/ bo*ar*d	/ŋ/ si*ng*	/g/ *g*one
/ʊ/ wood		
/u/ r*u*de		
Semivowels		
Glides		*Unvoiced*
/w/ *w*ent		/p/ *p*in
/j/ *y*ou		/t/ *t*on
Liquids		/k/ *k*ill
/l/ *l*et		
/r/ *r*an		

Figure 1.9: The phonemes of European English.

- Nasals and glides are always detected adjacent to vowels and may be identified by the formant transitions into and out of the sound (e.g. the word '*low*' sounds like "el⇗⇘ ow").
- Fricatives may be identified by the presence or absence of turbulent noise and often consists of a broad band spectrum with a very high ZCR.
- Plosives may be detected by a characteristic period of silence followed by an abrupt increase in signal level at the point of release which in turn is followed by fricative noise (e.g. /t/ in 'stop' sounds like "s___t≋≋op") and the ZCR depends on each plosive.

1.4.2 Phonetic Segmentation of Speech Signals.

The segmentation of speech signals into their individual phonemes has always proven to be a problem - one which if solved would allow an easier and more efficient analysis of speech signals. Transition from one phoneme to the next during analysis would no longer prove a problem yielding more regular results.

In 1967 Reddy [52] demonstrated how zero-crossing information may be used to segment a speech signal into portions, (i.e. approximate phoneme units) and then performed Fourier analysis on each segment.

Basztura [7] showed how the signal may be segmented by detecting the phonetic boundaries using zero-crossing information, spectral data and linear predictive coding. No significant advantages of one method over another were found.

However, research by Lovel [33] discovered that the Appel & Brandt† algorithm proved to be the most effective means of detecting a phonetic boundary.

1.5 Stationarity of Speech Signals.

Definition: A signal or process is said to be stationary if its statistical properties do not vary after a shift in time [34].

A more simplistic definition of stationarity is that the frequency content does not change in time [30].

---

† Appel & Brandt is a distance measure between sample values and is sensitive to both spectral shape and signal energy.

Speech signals may therefore be classified as '*piecewise quasi-stationary*' signals. This is so, as each section of the signal (e.g. a particular vowel) remains locally 'almost' stationary, but a global view of the signal shows it to be non-stationary. This may be explained by the speech signals in Figure 1.10.

The larynx (voice box) may therefore be described as a quasi-stationary source. A repeated vowel sound results in almost the same signal, as shown in Figure 1.10. It should be noted that this slow variation of frequency content will affect the zero-crossing rate of the signal. Hence, when working with signals that are not quite stationary, it may simplify matters if the signal is stationarised first. Kedem [25] explains some simple procedures to stationarise a signal.

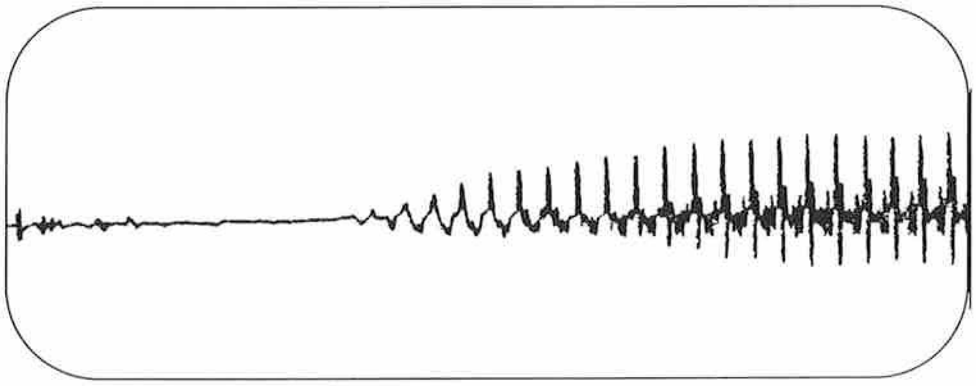


Figure 1.10: Syllable /ta/ showing quasi-stationarity of speech.

1.6 Pre-processing of Speech Signals.

Before applying any analysis routines to a speech signal, it is important to ensure that the signal being received is a true representation of the speech sound. A number of pre-processing steps are necessary to achieve this, some of which are explained in this section.

1.6.1 Pre-emphasis of Speech Data [37].

It has been found that for voiced speech there is an overall trend of approximately -6dB/Octave as frequency increases [46]. This is composed of the excitation source (-12dB/Octave) and radiation from the mouth (+6dB/Octave). In order to remove these effects from the final speech signal a pre-emphasis filter is used. A general first order pre-emphasis of speech data may be achieved using the filter:

$$P(z)=1-\mu z^{-1},$$

as shown in Figure 1.11.

A *first order* filter was found to be the most suitable so as not to introduce ill-conditioning[†] into the signal [37]. In order to maximise spectral flatness at the output, an optimum value of $\mu=r(1)/r(0)$ is chosen, where $r(n)$ is the nth correlation coefficient.

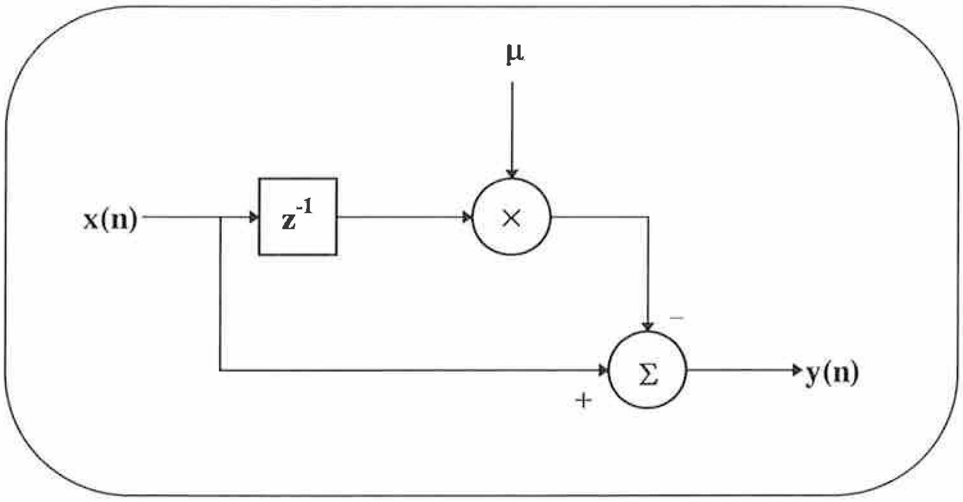


Figure 1.11: A first order pre-emphasis filter.

For most voiced sounds μ lies near one, owing to the fact that the sample amplitudes are highly correlated. This results in an approximate differencer (HPF) removing the spectral trend described above and reducing the effects of background low frequency noise.

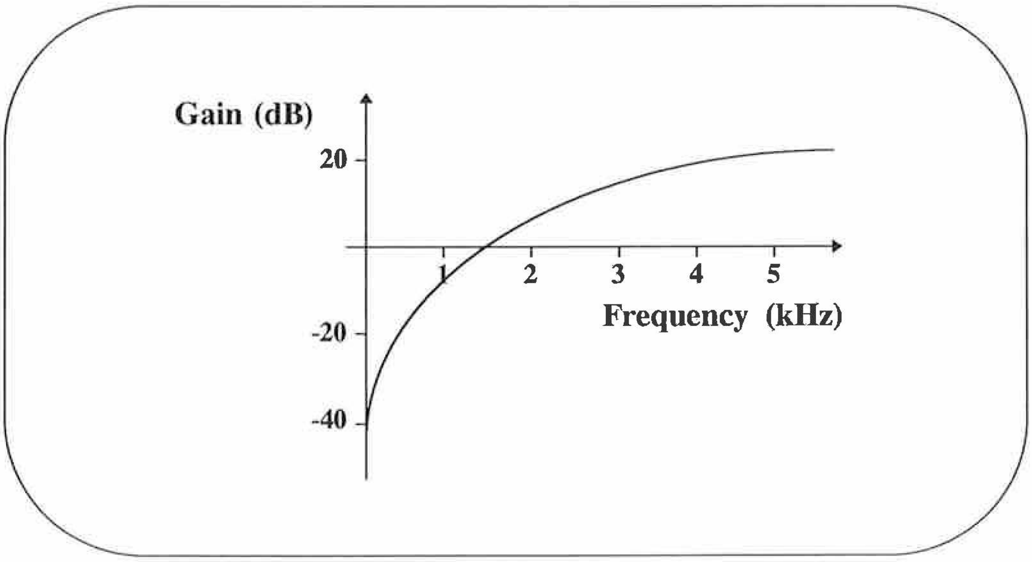


Figure 1.12: Frequency response of a pre-emphasis filter with $\mu=0.9$ and $T=100\mu s$.

[†] Ill-conditioning is the introduction of error into the signal data during mathematical computation.

For most unvoiced sounds μ is relatively small, due to the fast changing nature of the signal and so, the pre-emphasis filter has little effect. This is desirable as unvoiced speech does not exhibit any spectral trends.

A frequency response of a typical pre-emphasis filter with $\mu=0.9$ and $T=100\mu s$ is shown in Figure 1.12.

The pre-emphasis filter coefficient can be severely quantised since any value of μ between 0 and 2*(r(1)/r(0)) will enhance spectral flatness.

1.6.2 Pre-filtering before Sampling [37].

An analogue speech signal, like any other signal, must be filtered before sampling to ensure against aliasing.

An analogue filter of cut-off frequency $<<f_s/2$ is chosen in many signal processing applications. However, if this is applied in the case of a speech signal, it will increase the spectral dynamic range, decrease spectral flatness and increase ill-conditioning.

It was found instead that a cut-off frequency of $f_s/2$ or 'slightly below' resulted in a clearer signal description [37]. Care should be taken to ensure as little ripple as possible in the passband to avoid any distortion of the frequency components in the signal.

Chapter 2

2. Front Line Processing Technique I : Sinusoidal Instantaneous Frequency Extractor (SIFE).

This section introduces the first zero-crossing technique (SIFE) to be examined as a possible means of recognising speech. The SIFE system utilises information about the spacing between successive zero-crossings to generate a signal. This parameter is examined to establish if it contains enough signal information about the utterance to distinguish it from other utterances. This chapter shows how the SIFE can extract such signal information both through hardware and software. The quality of the SIFE signal is discussed and certain techniques are described in an attempt to improve this quality. The system's ability to distinguish different signals and reproduce consistent SIFE signals for utterances of the same word is also discussed. Finally, the various recognition and decision techniques are examined to reveal the most suitable one for the system and the SIFE's future as a speech recognition system is discussed.

2.1 Background to SIFE.

The SIFE system was first developed by de Paor [14] as a means of achieving a visual representation of a speech signal in order to retrain vocally impaired patients (through head injuries or strokes) to speak. The SIFE representation of the patient's speech signal would be displayed on a split monitor underneath a template SIFE achieved from an uninjured subject. The patient would then be assisted by a speech therapist as to where they require improvement. The system proved very successful as a speech signal visual representation. Later work by Coyle [12] attempted to implement the SIFE system as a means of speech recognition. The system was constructed in hardware and a C-program was used to calculate two parameters (mean square energy and absolute mean)[†] which were used to recognise the utterance. The system was tested using the vocabulary as mentioned in the Gr I set and although only basic recognition procedures were implemented, Coyle achieved quite reasonable success rates (between 70-90%). However, the system in [12] was never tested to its limits. Hence, this zero-crossing analyser will be the first to be studied in this thesis.

[†] Mean Square Energy = $\Sigma(e^2/N)$, Absolute Mean = $\Sigma(|e|/N)$.

2.2 How the SIFE System Operates.

The idea is to derive a signal whose amplitude is proportional to either the interval between consecutive zero-crossings or the instantaneous frequency present. A SIFE signal may be extracted from a speech signal as shown in Figure 2.1. The speech signal is first clipped by applying a comparator (in the form of a Schmitt trigger) resulting in what is termed *'Infinitely Clipped Speech'*. Hence, all amplitude information is discarded from the signal (refer to Figure 2.2). This signal is then differentiated yielding a series of positive and negative going impulses representing the presence of a zero-crossing. This pulse train is then rectified in such a manner as to yield the only positive or negative pulses[†]. These pulses are now used to reset an integrator that ramps up to each impulse and whose value is sampled and held at the instant before reset. The resulting signal (SIFE'=1/SIFE) is then inverted to yield the SIFE signal. SIFE' gives *instantaneous period* information (i.e. timing information between zero-crossings) while the SIFE gives *instantaneous frequency* information.

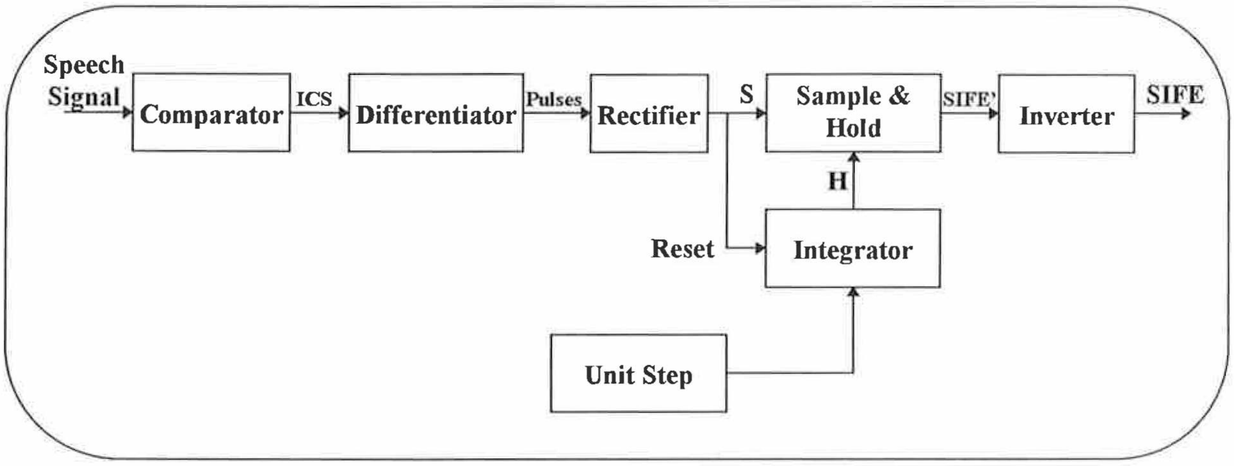


Figure 2.1: The production of a SIFE signal.

Four important points to be noted about the resulting SIFE signal may be described as follows:

- The amplitude at time t_n is proportional to the distance between ZC_n and ZC_{n-1}.
- A flat response implies a constant period is present (i.e. the speech signal is crossing the zero-axis at a constant rate) and a rapidly changing SIFE signal implies the presence of constantly changing periods.

[†] Full-wave rectification will result in too much signal information.

- A cyclic SIFE signal implies the presence of cyclic periods (i.e. recurring periods).
- SIFE' and SIFE contain practically the same information (except quantisation effects).

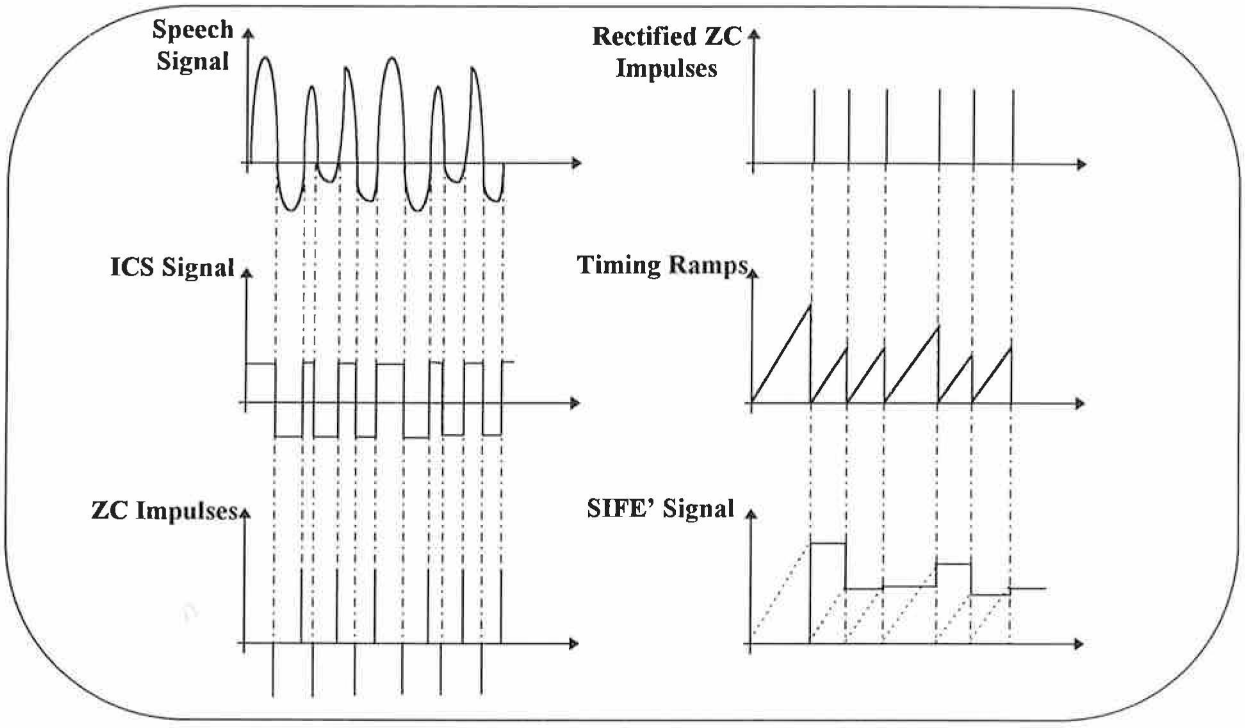


Figure 2.2: Waveforms generated in the extraction of a SIFE signal.

2.3 Difficulties with SIFE Hardware.

The SIFE system used by Coyle in [12] derives a SIFE signal as described in Appendix B1. As this appendix explains, it was necessary to write a number of statistical algorithms in order to improve Coyle's system.

After testing this system, it was discovered that it was extremely sensitive to background noise and so very irregular results were achieved. The software was also problematic with the PC crashing on numerous occasions. Hence, it was decided that it was necessary to (i) record the speech signals in a quieter environment (overcoming the need for an expensive microphone) and (ii) rewrite the software so that a more user-friendly and reliable package was available for testing[†].

[†] Refer to C-code in Appendix E.

2.3.1 Initial Results from SIFE Hardware and C-Program.

This newly updated software was then tested but problems still arose. The software itself proved to be very effective yet the resulting SIFE signal appeared to be very distorted. The following statistics were calculated for each SIFE signal in an attempt to characterised each utterance: *mean square energy*, *absolute mean*, *variance*, *coefficient of variation*, *area under SIFE signal* and *first delay value to give zero in autocorrelation function*. The results are shown in Table 2.1 and it is clear that these values are of little help in recognising the utterances. There exists much overlap in the values for different words and very little consistency and between like words.

After much investigation, it appeared that this distortion was due to the hardware's sensitivity to ambient noise and a problem with a straying offset in its output signal. Henceforth, it was decided, after some consideration, to design a software simulation of the system instead of consuming a large amount of time designing noise reduction and offset control circuitry.

2.4 The SIFE Simulator: SIFESIM.

The development of a software simulation of the SIFE system seemed to be the only logical means to establish whether or not the SIFE parameter may be used in speech recognition.

SIFESIM, described in Figure 2.3, was designed in Simulink. When examining the SIFE parameter's capacity to recognise speech, this system has three main advantages over its hardware predecessor:

- All the sounds are pre-recorded and virtually noise-free.
- The absence of hardware means no interference of circuitry can arise.
- No timing or synchronisation problems are associated with the integrator as encountered in [12].

Utterance	MSE	Abs	Var	Cv	Area	Cross
Go	5.16	1.98	1.24	63	1.46	46
	7.14	2.23	1.48	96	3.72	37
	4.3	1.68	2.77	88	2.82	35
	2.5	1.36	0.66	49	3.3	42
	3.31	1.6	0.74	46	3.8	39
	2.91	1.5	0.64	43	12.2	124
	3.04	1.51	0.78	52	4.73	43
	2.79	1.45	0.68	47	5.3	48
	5.25	1.91	1.59	83	8.04	52
	3.53	1.61	0.94	58	6.09	44
Stop	20.56	4.43	0.86	20	12.2	61
	20.46	4.45	0.67	15	13.3	62
	22.61	4.72	0.33	7	14.2	59
	22.91	4.71	0.66	14	1.18	63
	21.72	4.6	0.65	14	2.57	70
	16.8	3.7	5.13	139	1.01	99
	20.69	4.42	1.12	25	2.11	68
	20.56	4.33	2.5	58	4.99	98
	17.6	3.89	2.68	69	1.64	106
	17.1	3.98	1.27	32	8.83	60
Reverse	8.98	2.46	3.76	153	19.74	96
	13.08	3.24	2.63	81	20.13	67
	9.28	2.56	2.56	97	37.58	151
	14.99	3.56	2.29	64	26.01	73
	9.07	2.48	3.72	150	0.83	90
	16.38	3.76	2.22	59	1.98	66
	14.46	3.45	2.56	74	2.42	78
	15.11	3.56	2.33	65	3.68	86
	15.57	3.75	1.49	40	4.46	70
	17.35	3.91	2.2	56	6.21	76
Left	13.61	3.41	2.03	60	1.2	70
	8.96	2.52	2.63	104	0.32	18
	14.31	3.58	1.43	40	0.24	17
	18.83	4.2	1.12	27	0.14	5
	7.58	2.29	2.31	101	0.52	19
	7.11	2.31	1.78	77	0.81	22
	10.98	2.85	2.84	99	1.68	31
	8.96	2.27	3.79	167	1.53	28
	15.1	3.52	2.68	76	0.36	26
	8.6	2.46	2.52	102	0.45	26
Right	2.78	1.25	1.22	98	1.53	30
	11.21	2.98	2.35	79	4.02	30
	9.82	2.86	1.74	61	7.36	54
	10.23	2.94	1.51	51	8.55	58
	13.42	3.55	0.78	22	11.73	60
	6.42	1.97	2.52	128	3.14	27
	18.33	4.11	1.38	34	6.23	24
	15.78	3.63	2.62	72	8.02	33
	7.98	2.38	2.3	97	4.79	28
	6.97	2.17	2.32	107	5.17	32

NOTE:

$$\mathrm{MSE} = \frac{\Sigma(x_n)^2}{N}$$

$$\mathrm{Abs} = \frac{\Sigma|x_n|}{N}$$

$$\mathrm{Var} = \frac{\Sigma(x_n-\bar{x})^2}{N}$$

$$C_v = \frac{\sqrt{\mathrm{Var}}}{\bar{x}}$$

Area = area under the SIFE calculated using the trapezoidal approximation.

Cross = the time (sample) where the autocorrelation function first crosses the abscissa.

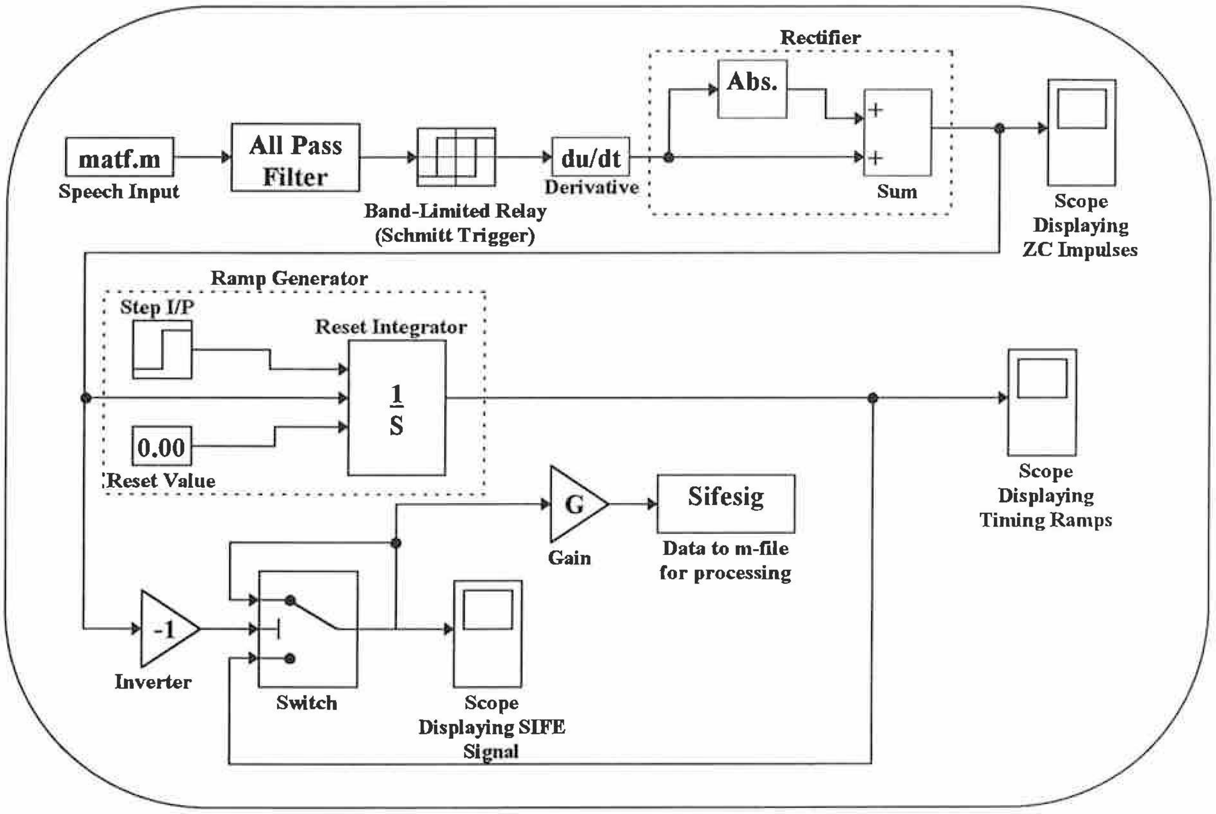
Table 2.1: Statistics for various SIFE signals using the hardware and C-program in Appendix E.

2.4.1 Description of SIFESIM Simulink Model.

SIFESIM operates in a very similar manner to its hardware counterpart. The model may be divided into three stages:

- The Zero-Crossing Impulse Stage;
- The Reset Integrator Stage;
- The Sample & Hold Stage.

An in-depth explanation of this stages and the necessary modifications is given in Appendix B2.



Rectifier
Abs.
matf.m
Speech Input
All Pass Filter
du/dt
Derivative
Sum
Band-Limited Relay (Schmitt Trigger)
Scope Displaying ZC Impulses
Ramp Generator
Step I/P
Reset Integrator
1/S
0.00
Reset Value
Scope Displaying Timing Ramps
G
Gain
Sifesig
Data to m-file for processing
-1
Inverter
Switch
Scope Displaying SIFE Signal

Figure 2.3: Simulink block diagram for SIFESIM.

2.4.2 Recording and Formatting of Matf.m.

Five utterances of each word in vocabulary set Gr I were recorded as described in Appendix A4 resulting in *.wav* files of speech data. These had to be converted into the appropriate Matlab format, (as described in Appendix B3), before they could be used on the model.

2.4.3 Results from SIFESIM.

Once again, the results from the statistical analysis proved unsuccessful. However, owing to the fact that the system was in the form of a software model, any aspect of the process could be examined and altered if necessary. It is clear, simply by examining the plots in Figures 2.4-2.8, that the SIFE signals for the utterances '*Stop*', '*Left*' and '*Right*' are not very different. As may be observed from Figure 2.9, the SIFE parameter is not reliably repeatable either. These (distinguishability and repeatability), being two of the most important factors in a speech recognition system, had to be improved upon before SIFESIM could be seriously considered as a speech recognition system.

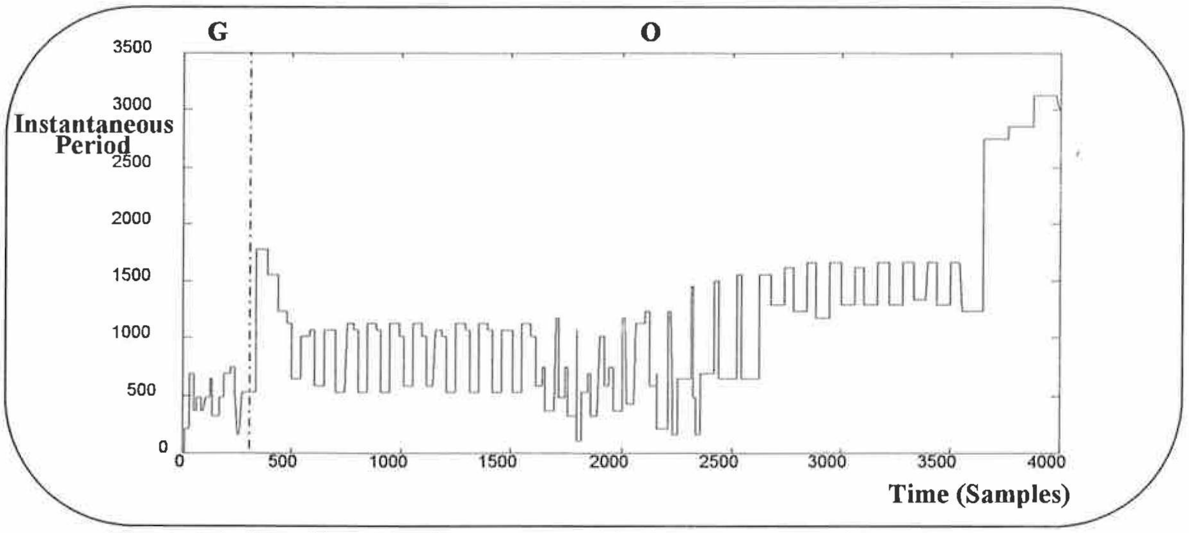


Figure 2.4: SIFE Signal for the utterance '*Go*' with approximate phonemic sections.

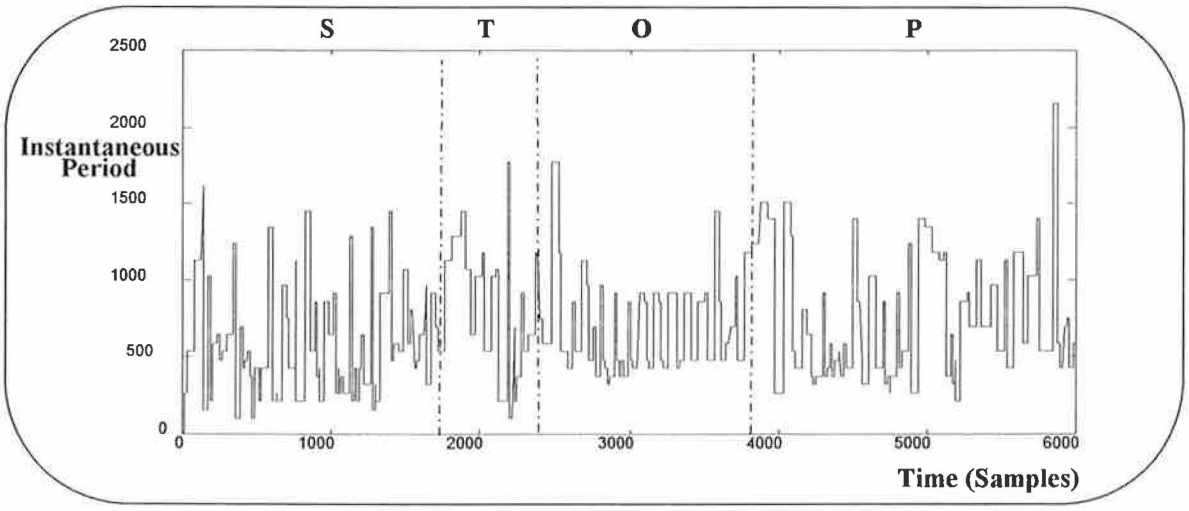


Figure 2.5: SIFE Signal for the utterance '*Stop*' with approximate phonemic sections.

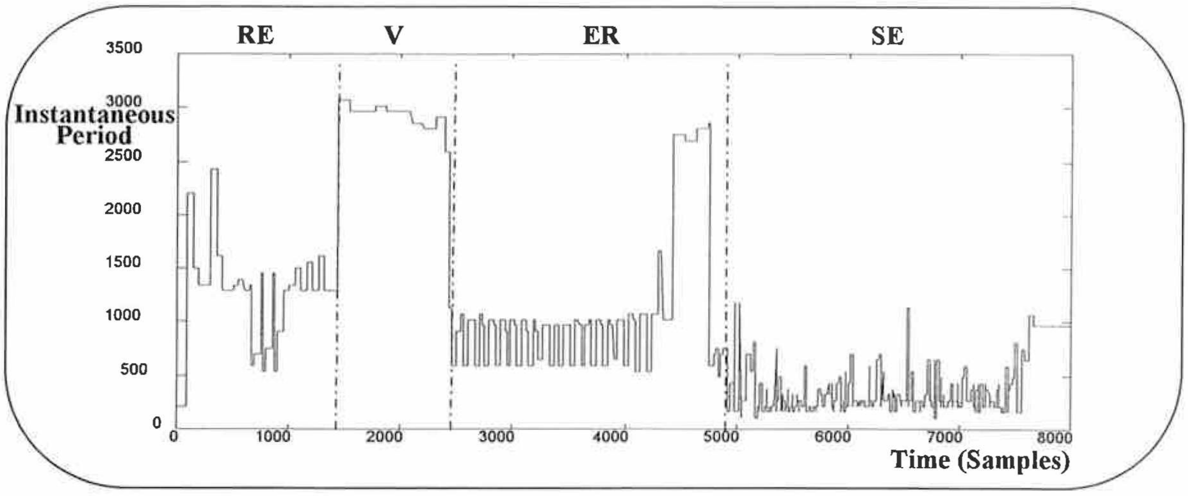


Figure 2.6: SIFE Signal for the utterance '*Reverse*' with approximate phonemic sections.

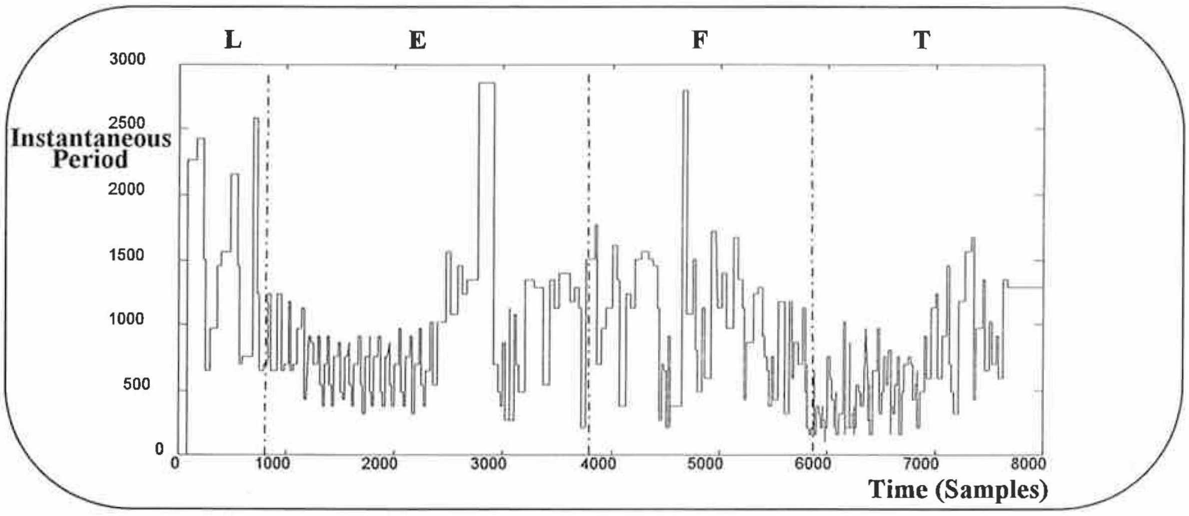


Figure 2.7: SIFE Signal for the utterance '*Left*' with approximate phonemic sections.

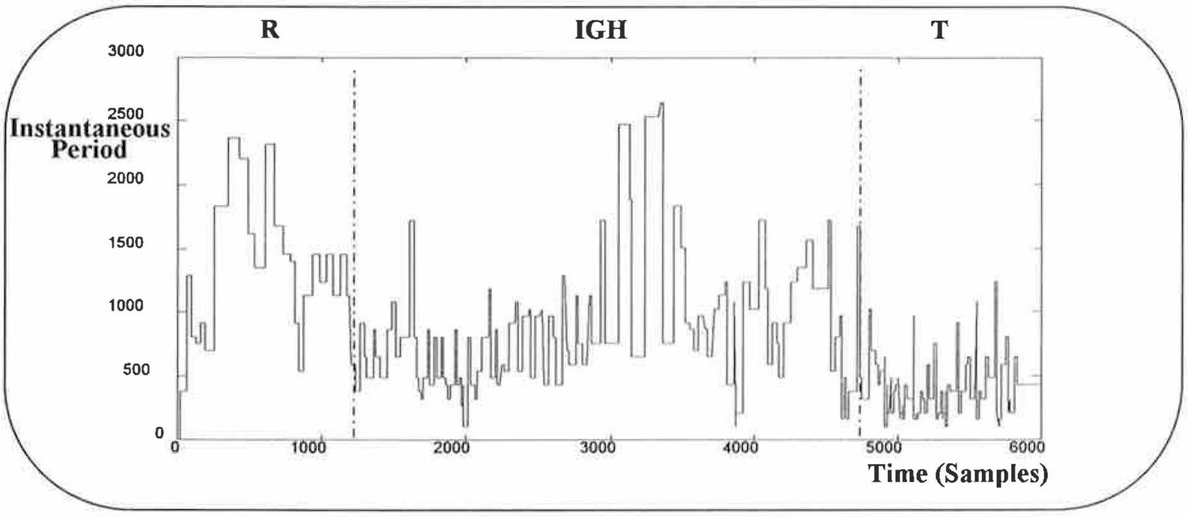


Figure 2.8: SIFE Signal for the utterance '*Right*' with approximate phonemic sections.

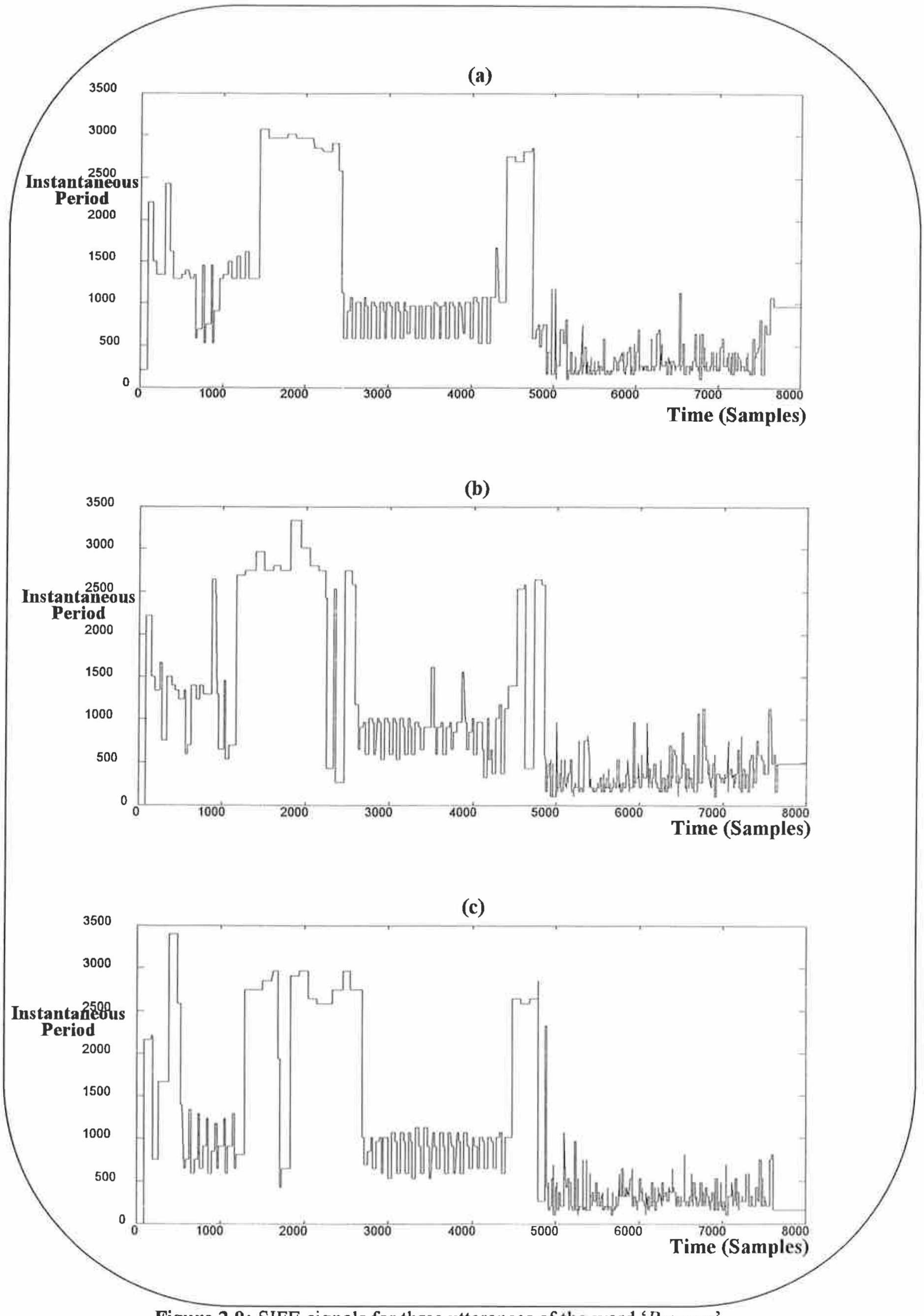


Figure 2.9: SIFE signals for three utterances of the word '*Reverse*'.

2.4.4 Alterations to SIFESIM.

The following test was executed on SIFESIM to explain the problematic aspects associated with the generation of a SIFE signal and to determine the necessary modifications to improve the quality of the signal.

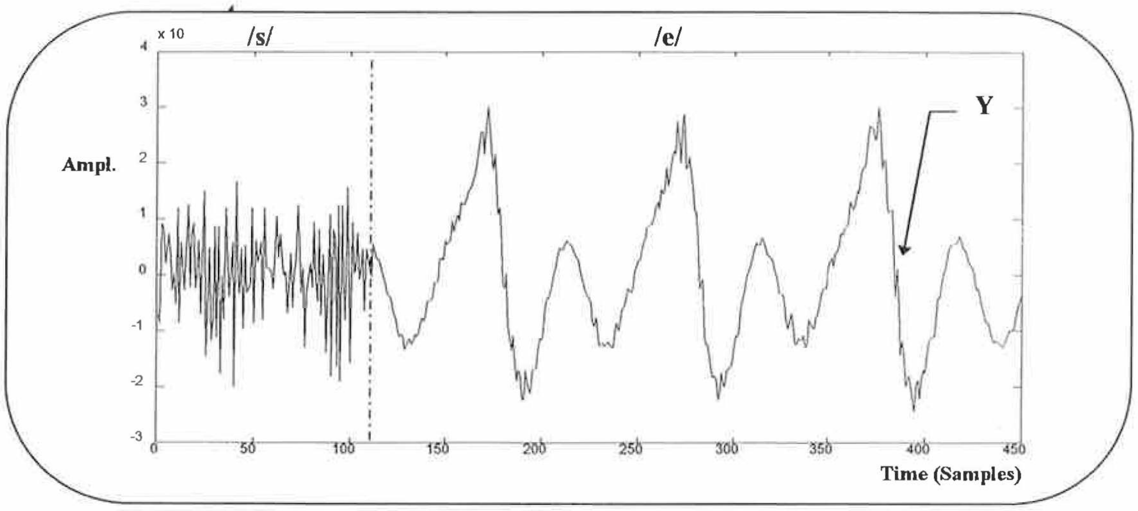


Figure 2.10: A 40ms section of the artificially joined sounds /s/ and /e/ showing a slightly non-stationary vowel section.

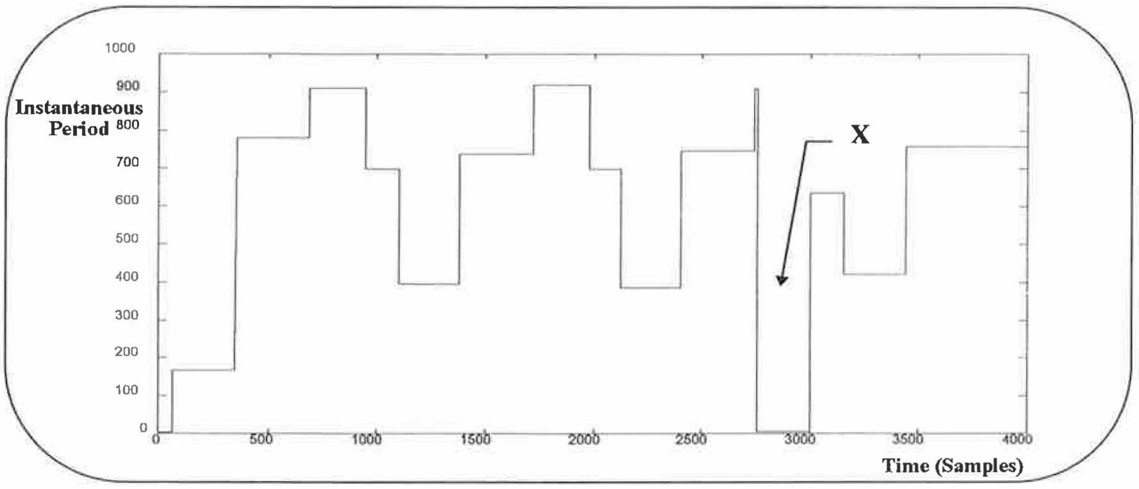


Figure 2.11: The SIFE signal for the /e/ section of the signal in Figure 2.10 clearly, demonstrating the effect of the slight non-stationarity.

A 40ms section of sound was created as a test signal (Figure 2.10), by artificially connecting sections of the sounds /s/ and /e/ recorded at 11kHz (16 bit). A 'zoomed-in' view of the /e/ section of the corresponding SIFE signal is shown in Figure 2.11. The high frequency glitch at point X clearly causes difficulty in the interpretation of the SIFE signal as this section of the signal, being a vowel sound, should really result in a fairly constant high SIFE' value (low SIFE value). This glitch is in fact due to the slight yet significant crossover at point Y in Figure 2.10 which highlights the quasi-stationary nature of speech signals. In the next subsections a number of possible alterations are discussed to overcome this problem.

2.4.4.1 Low Pass Filter Approach.

Rather than attempting to stationarise[†] the speech signal, (which can prove to be a slow process), a simple *low pass filter* in the form of a summation filter was applied to the signal before SIFE analysis. However, due to the large number of samples (a result of a high sampling rate) in each 'blip' (such as X in Figure 2.11) an extremely high order filter was required and so this approach was discarded.

2.4.4.2 Deadband Approach.

The next approach to be considered was to insert a *deadband* into the Schmitt trigger in the SIFESIM model. With a deadband the output of the relay may only turn *on* when the input reaches $+U_{Th}$, (instead of the rising edge zero as before), and may only turn *off* when the input reaches $-U_{Th}$, (instead of the falling edge zero as before). Figure 2.12 clearly explains the operation of the deadband. It was found that a $|U_{Th}|$ value of ~2% of the maximum amplitude was necessary to eliminate the effect of the 'blip'.

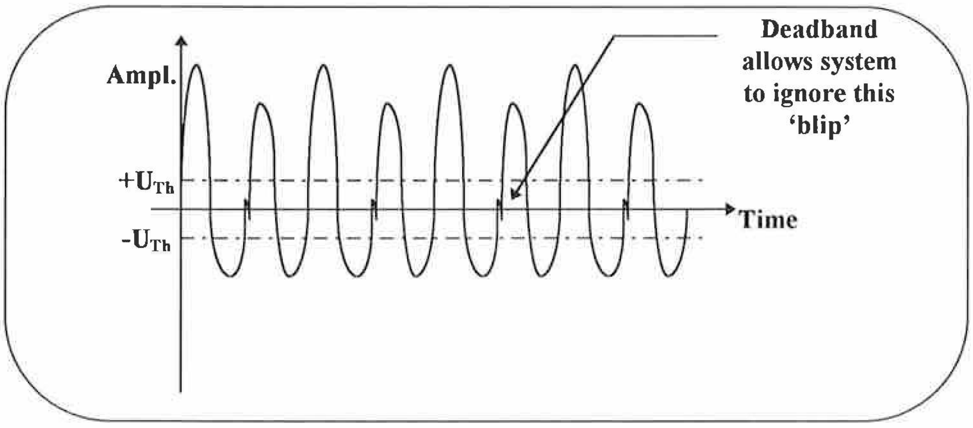


Figure 2.12: A deadband approach removes the effect of small blips in speech signals.

[†] The process of stationarising a signal is briefly discussed in Section 1.5.

It was initially feared that a deadband would affect the low energy content of the /s/, however on testing no problems arose. Yet on testing this system on a 'real' utterance, the results were unsatisfactory: the high frequencies *were* affected by the deadband; the presence of some were not even acknowledged by the system. For example, the deadband proved successful for the utterance '*Go*' (Figure 2.13), but the utterance '*Stop*' required a $|U_{Th}|$ value of 12% of the maximum amplitude to remove the unwanted crossovers which actually removed the /s/, /t/ and /p/ sections of the signal (refer to Figure 2.14). Hence, this approach also proved unsatisfactory.

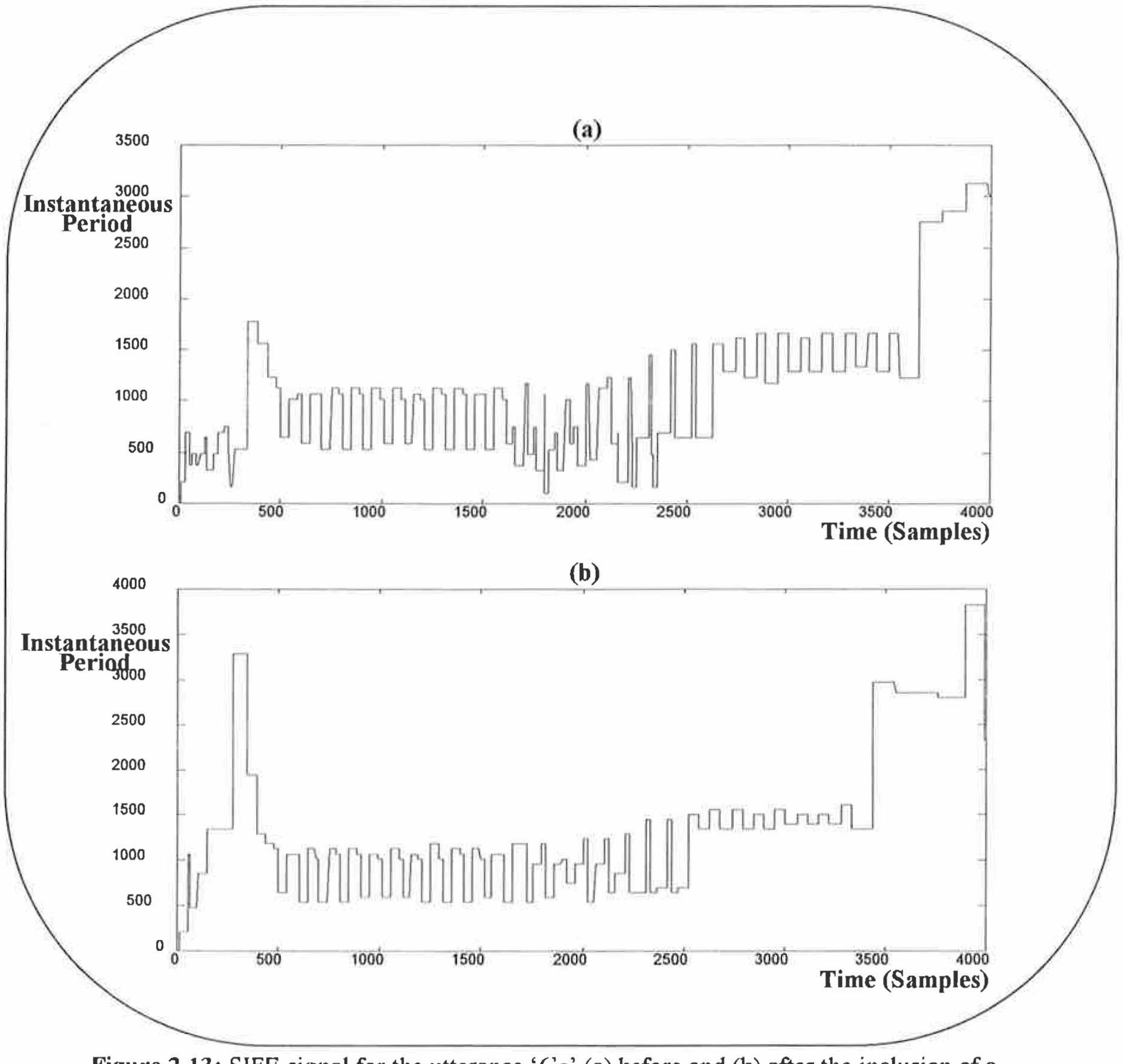


Figure 2.13: SIFE signal for the utterance '*Go*' (a) before and (b) after the inclusion of a deadband.

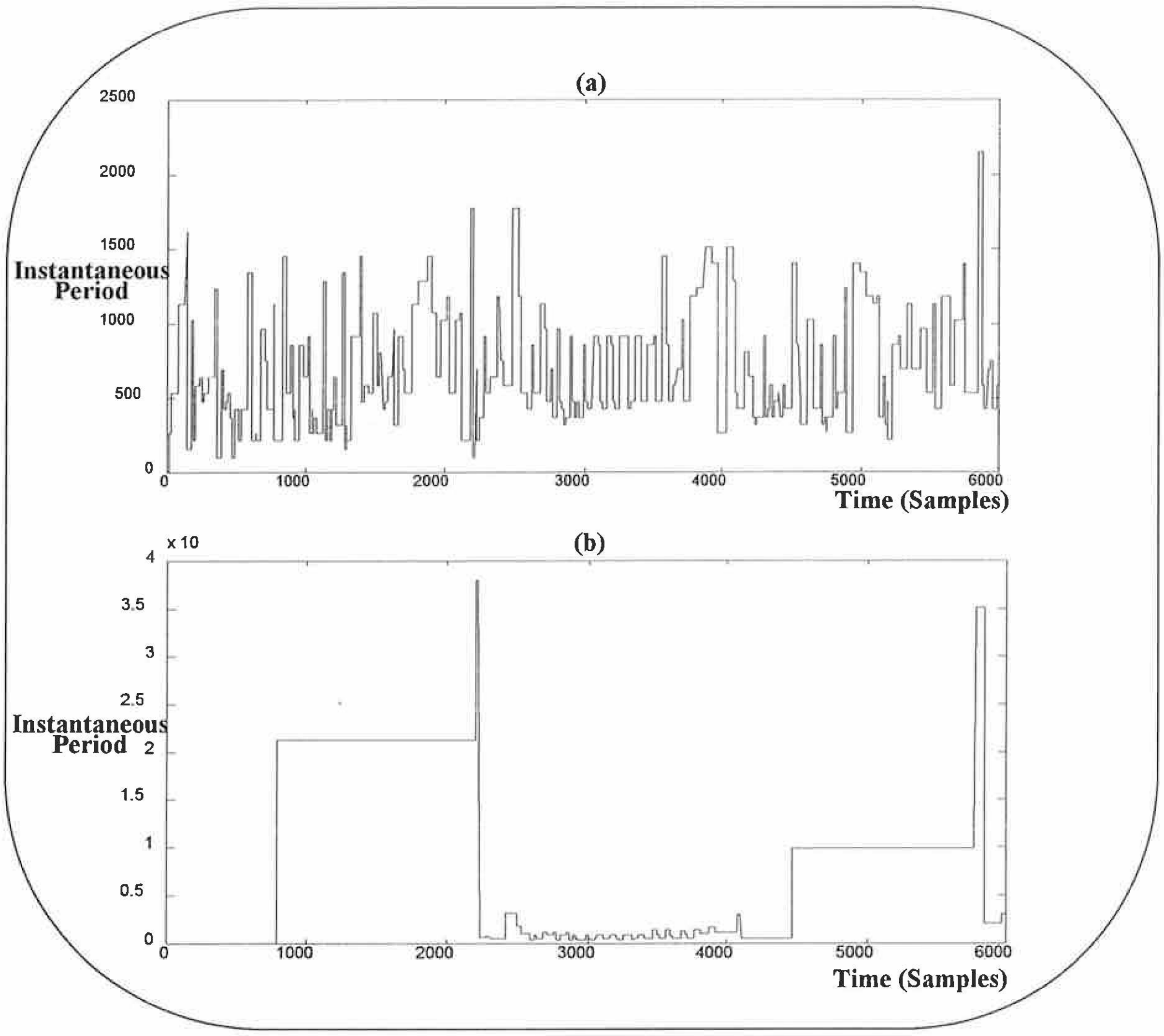


Figure 2.14: SIFE signal for the utterance '*Stop*' (a) before and (b) after the inclusion of a deadband.

2.4.4.3 Smoothing Filter Approach.

Another approach considered was to apply post processing to the SIFE signal itself. It was clear that some sort of smoothing filter was required to dampen the erratically fluctuating nature of the SIFE signal. The simplest technique to remove these spurious signal changes would be to apply a *linear low pass filter* to the SIFE signal. However, this would have the disadvantage of smoothing out any rapid signal changes resulting from a transition from say a voiced to an unvoiced speech section. A clean and sharp transition would be desirable. Also, a linear filter would give equal weighting to *all* samples in the signal, thus '*blending in*' any error samples instead of removing them.

Instead, a *median filter* was chosen because it preserves sharp signal transitions while eliminating any fine irregularities and outliers. The median filter chooses a single value as its output sample rather than a combined number of samples as in the case of a linear filter. An example of how a median filter operates is described in Figure 2.15 [45].

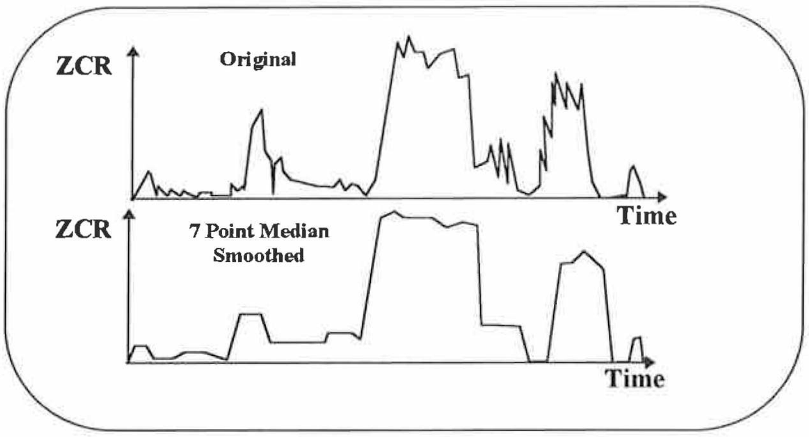


Figure 2.15: The effect of a median filter on a signal containing rapid fluctuations.

On applying the median filter to the SIFE signals, its effect became clearly evident as may be seen in Figure 2.16. The signal has now been '*cleaned up*' dramatically, however, a median filter of the order ~100 was necessary to achieve this standard due to the large number of samples present in these glitches. This proved to be extremely slow and so was not a very advisable approach.

2.4.4.4 Decreased Sampling Rate Approach.

The final method studied to improve the quality of the SIFE signal was to decrease the sampling rate of SIFESIM. The sampling rate of 11kHz was lowered to 8kHz (in the low pass filter block) and the resulting output signals were of similar quality to those of the median filter. An advantage with this technique is that it increases the overall speed of the process. The main disadvantage is that a certain degree of signal information is lost in the case of high frequency sounds such as /s/, /f/ and /ʃ/. Nevertheless, this being the only reasonable approach to improve signal quality, the reduction in high frequency information is a factor that may have to be accepted in order to allow the SIFE signal to be considered as a speech recognition parameter.

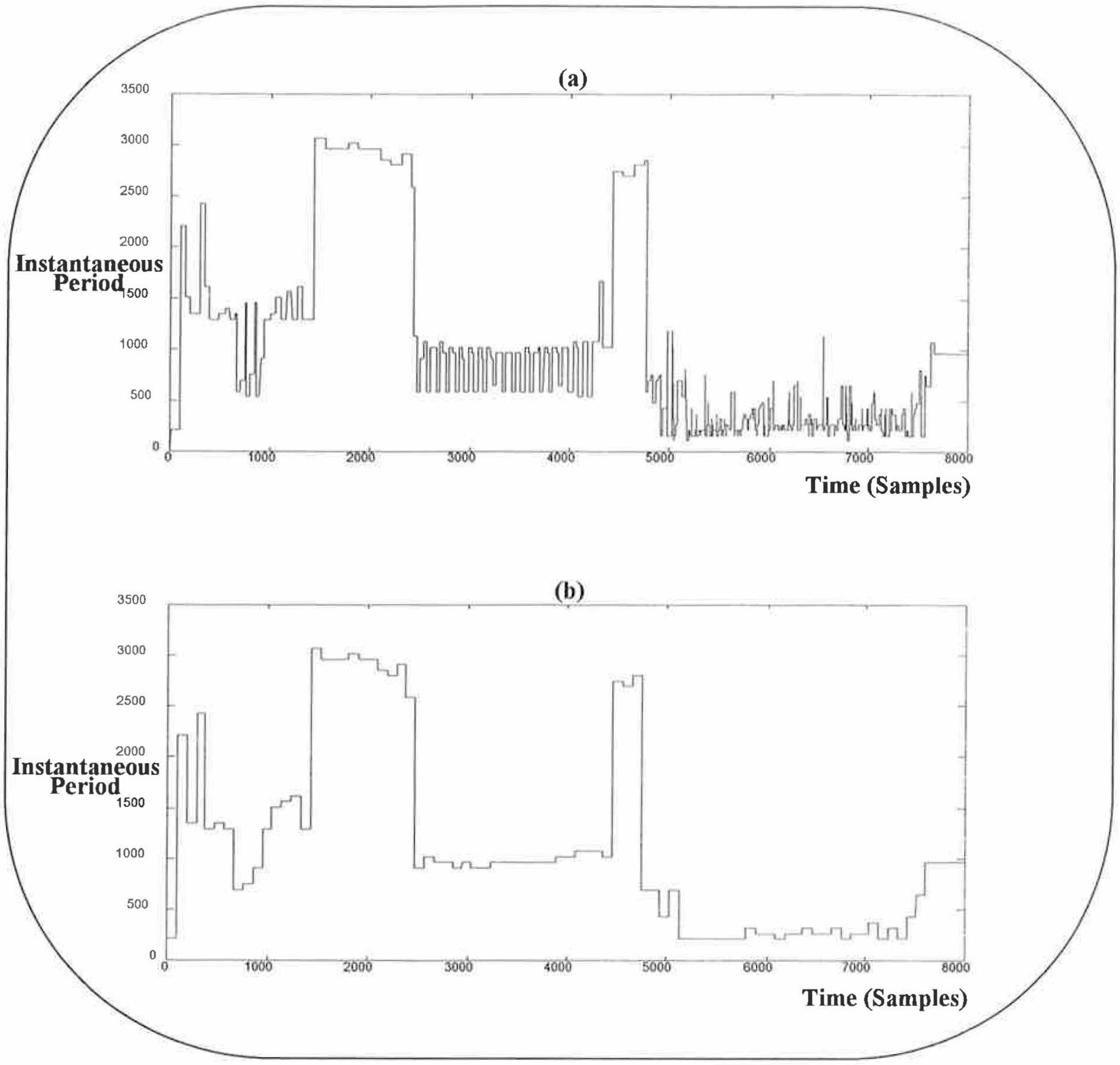


Figure 2.16: The SIFE signal for the utterance '*Reverse*' (a) before and (b) after the application of a *Median Filter*.

2.5 SIFE Recognition Stage.

It is clear from the results in the previous section that a SIFE signal is not a very reliable parameter on which to base a word recognition system. The exists very poor repeatability in the system for the same words and weak discriminatory powers for different words. Hence, the only possible recognition stage that may be implemented would be a *Voiced-Unvoiced Decision*†.

Although such a recognition routine is easy to implement, it is quite obvious that it would prove an unsuccessful means of speech recognition with the SIFE system for the following reasons:

† One such recognition routine is described in Appendix B4.

- There exists poor repeatability in the SIFE signal.
- The discriminatory powers of the SIFE signal have not proven to be satisfactory.
- As described in Section 1.3.2, there is a considerable amount of overlap between voiced and unvoiced zero-crossing data. Hence, the probability of a successful recognition being made using a voiced-unvoiced decision on a poor quality signal is slim.
- The SIFE signal tends to display certain sounds misleadingly. For example, the /v/ in the utterance '*Reverse*' as in Figure 2.6 is portrayed as a purely voiced sound and the high frequency content (the unvoiced component) is neglected (a problem with most zero-crossing analysers).

Chapter 3

3. Front Line Processing Technique II - Higher Order Crossing Analysis.

This section of the thesis introduces the second zero-crossing method to be investigated as a possible speech recognition technique. The background of the signal analysis technique known as Higher Order Crossing (HOC) analysis is first described, indicating the reasons why it was considered as a speech recognition technique. The theory behind HOC analysis is then described in detail. As the main reference in this section of the thesis is [25], a synopsis of Kedem's findings is also given, since a knowledge of his research is important in understanding further work into HOC analysis. In this synopsis such factors as the effects of dominant frequencies in a signal, the convergence to the highest present frequency, the detection and estimation of discrete frequencies and the effects of linear filtering are examined. A section discussing the choice of filters to be used in HOC analysis and how they affect the signal then follows.

3.1 Background to HOC Analysis.

HOC analysis was originally developed for the detection of signals in noisy environments [25]. An example of this is demonstrated in Figure 3.1 which shows a typical *log spectrogram* of the vocal sound of a whale and what is termed a '*HOC-gram*' of the same signal. In this case the noisy environment is the ocean. It can be seen that a clearly defined pattern is present in the HOC-gram, yet no such signal is detected by the log spectral-gram. Much research into a variety of uses of HOCs has been carried out based on this power of signal detection.

In [25] Kedem suggests that conducting HOC analysis on speech signals may assist speech recognition. However, he does not investigate in any great detail the possibility of this application. Henceforth, this section investigates this phenomenon as a rival to the SIFE as a ZC technique.

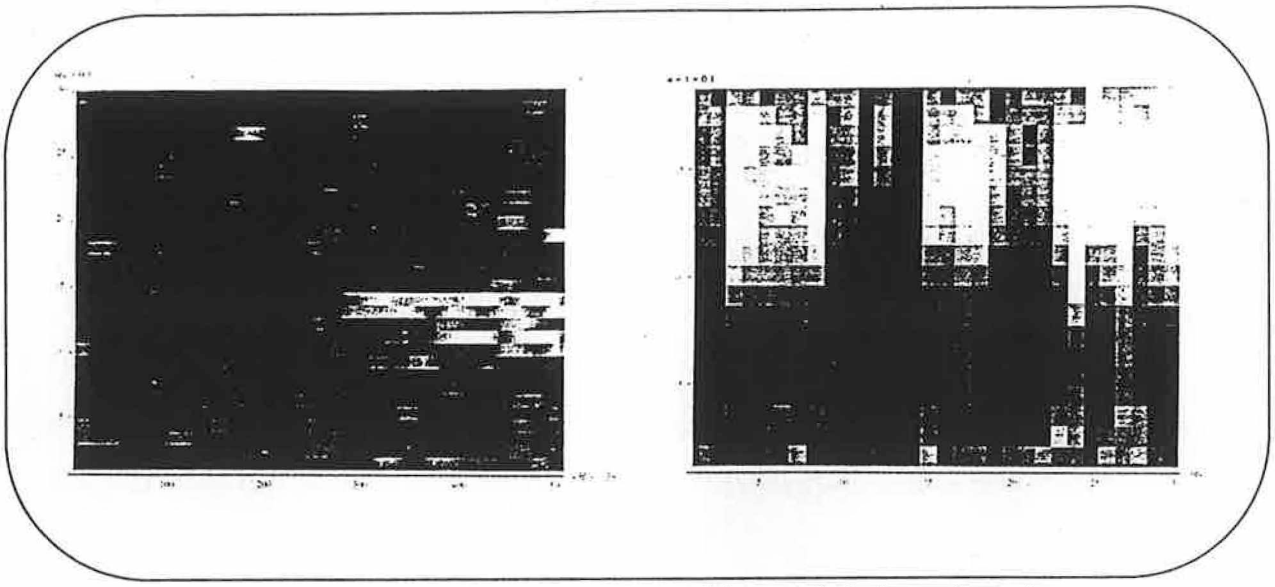


Figure 3.1: Log Spectral-gram and HOCgram of whale vocal sound.

3.2 Theory of HOC.

Definition: A Higher Order Crossing is present when a zero-crossing occurs in a filtered[†] version of the original signal. HOC information may be obtained from a signal using the system described in Figure 3.2. Simply by passing the signal through a series of N linear filters, N filtered (and so 'higher order') signals result.

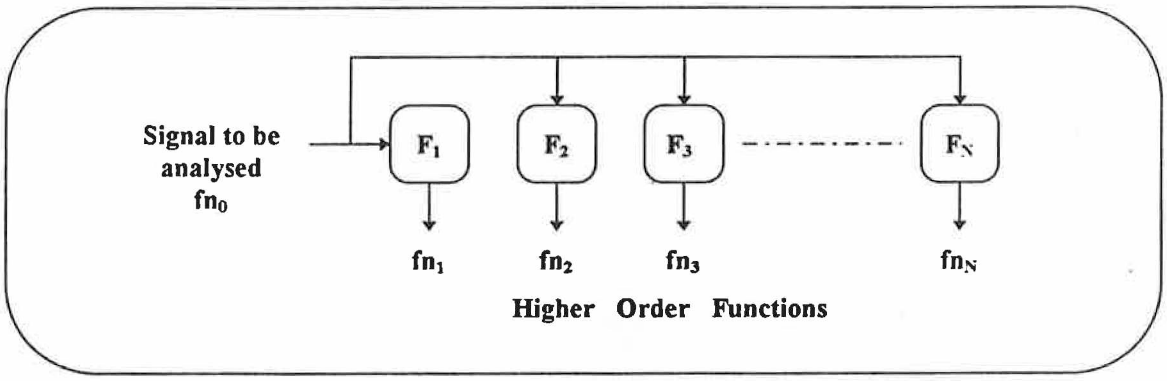


Figure 3.2: A simplified description of HOC analysis.

The application of a linear filter to any signal alters the oscillations and hence the zero-crossing data. It is this alteration of zero-crossings that will be studied and implemented as a means of speech recognition.

[†] The particular filter types are discussed in Section 3.3.

3.2.1 Dominant Frequency Principle [25].

If a series of linear filters is applied to a signal, the number of zero-crossings after normalisation[†] tends to approach the dominant frequency in the signal [25]. To explain the phenomenon known as the Dominant Frequency Principle consider a signal comprising two sinusoids. This signal as shown in Figure 3.3 [25] may be described by:

$$x_t = 3.86211\mathrm{Sin}(0.5t) + 1.11689\mathrm{Sin}(2t) \qquad t = 1,2,3,...,1050.$$

Counting the number of zero-crossings (D_0)[‡] in the section t=51...1050, a value of 159 is obtained and if this is then normalised the following results:

$$\omega_A' = \frac{\pi D_0}{N-1} = \frac{\pi(159)}{999} = .500013$$

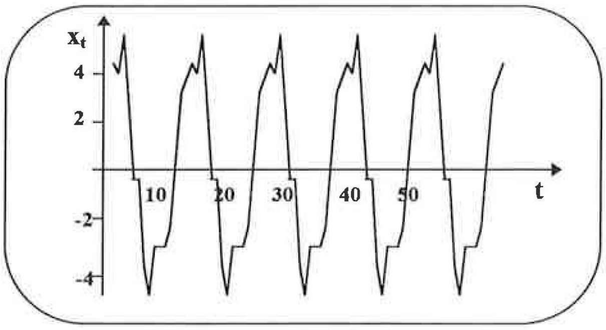


Figure 3.3: Original signal x_t.

It is clear that ω_A' is an estimate of the frequency ω_A=0.5 which is dominant as its amplitude is significantly greater than that of ω_B=2.

However, after applying a second order differencing filter to x_t, a new signal is obtained as shown in Figure 3.4 and may be described by:

$$y_t = x_t'' = x_t - 2x_{t-1} + x_{t-2}$$

Counting the number of zero-crossings in the same time frame as before, a value of 636 results for D_2.

After normalising, an estimate of ω_B is acquired:

$$\omega_B' = \frac{\pi D_2}{N-1} = \frac{\pi(636)}{999} = 2.00005$$

(Note: D_1 gives an average value between the two frequencies present).

[†] Normalisation: ($\pi \times$ No. of ZCs)/(No. of Samples - 1).

[‡] D_j is the number of ZCs after j filtering stages (i.e. jth order HOC data).

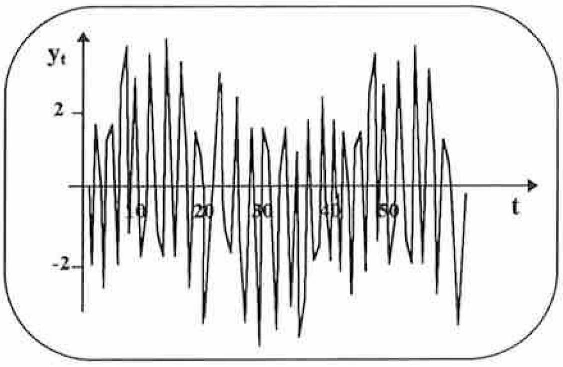


Figure 3.4: Twice differenced signal $y_t = x_t''$.

However, if the amplitudes of the two sinusoids are such that neither frequency is dominant (e.g. $A_A=A_B=1$) the normalised zero-crossing count tends to land in between the two frequencies:

$$x_t = \mathrm{Sin}(0.5t) + \mathrm{Sin}(2t)$$

$$D_0 = 448$$

$$\omega' = \frac{\pi D_0}{N-1} = \frac{\pi(448)}{999} = 1.40884$$

Nevertheless, it cannot be denied that (D_0,D_2) contains very useful spectral information about the signal.

3.2.2 Convergence Rule [25].

The Dominant Frequency Principle shows how filtering (differencing in this case) a signal may yield a normalised zero-crossing count which approaches the dominant frequency, but if the signal was continually filtered what ultimate effect would this have on the zero-crossings?

As was observed in the example in Section 3.2.1, the normalised zero-crossing count approached $\omega=2$ after two filtering stages. If this signal is repeatedly differenced, the normalised zero-crossing count tends to converge to and remain at $\omega=2$ (as shown in Figure 3.5) as no higher frequency is present. This Convergence Rule holds true for all signals.

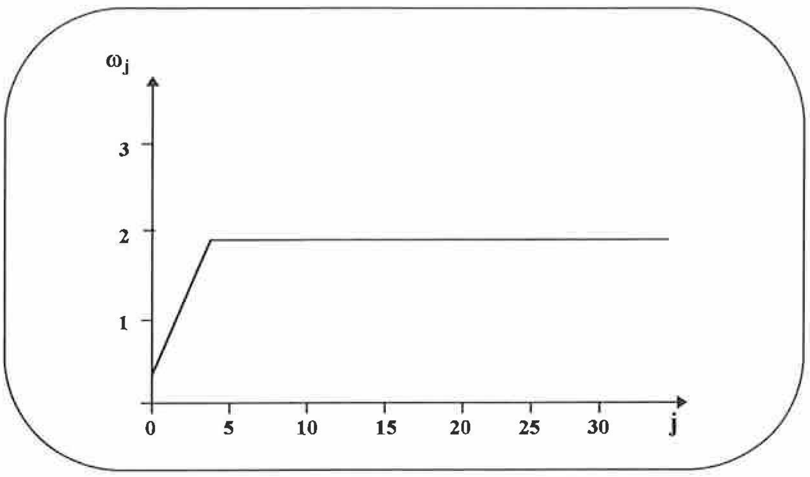


Figure 3.5: Convergence of normalised zero-crossing count after *differencing*.

3.2.3 Detection of Discrete Frequencies using HOCs.

The example in Section 3.2.1 not only describes the Dominant Frequency Principle but also how discrete frequencies may be detected. The appropriate choice of filters and the number of filter stages can result in a full description of any discrete frequencies present in a signal.

3.2.4 Autocorrelation.

It has been shown in [25] that there exists a direct relationship between the number of zero-crossings and the first order autocorrelation of a signal. For a stationary Gaussian process of zero-mean:

$$\rho_1 = \mathrm{Cos}\,\frac{(\pi E[D_0])}{N-1} \qquad \text{where } \rho_1 = \text{First Order Autocorrelation},\quad E[D_0] = \text{Expected no. of ZCs.}$$

This is perfectly reasonable as when there is high correlation between consecutive samples (i.e. $\rho_1 \rightarrow +1$) a low zero-crossing count results. Conversely, if there is poor correlation between consecutive samples, a high zero-crossing count results. However, as Kedem states in [25] there are certain advantages of each parameter over the other.

3.3 Linear Filters.

A time invariant filter applied to a discrete signal may be mathematically described by convolution:

$$\mathcal{L}(x_t) = \sum_{j=-\infty}^{\infty} h_j x_{t-j}$$ where h_j is the impulse response of the filter. (i.e. The coefficients of the filter).

The following rules hold for all linear filters:

- $(a_1\mathcal{L}_1 + a_2\mathcal{L}_2)(x_t) = a_1\mathcal{L}_1(x_t) + a_2\mathcal{L}_2(x_t)$
- $\mathcal{L}_2\mathcal{L}_1(x_t) = \mathcal{L}_2(\mathcal{L}_1(x_t))$

The latter rule is important and in the next chapter it is demonstrated how this may be exploited to reduce computational expense when applying HOC analysis to a signal.

3.3.1 Backward Shift Operator.

This is a simple term used to describe the action of a filter and is synonymous with the z-operator in the Z-transform. It will be used in this thesis in the description of any filter. It may be defined as:

$$\beta x_t \equiv x_{t-1}$$

with Gain = 1,

and $\phi = e^{-i\omega}$ where ω is Frequency.

Hence,

$$\beta^{-1} x_t = x_{t+1}$$

And the filter family may be described by,

$$\beta^j x_t = x_{t-j}$$ where j is the filter order.

3.3.2 Differencing Operator.

Also referred to as a differencing filter or delta operator, it is defined as:

$$\nabla x_t \equiv (1-\beta)x_t = x_t - x_{t-1}$$

The frequency response of this filter may be described as:

$$|H(\omega)|^2 = 2^j(1-\text{Cos}\omega)^j$$ where j is the filter order.

and has a filter family described by:

$$\nabla^j x_t = (1-\beta)^j x_t$$

The differencing filter acts as a high pass filter and a plot of the squared gain for a first order filter may be seen in Figure 3.6.

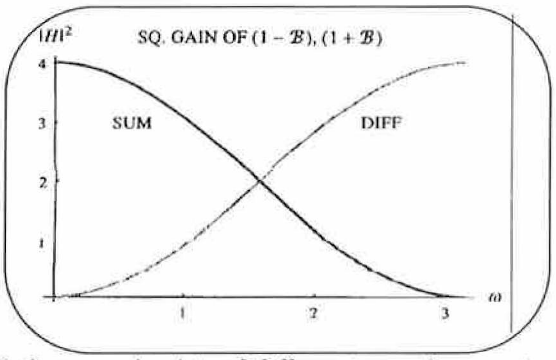


Figure 3.6: Square gain plots of *differencing* and *summation* filters.

3.3.3 Summation Operator.

The summation operator acts as a low pass filter and may be defined as:

$$\check{S}x_t \equiv (1+\beta)x_t = x_t+x_{t-1}$$

The frequency response of this filter may be described as:

$$|H(\omega)|^2 = 2^j(1+\text{Cos}\omega)^j \qquad \text{where j is the filter order.}$$

Its family of filters may be described as:

$$\check{S}^j x_t = (1+\beta)^j x_t$$

A plot of the squared gain of the first order summation filter is also shown in Figure 3.6.

3.3.4 Slutsky Filter.

This is a combination of a differencing and summation filter resulting in a band pass filter. It may be defined as:

$$\mathcal{L}_{m,n}(x_t) \equiv (1-\beta)^{m-1}(1+\beta)^{n-1}$$

Note that j has now become a vector j=(m,n).

3.3.5 Alternative Filters Suitable in HOC Analysis.

- Complex Filter:

$$\mathcal{L}(\beta) \equiv (1+e^{i\theta}\beta)^n \qquad \text{where } \theta\in[0,\pi]$$

n is a positive integer.

The frequency response of this filter may be described as:

$$|H(\omega)|^2 = 4^n \mathrm{Cos}^{2n}([\theta-\omega]/2)$$

Note: when n becomes very large, a band pass filter centred at θ results.

- Recursive Filter:

$$y_t = -a_1 y_{t-1} - \dots - a_m y_{t-m} + b_0 x_t + b_1 x_{t-1} + \dots + b_n x_{t-n}$$

where y_t is present output

x_t is present input.

- AR(1) or α-Filter:

$$y_t \equiv \alpha y_{t-1} + x_t$$

Or, $$y_t = \sum \alpha^j x_{t-j} \qquad |\alpha|<1$$

$$H(\omega) = (1-\alpha e^{-i\omega})^{-1}$$

Note: If $\alpha<0$, a high pass filter results and if $\alpha>0$, a low pass filter results.

3.4 Effects of Linear Filters on Signals.

So, how does the application of a linear filter affect a signal? Firstly, one may say that the filter (whether low, high or band pass) attenuates certain frequency components in the signal. This results in a new signal (perhaps containing a higher or lower dominant frequency) with different zero-crossing information. Further filtering results in further changes to the zero-crossing information. This thesis proposes that with the correct choice of filter, this alteration in zero-crossing information may be used in some manner in order to distinguish between different speech signals.

3.4.1 Effects of the Differencing Filter on Zero-Crossing Information.

As previously mentioned, the differencing filter acts as a high pass filter; the greater the order the more pronounced the filter. Hence, if a signal is repeatedly differenced, a lower band of frequencies is attenuated each time; it is therefore reasonable to suggest that the number of zero-crossings increases after each filter stage. This assumption proves true for any signal and a plot of the number of zero-crossings against the filter order (termed a HOC Plot) such as that in Figure 3.7 results. Adhering to Kedem's convention in [25], from this point on in the thesis, the number of zero-crossings obtained from differencing is referred to as D_j and from summation as ${}_jD$.

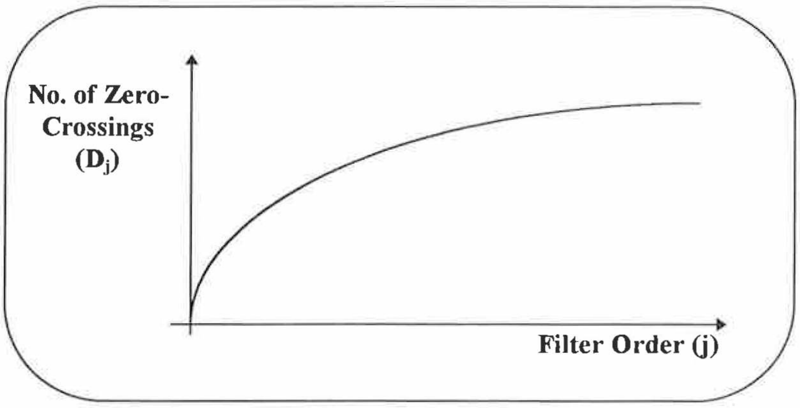


Figure 3.7: Typical HOC plot after repeated *differencing*.

It can be observed from Figure 3.8 that rapidly changing signals (i.e. those of little inter-sample correlation) display significantly different HOC plots to those of smoother signals from a similar process, but as $j \to \infty$, D_j converges to the same value for each signal.

The differencing filter HOC plot for any signal is always a monotonic increasing function and converges to the highest frequency present in the signal (Convergence Rule).

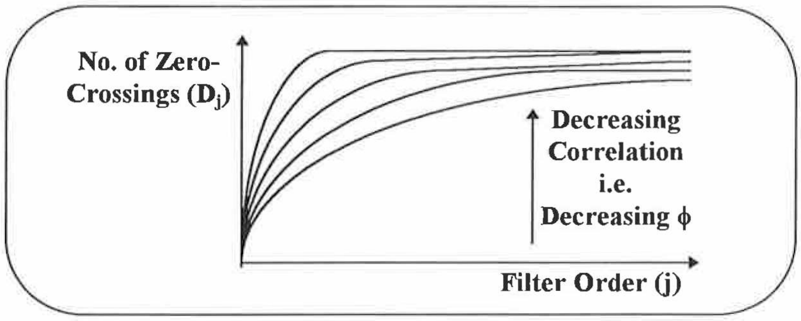


Figure 3.8: HOC plots for signals of varying smoothness.

3.4.2 Effects of the Summation Filter on Zero-Crossing Information.

The operation of the summation filter is opposite to that of the differencing filter, in that it acts as a low pass filter attenuating higher bands of frequencies, thus lowering the number of zero-crossings. The corresponding HOC plot (as seen in Figure 3.9) shows a monotonic decreasing function, converging to the lowest frequency present.

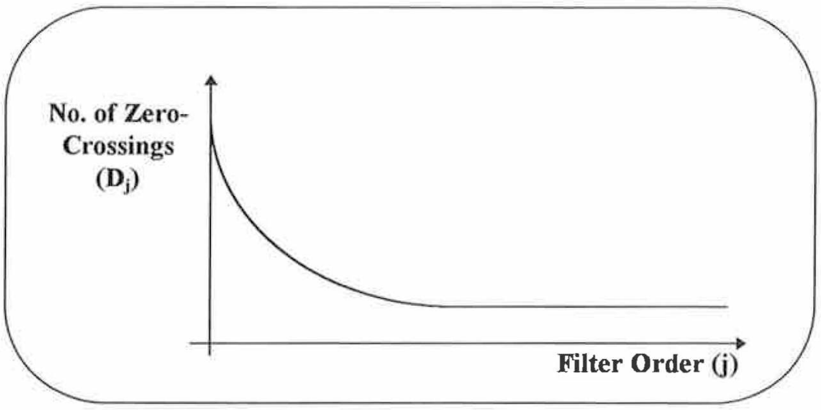


Figure 3.9: Typical HOC plot after repeated *summation*.

3.4.3 Effects of the Slutsky Filter on Zero-Crossing Information.

The two dimensional band pass filter nature of the Slutsky Filter yields a more complex three dimensional HOC plot whose exact shape is dependent on the signal. An example is shown in Figure 3.10.

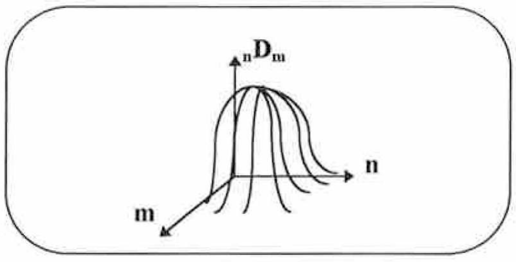


Figure 3.10: Typical HOC plot for a *Slutsky filter*.

Chapter 4

4. Implementation of Higher Order Crossing Analysis.

This chapter explains how HOC analysis was implemented using Matlab and what modifications were made to the original analysis technique described by Kedem [25] in order to improve on performance. The newly designed system based on HOC analysis (termed CHOC) is then introduced with an explanation of how it is better than its predecessor. The results from various tests on the CHOC system are then presented and discussed. Among these are a repeatability test, discrimination powers for whole utterances, vowels and consonants and the systems ability to perform phonemic segmentation.

4.1 Investigation into Suitability of Filters.

Before developing a speech recognition system based on HOC analysis, it is important to find the family of filters most suited to speech signals. It was necessary to simulate each filter, apply them to appropriate signals and evaluate their suitability to speech analysis.

4.1.1 Implementation of Filters.

Initially, the differencing and summation filters were implemented in Matlab using the m-files *Diffil.m* and *Sumfil.m*, respectively, as described by their flowcharts in Figures 4.1 and 4.2. For a more in-depth description of these and their associated m-files refer to Appendix C1.

The delay operator was achieved using *Roll.m* (described in Figure 4.3), which simply rotates the data sequence n positions to the left or right depending on whether a β^{-n} or β^{n} is required. This significance of this m-file is explained in detail in Section 4.2.1.

On testing these m-files it was found that they ran at a totally impractical speed (~50min/word). A re-examination of them discovered that it was not necessary to calculate the filter coefficients each time and that the zero-crossing counter could be incorporated into the filtering m-files. The filter rule mentioned in Section 3.3,

$$\text{i.e. } \mathcal{L}_2\mathcal{L}_1(x_t) = \mathcal{L}_2(\mathcal{L}_1(x_t))$$

allowed the HOC system to be designed as in Figure 4.5(b).

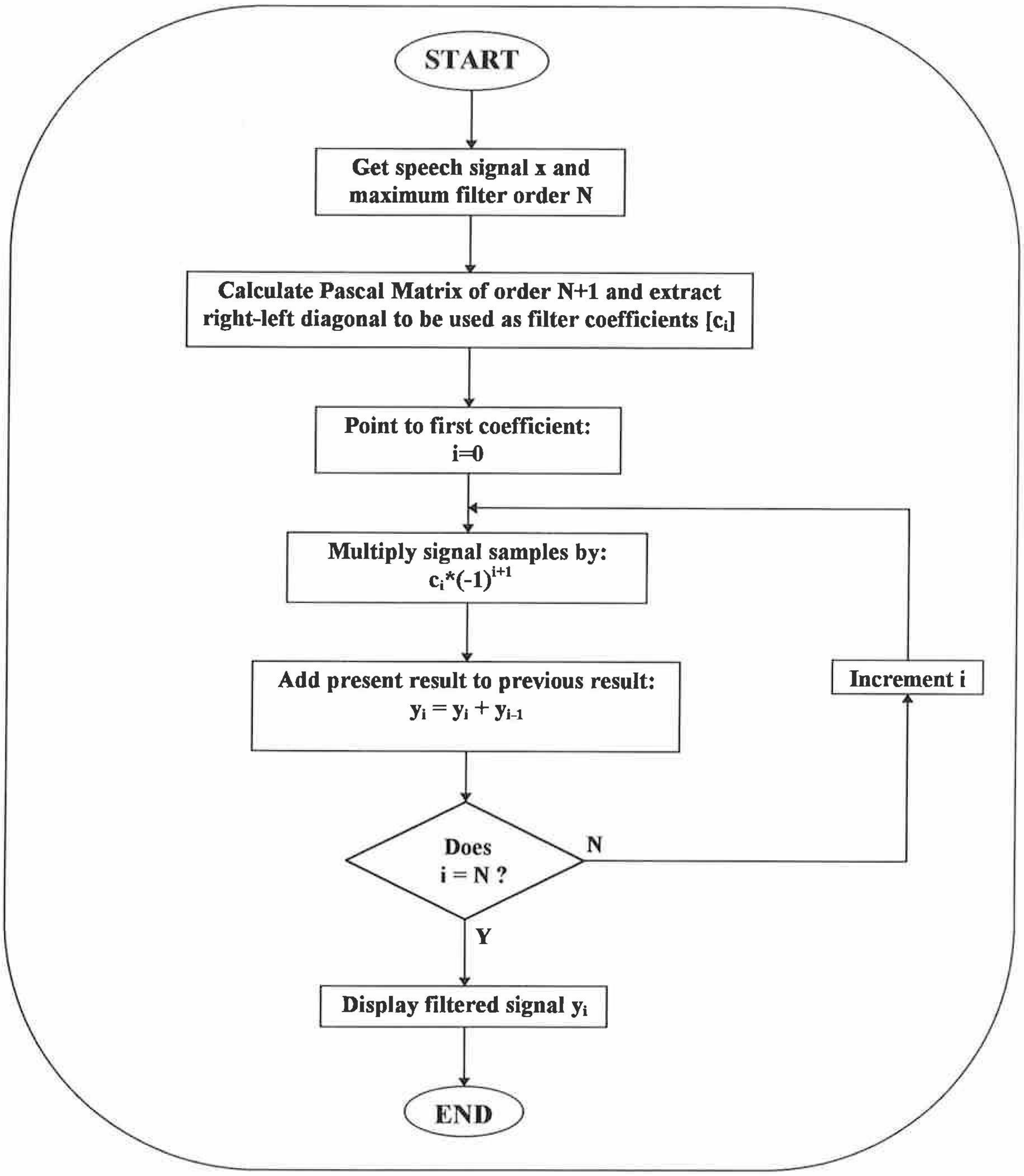

START
Get speech signal x and maximum filter order N
Calculate Pascal Matrix of order N+1 and extract right-left diagonal to be used as filter coefficients [c_i]
Point to first coefficient: i=0
Multiply signal samples by: $c_i*(-1)^{i+1}$
Add present result to previous result: $y_i = y_i + y_{i-1}$
Increment i
Does i = N ?
N
Y
Display filtered signal y_i
END

Figure 4.1: Flowchart describing how Diffil.m filters a speech signal.

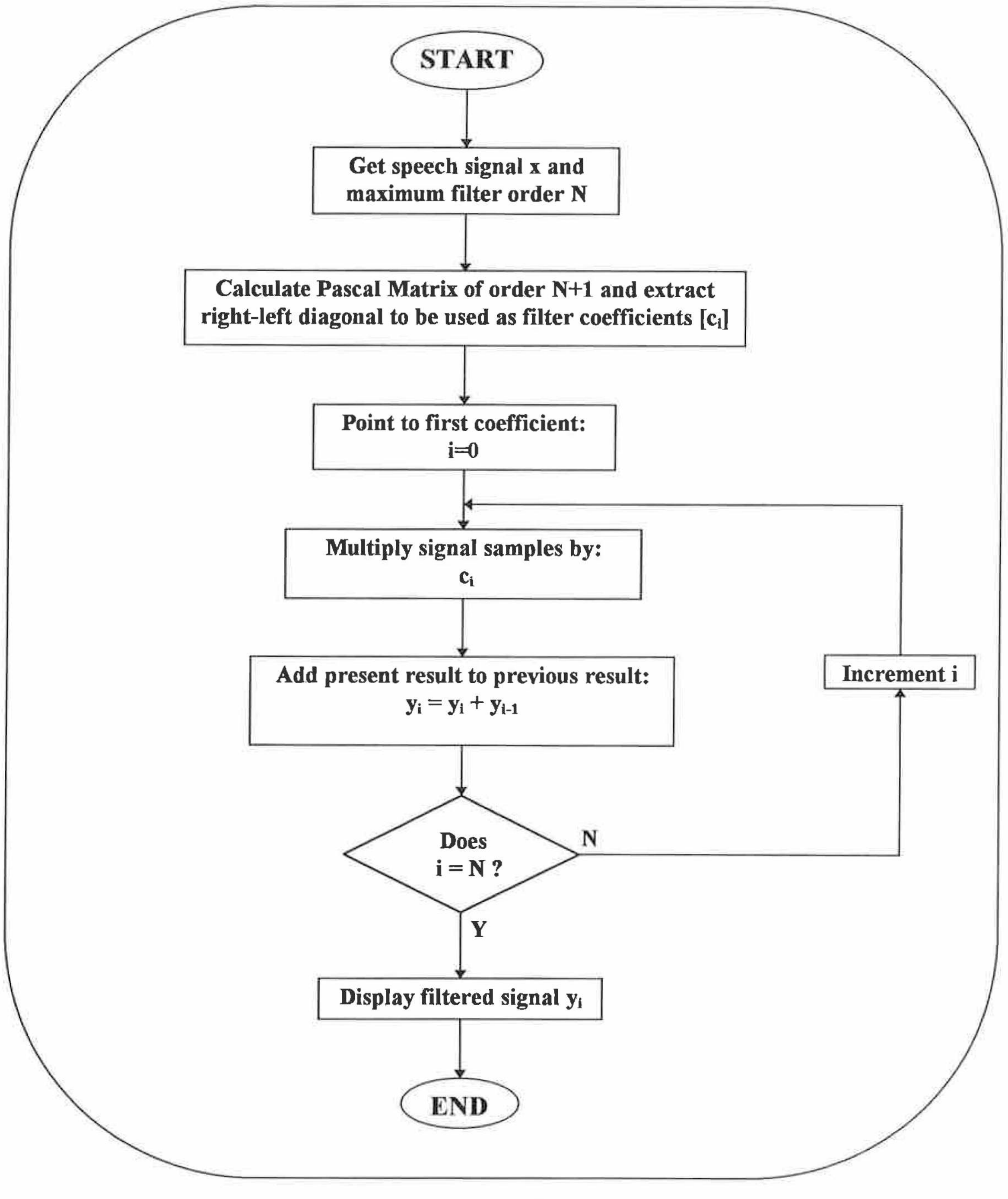

START
Get speech signal x and maximum filter order N
Calculate Pascal Matrix of order N+1 and extract right-left diagonal to be used as filter coefficients [c_i]
Point to first coefficient: i=0
Multiply signal samples by: c_i
Add present result to previous result: y_i = y_i + y_i-1
Increment i
Does i = N ?
N
Y
Display filtered signal y_i
END

Figure 4.2: Flowchart describing the filtering achieved by Sumfil.m.

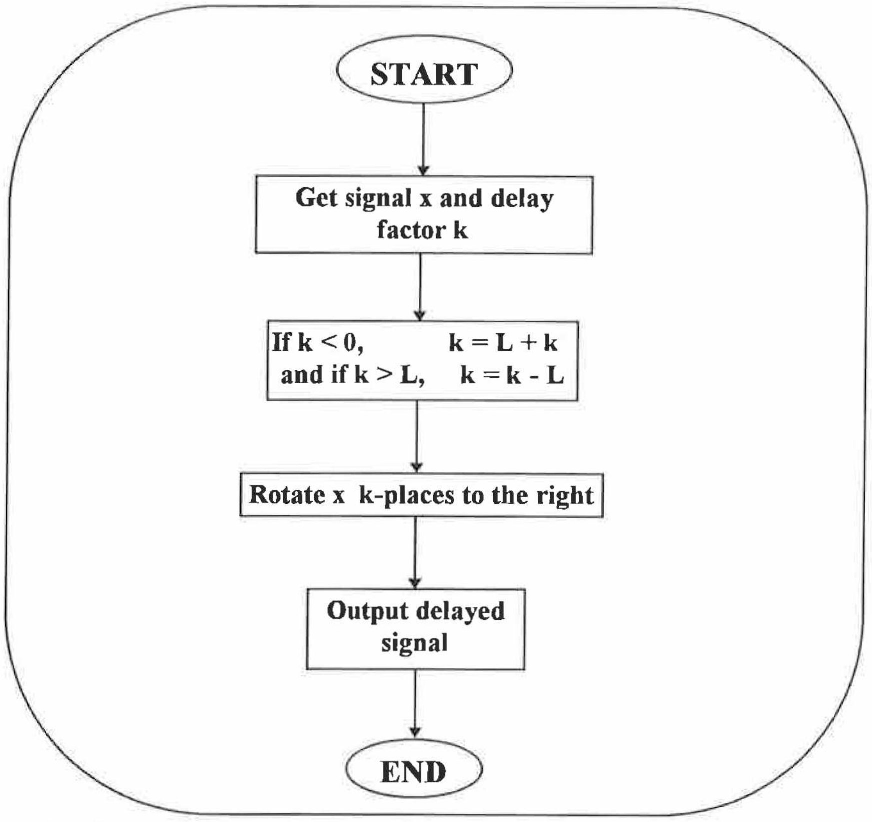
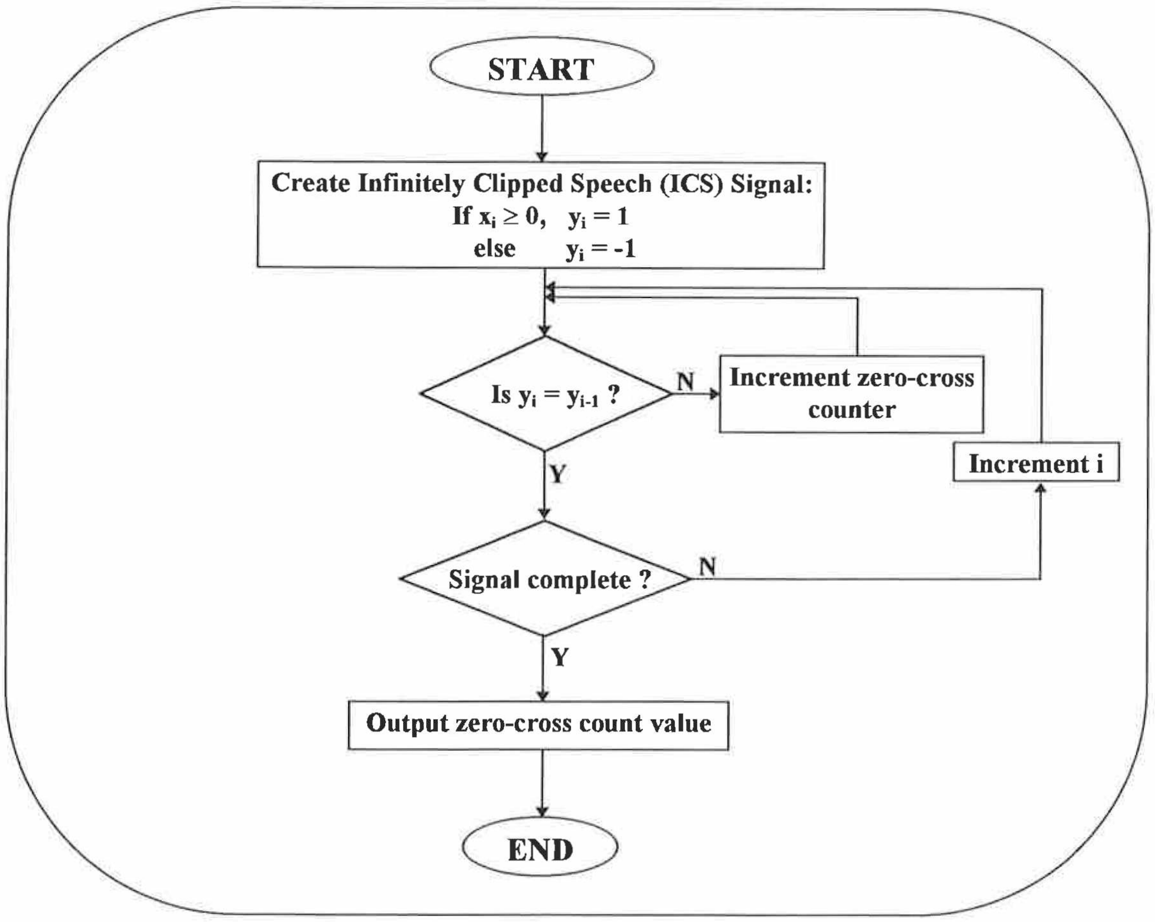


Figure 4.3: Flowchart expaining how Roll.m creates a delayed version of a speech signal.



START
Create Infinitely Clipped Speech (ICS) Signal:
If $x_i \geq 0$, $y_i = 1$
else $y_i = -1$
Is $y_i = y_{i-1}$?
N
Increment zero-cross counter
Increment i
Y
Signal complete ?
N
Y
Output zero-cross count value
END

Figure 4.4: Flowchart showing how Zcount.m counts the number of zero-crossings in a speech signal.

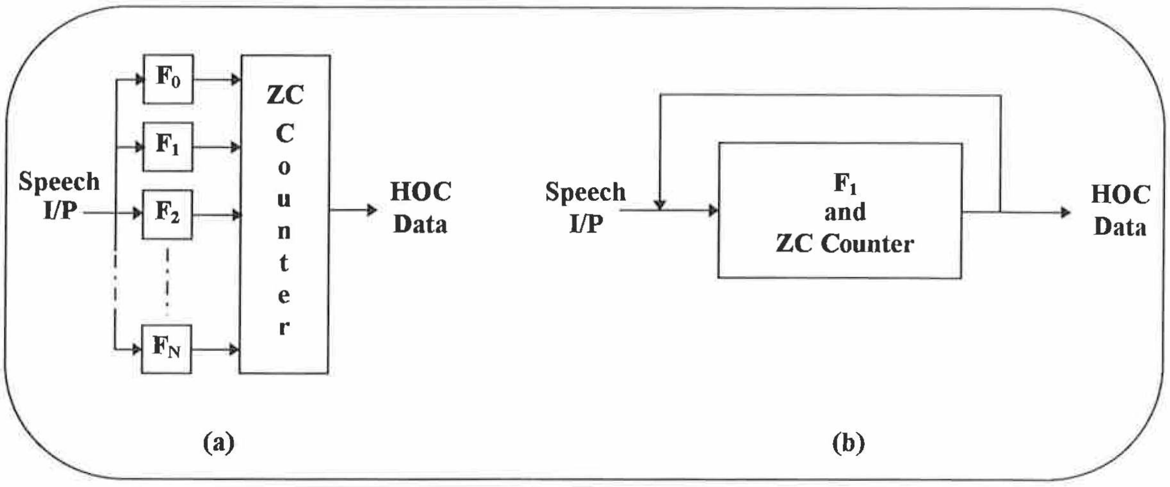


Figure 4.5: Extraction of HOC data using (a) *Diffil.m/Sumfil.m/Zcount.m* and (b) *Hocdif.m/Hocsum.m*

Hence, *Hocdif.m* and *Hocsum.m* were developed and executed much faster than their predecessors (~5min/word). These functions are described by the block diagrams in Figure 4.5 and the flowcharts in Figures 4.6 and 4.7.

Also developed was an m-file to simulate a Slutsky filter called *Hocssky.m*. For the purposes of the work described in this thesis, it was sufficient to let m=n at all times, thus avoiding problems presented by two-dimensional arrays.

4.1.2 Testing the Filters.

4.1.2.1 Initial Tests.

An initial test was performed on each filtering technique using the data sequence below (first used by Kedem [25]) to ensure that the m-files were executing correctly.

$$x = [-3.4, 1.6, -3.4, 2.6, 3.6, 4.6, -2.4, -1.4, -4.4, 2.6]$$

The following results were obtained from *Hocdif.m*:

$$x = [-3.4, 1.6, -3.4, 2.6, 3.6, 4.6, -2.4, -1.4, -4.4, 2.6]$$

$$\nabla x = [-6, 5, -5, 6, 1, 1, -7, 1, -3, 7]$$

$$\nabla^2 x = [-13, 11, -10, 11, -5, 0, -8, 8, -4, 10]$$

Yielding: $D_0 = 5,\ D_1 = 7,\ D_2 = 7 \Rightarrow$ Monotonic Increasing and Converging!

(Note: These results do not exactly agree with those obtained in [25] as a different technique was employed to deal with end effects - Refer to Section 4.2.1).

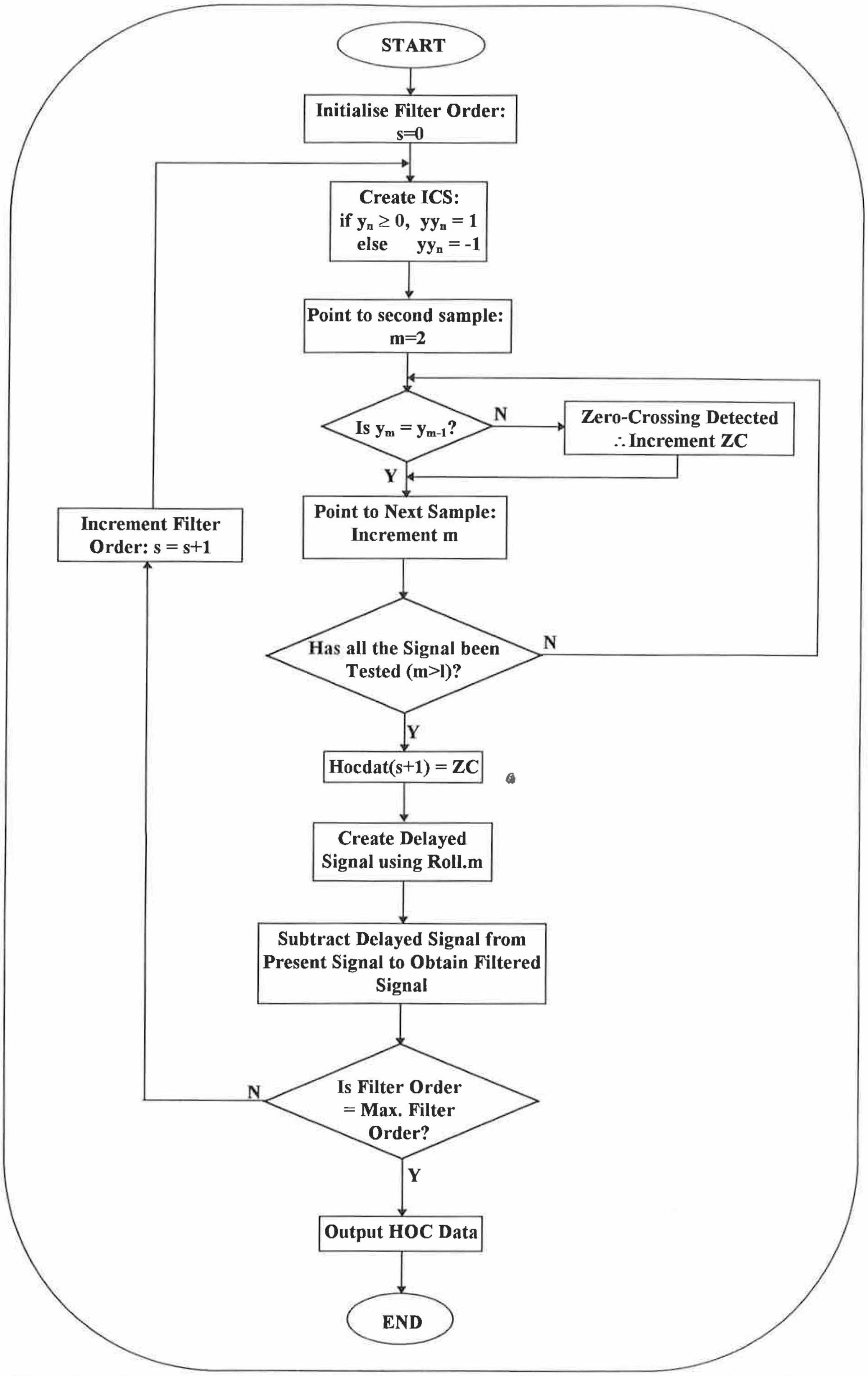

START
Initialise Filter Order: s=0
Create ICS: if $y_n \geq 0$, $yy_n = 1$ else $yy_n = -1$
Point to second sample: m=2
Is $y_m = y_{m-1}$?
N
Zero-Crossing Detected ∴ Increment ZC
Y
Point to Next Sample: Increment m
Increment Filter Order: s = s+1
Has all the Signal been Tested (m>l)?
N
Y
Hocdat(s+1) = ZC
Create Delayed Signal using Roll.m
Subtract Delayed Signal from Present Signal to Obtain Filtered Signal
N
Is Filter Order = Max. Filter Order?
Y
Output HOC Data
END

Figure 4.6: Flowchart describing the more efficient method of differencing a speech signal: Hocdif.m.

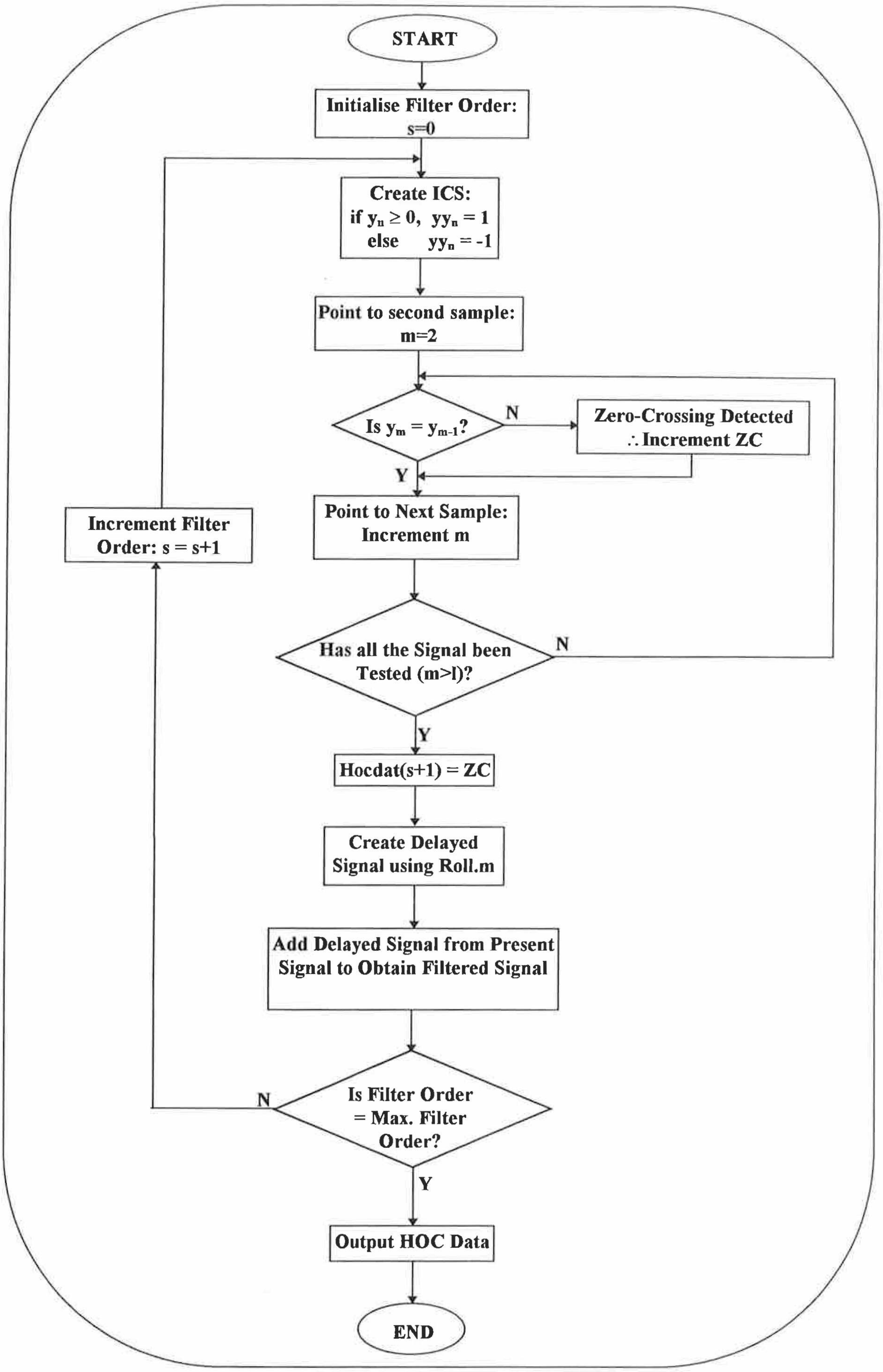

START
Initialise Filter Order: s=0
Create ICS: if $y_n \geq 0$, $yy_n = 1$ else $yy_n = -1$
Point to second sample: m=2
Is $y_m = y_{m-1}$?
N
Zero-Crossing Detected ∴ Increment ZC
Y
Point to Next Sample: Increment m
Increment Filter Order: s = s+1
Has all the Signal been Tested (m>l)?
N
Y
Hocdat(s+1) = ZC
Create Delayed Signal using Roll.m
Add Delayed Signal from Present Signal to Obtain Filtered Signal
N
Is Filter Order = Max. Filter Order?
Y
Output HOC Data
END

Figure 4.7: Flowchart showing the more efficient summation filtering technique: Hocsum.m.

The following results were obtained from *Hocsum.m*:

$$x = [-3.4, 1.6, -3.4, 2.6, 3.6, 4.6, -2.4, -1.4, -4.4, 2.6]$$

$$\check{S}x = [-.8, -1.8, -1.8, -.8, 6.2, 8.2, 2.2, -3.8, -5.8, -1.8]$$

$$\check{S}^2x = [-2.6, -2.6, -3.6, -2.6, 5.4, 14.4, 10.4, -1.6, -9.6, -7.6]$$

Yielding: $_0D = 5,\ _1D = 2,\ _2D = 2 \Rightarrow$ Monotonic Decreasing and Converging!

The above results proved that the m-files were operating as desired and were ready for testing on real signals.

4.1.2.2 Tests on Real Signals.

The differencing, summation and Slutsky filters were executed up to the 10th order on five utterances of each word in the Gr I vocabulary set (as described in Appendix A4). The HOC sequences in Tables 4.1 to 4.3 were obtained, yielding the HOC plots in Figures 4.8 to 4.10.

The HOC plots in Figure 4.8, corresponding to the differencing filter, display reasonable signal discrimination between the different words, with only a slight overlap between Right3 & Rev4 and Right5 & Left5. These clearly defined areas on the HOC plane for each word are a direct result of the different correlation functions and different degrees of higher frequency content. Hence, Go (consisting mainly lower frequencies and closely correlated samples) assumes a position in a lower portion of the HOC plane.

A set of HOC ranges, as in Table 4.4, may now be established for use as a template system for comparison to the HOC data of an unknown test signal. That is, if *all* the test signal's D_j values fall into a particular range, the signal may be recognised as that word.

The HOC data in Table 4.2 was obtained from Hocsum.m and yielded the series of HOC plots in Figure 4.9. This filtering technique is therefore unsuitable due to the unacceptable amount of overlap.

The Slutsky filter was then tested, with n=m to avoid the complications of three dimensional arrays of data. The resulting HOC data in Table 4.3 and HOC plots in Figure 4.10 suggest that this filter is slightly better than the differencing filter at discriminating between signals.

Word/ Fil.Ord	Go 1	Go 2	Go 3	Go 4	Go 5	Stop 1	Stop 2	Stop 3	Stop 4	Stop 5	Rev 1	Rev 2	Rev 3	Rev 4	Rev 5	Left 1	Left 2	Left 3	Left 4	Left 5	Right 1	Right 2	Right 3	Right 4	Right 5
0	231	254	218	219	212	310	217	203	258	293	343	269	288	316	276	414	356	411	353	299	317	304	264	310	380
1	607	622	607	586	642	951	1158	1043	1136	1212	1210	1157	1180	1280	1362	2050	1820	2003	1996	1745	1804	1686	1592	1768	1952
2	1086	1219	1171	944	1077	1782	1936	1723	1844	1954	2100	1993	1996	2179	2275	3328	3318	3215	3368	3129	2810	2844	2510	2838	3240
3	1492	1827	1643	1357	1519	2322	2406	2181	2194	2434	2507	2405	2362	2604	2644	3901	3806	3691	3820	3609	3336	3298	2902	3276	3629
4	1973	2273	1985	1735	1915	2628	2713	2487	2415	2717	2747	2641	2596	2882	2836	4195	4042	3949	4042	3857	3576	3566	3092	3508	3828
5	2267	2433	2203	1919	2067	2786	2909	2653	2583	2847	2889	2811	2783	3067	3028	4403	4148	4093	4187	4007	3697	3678	3214	3622	3977
6	2399	2495	2323	2053	2157	2915	3046	2755	2715	2917	3045	2945	2983	3231	3170	4531	4248	4207	4275	4101	3784	3772	3296	3708	4083
7	2515	2557	2389	2135	2213	2973	3136	2805	2783	2975	3179	3077	3111	3355	3266	4621	4333	4261	4337	4153	3858	3846	3362	3778	4163
8	2587	2618	2437	2211	2261	3031	3184	2848	2819	3017	3343	3161	3207	3449	3362	4689	4383	4313	4393	4209	3901	3910	3408	3822	4215
9	2629	2660	2475	2262	2293	3059	3237	2886	2852	3063	3407	3245	3247	3547	3406	4731	4425	4368	4459	4255	3953	3952	3474	3882	4273
10	2665	2699	2505	2292	2335	3081	3265	2928	2890	3083	3451	3267	3284	3596	3474	4773	4469	4398	4493	4293	3993	3989	3522	3920	4293

Table 4.1: Results after HOC analysis of Gr I vocabulary set using a *differencing* filter.

Word/ Fil.Ord	Go 1	Go 2	Go 3	Go 4	Go 5	Stop 1	Stop 2	Stop 3	Stop 4	Stop 5	Rev 1	Rev 2	Rev 3	Rev 4	Rev 5	Left 1	Left 2	Left 3	Left 4	Left 5	Right 1	Right 2	Right 3	Right 4	Right 5
0	231	254	218	219	212	310	217	203	258	293	343	269	288	288	254	414	356	411	353	299	317	304	264	310	380
1	222	242	200	218	198	282	198	178	222	258	302	231	260	264	232	344	310	360	298	261	267	268	238	286	312
2	218	234	192	214	194	258	184	172	208	246	286	219	252	246	208	308	294	334	270	253	254	260	222	266	280
3	218	226	188	210	192	248	182	158	196	230	260	215	242	244	204	302	284	318	262	251	248	246	212	256	262
4	214	224	182	208	182	238	178	148	184	220	254	209	236	238	198	284	276	298	250	247	240	238	204	256	258
5	212	222	180	204	178	234	176	142	180	212	252	205	230	230	194	272	269	286	246	241	232	232	198	250	256
6	210	222	174	202	176	230	174	138	178	204	244	203	222	228	186	262	267	276	242	237	230	232	190	244	244
7	208	222	172	202	174	227	174	138	172	200	244	199	218	226	184	256	260	264	240	235	224	230	184	244	234
8	206	222	172	200	174	221	172	136	172	198	240	197	212	226	176	248	256	260	238	233	222	226	180	242	234
9	206	218	170	197	172	216	170	134	166	196	238	197	210	224	176	246	256	256	236	231	220	225	178	242	230
10	200	216	168	193	170	212	168	134	164	192	232	193	206	220	176	238	250	250	230	229	220	223	176	238	230

Table 4.2: Results after HOC analysis of Gr I vocabulary set using a *summation* filter.

Word/ Fil.Ord	**Go 1**	**Go 2**	**Go 3**	**Go 4**	**Go 5**	**Stop 1**	**Stop 2**	**Stop 3**	**Stop 4**	**Stop 5**	**Rev 1**	**Rev 2**	**Rev 3**	**Rev 4**	**Rev 5**	**Left 1**	**Left 2**	**Left 3**	**Left 4**	**Left 5**	**Right 1**	**Right 2**	**Right 3**	**Right 4**	**Right 5**
0	231	254	218	219	212	310	217	203	258	293	343	269	288	316	276	414	356	411	353	299	317	304	264	310	380
1	531	520	494	518	530	751	864	776	832	908	968	889	865	986	1046	1478	1322	1382	1368	1263	1295	1212	1120	1196	1316
2	865	842	802	748	779	1180	1298	1117	1172	1345	1568	1395	1390	1574	1588	2206	2042	2058	2141	1951	1897	1898	1688	1904	2161
3	996	1001	954	848	889	1408	1514	1339	1392	1519	1796	1651	1655	1797	1850	2507	2358	2378	2483	2289	2081	2150	1906	2116	2403
4	1066	1115	1066	924	964	1534	1644	1471	1514	1609	1906	1777	1714	1869	1908	2635	2508	2462	2571	2427	2166	2274	1994	2220	2497
5	1124	1207	1146	990	1086	1599	1706	1541	1540	1691	1958	1833	1766	1945	1956	2707	2556	2516	2630	2491	2250	2334	2032	2255	2539
6	1220	1335	1224	1118	1143	1709	1754	1565	1594	1735	2023	1853	1830	1991	1976	2751	2598	2542	2646	2525	2302	2363	2082	2307	2569
7	1318	1453	1320	1216	1223	1752	1790	1611	1614	1790	2033	1890	1859	2021	2002	2773	2623	2580	2672	2547	2342	2397	2094	2339	2599
8	1391	1591	1372	1317	1299	1793	1808	1627	1642	1810	2058	1900	1869	2051	2046	2805	2637	2586	2673	2561	2394	2403	2138	2359	2621
9	1491	1683	1454	1401	1383	1823	1820	1663	1649	1828	2088	1909	1893	2062	2054	2831	2651	2623	2677	2584	2415	2421	2142	2389	2639
10	1565	1745	1507	1447	1443	1853	1854	1689	1693	1860	2090	1923	1901	2088	2074	2859	2657	2617	2691	2592	2441	2417	2160	2395	2651

Table 4.3: Results after HOC analysis of Gr I vocabulary set using a *Slutsky* filter with m = n.

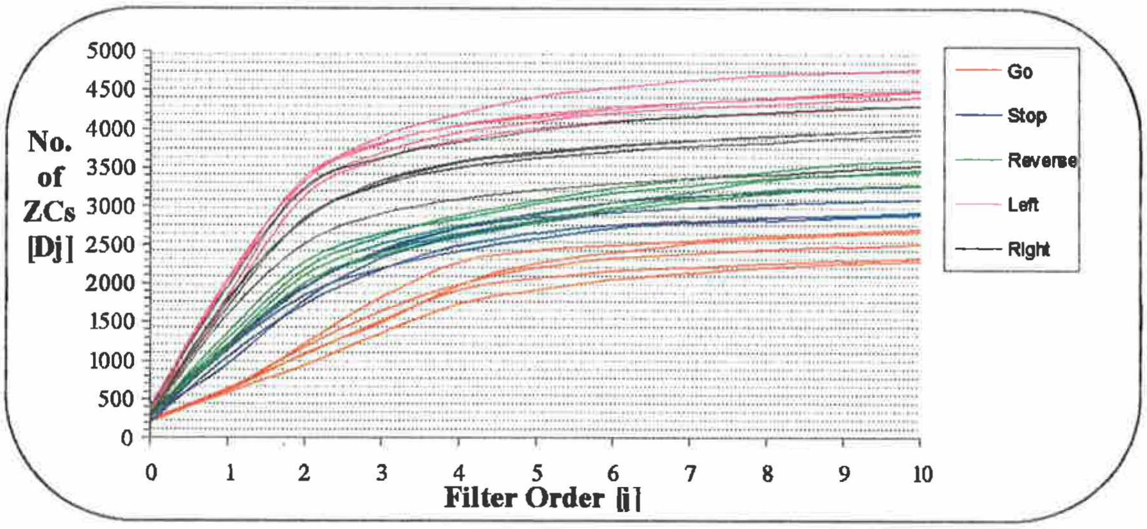


Figure 4.8: HOC plots for Gr I vocabulary set using a differencing filter.

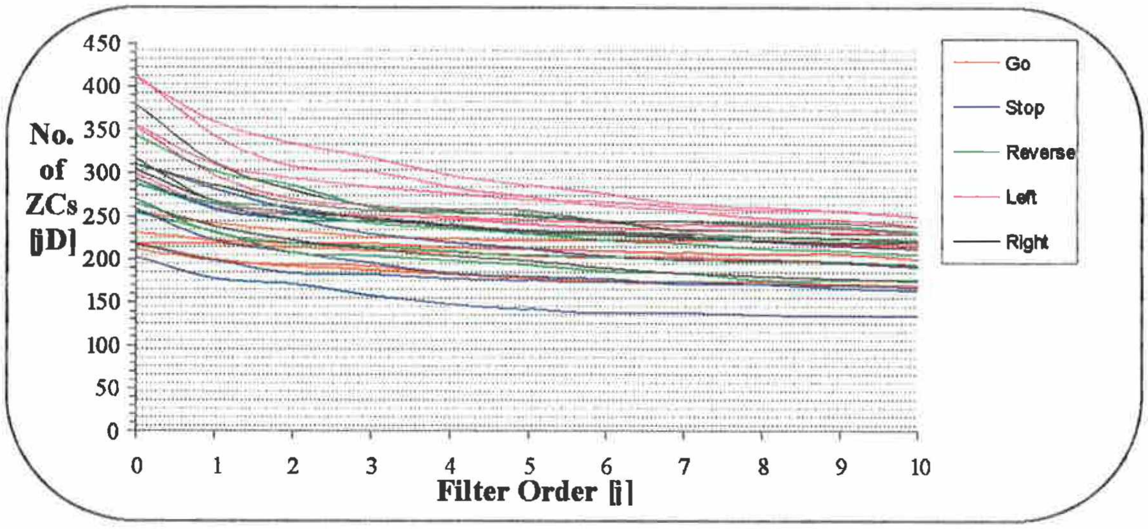


Figure 4.9: HOC plots for Gr I vocabulary set using a summation filter.

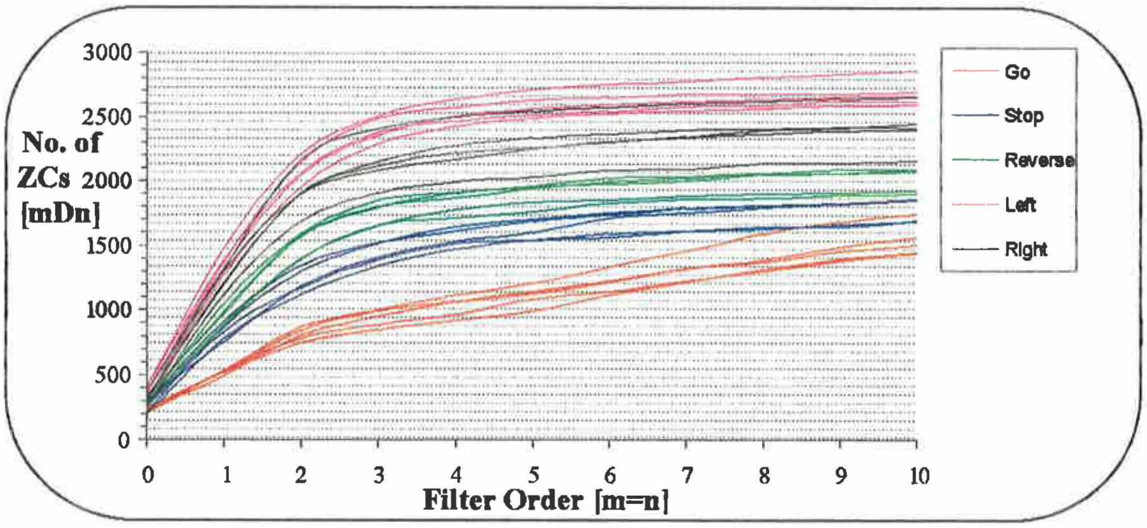


Figure 4.10: HOC plots for Gr I vocabulary set using a Slutsky filter.

Go											
Filter Order	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ZCmin	200	565	925	1335	1715	1910	2035	2115	2190	2240	2270
ZCmax	275	660	1240	1845	2295	2455	2515	2575	2640	2680	2720
Stop											
Filter Order	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ZCmin	185	930	1700	2160	2395	2565	2695	2765	2800	2830	2870
ZCmax	330	1180	1975	2455	2735	2930	3065	3155	3205	3260	3285
Reverse											
Filter Order	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ZCmin	250	1135	1970	2340	2575	2765	2925	3060	3140	3225	3245
ZCmax	365	1380	2295	2665	2900	3090	3250	3375	3470	3570	3615
Left											
Filter Order	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ZCmin	280	1725	3110	3590	3835	3990	4080	4135	4190	4235	4275
ZCmax	430	2070	3390	3920	4215	4420	4550	4640	4710	4710	4795
Right											
Filter Order	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
ZCmin	245	1570	2490	2880	3070	3095	3275	3340	3390	3455	3500
ZCmax	340	1825	2865	3355	3595	3715	3805	3880	3930	3975	4010

Table 4.4: HOC Ranges for *differencing* filter.

The data in Table 4.3 was normalised to 100ms in an attempt to improve discrimination. This resulted in improved grouping between identical words, but deteriorated separation between different words.

Owing to the extra computational expense associated with the Slutsky filter, it was decided that the differencing filter was the most efficient.

However, as one can imagine, as the size of vocabulary increases the amount of HOC plane occupied will also increase, resulting in a larger number of overlapping words. Thus, the limited nature of this technique would prove problematic and so, an alternative system was necessary.

4.2 Solving the Problems of Limited HOC plane.

It was decided that the problems associated with limited HOC plane and overlapping was a result of the poor approach taken in [25] to extracting HOC data from a signal. It is of course unreasonable to simply take a whole utterance and determine one set of HOC data corresponding to this signal. To do this results in a total loss of vital timing information: HOC information is constantly changing throughout the course of each utterance. To ignore the changing nature of this parameter and 'level out' or 'average' the HOC data to one sequence, would only mean discarding valuable signal information which would most certainly help in signal discrimination.

Hence, a new approach had to be taken to ensure as much timing information as possible is retrieved form the signal. The algorithm (described by Figures 4.11 and 4.12) called *Hocalg.m* was developed to solve this problem.

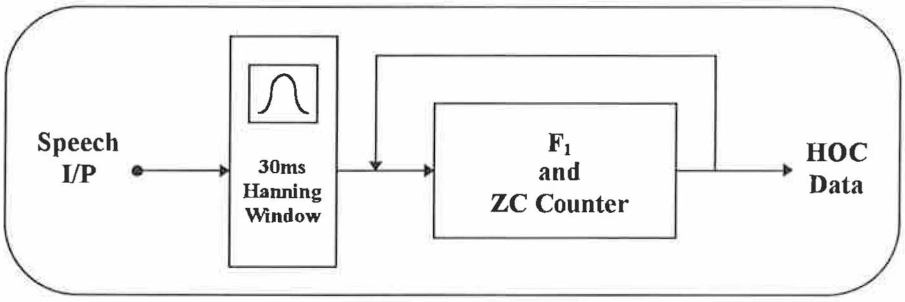


Figure 4.11: Block diagram of *Hocalg.m.*

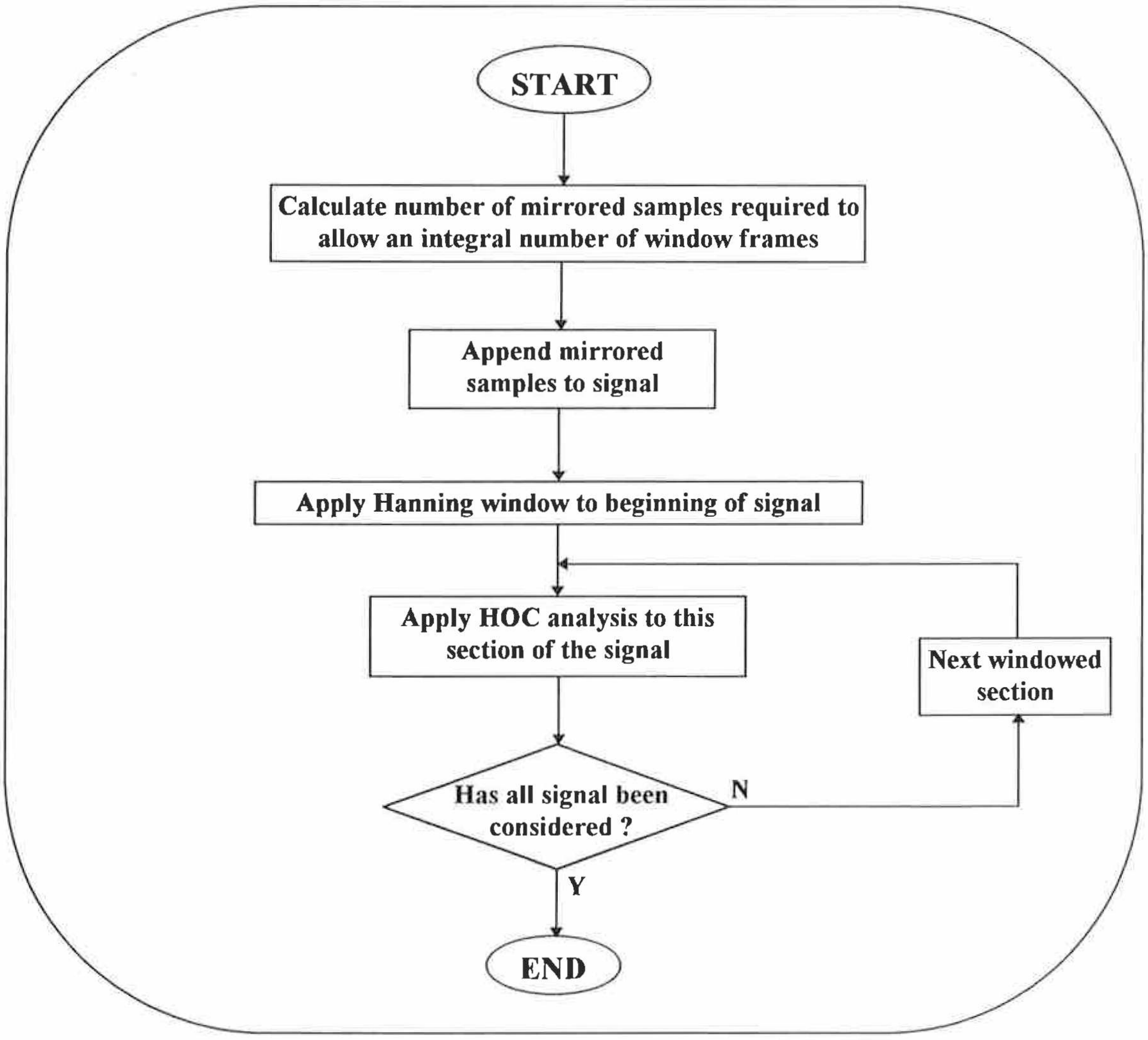

START
Calculate number of mirrored samples required to allow an integral number of window frames
Append mirrored samples to signal
Apply Hanning window to beginning of signal
Apply HOC analysis to this section of the signal
Next windowed section
Has all signal been considered ?
N
Y
END

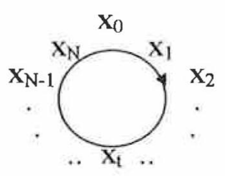
Figure 4.12: Flowchart describing the operation of *Hocalg.m.*

4.2.1 Loss of Data due to Treatment of End Effects.

End effects introduce problems when differencing as the t-1 term does not exist in a causal finite data series[†]. Hence, x_t-x_{t-1} @ t=0 (i.e. x_{-1}) is indeterminable. For each kth order of differencing k data values are unknown (i.e. x_{-1} to x_{-k}). In [25], Kedem implements a *shifted index*[‡] technique to overcome the problem of indeterminable data values. This method, however, results in a loss of data because for a kth order filter only the data x_{k+1} to x_N are considered; the data x_0 to x_k are totally ignored. These samples may, however, be vital to the signals description, especially in the case of a speech signal (e.g. a plosive of short duration such as /t/). Hence, a different technique had to be implemented in controlling end effects. *Roll.m* incorporates a circular buffering procedure to reduce end effects. Now,

$$x_{-1} = x_N, \quad x_{-2} = x_{N-1}, \quad \ldots \quad x_{-k} = x_{N-k+1}$$

i.e.



This allows the data to be processed without the loss of samples.

4.2.2 Necessity of Hanning Window.

The use of a circular buffer to avoid loss of data is not without problems. There is no guarantee that the samples x_N and x_0 are equal or even remotely similar (refer to Figure 4.13).

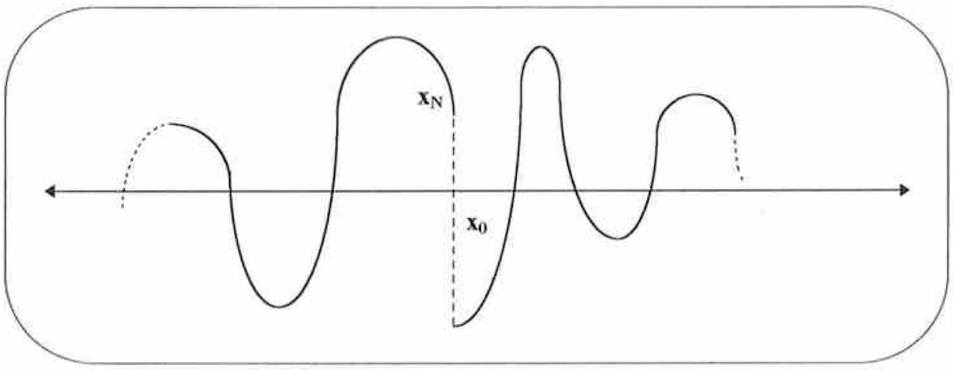


Figure 4.13: An example of the discontinuity between x_N and x_0.

[†] Usually these 'lost' samples are let equal zero (i.e. the signal is zero-padded).
[‡] The kth sample is let equal the first valid sample (where k is the number of samples delayed).

Therefore, if a rectangular window is used to extract the data for the circular buffer, a 'click' may be generated in traversing from x_N to x_0. This click results from the discontinuity in the signal and consists a broad band of frequencies as shown in Figure 4.14. This click affects the spectrum of the signal by introducing numerous sidelobes as can be observed in Figure 4.15.

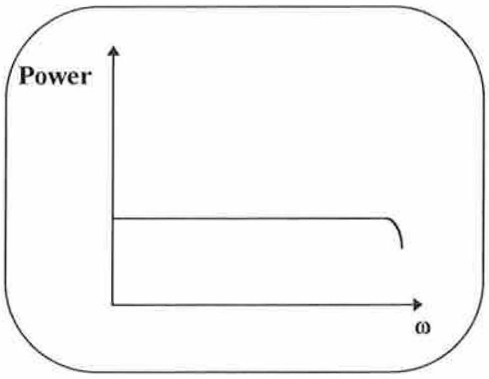


Figure 4.14: Spectrum of a 'click'.

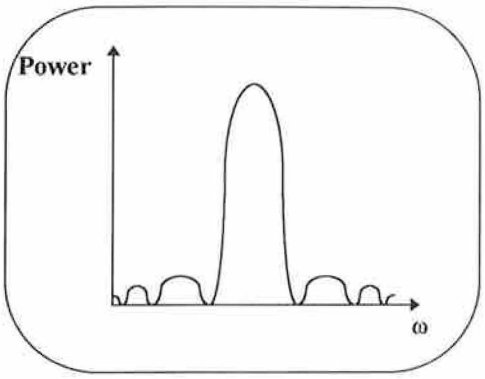


Figure 4.15: Sidelobes due to a click.

Usually an integral number of cycles is selected for analysis to eliminate this problem. Unfortunately, it is extremely difficult to extract an integral number of cycles from a voiced speech signal without knowing its fundamental frequency; it is virtually impossible to do so for unvoiced speech as it is generally not cyclic.

Hence, a Hanning window is incorporated to smooth out any differences between x_N and x_0, thus dampening the effects of the click and attenuating the spectral sidelobes. However, it must be noted that windowing, although reducing *spectral leakage*, also introduces an amount of *smearing* (i.e. the amplitudes of the signal section are slightly distorted). To overcome this problem a window as described in Figure 4.16 may be implemented.

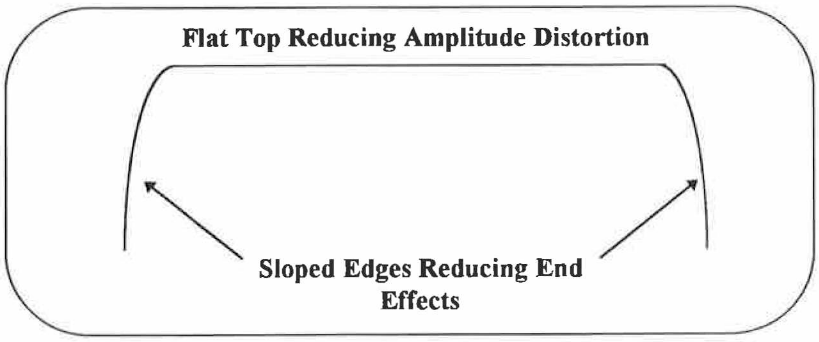

Flat Top Reducing Amplitude Distortion
Sloped Edges Reducing End Effects

Figure 4.16: Possible window to reduce both spectral leakage and smearing.

However, after comparing the spectrum of the multi-formant signal /æ/ to that of a Hanning windowed version, it was observed that all the frequency components remained present and only the overall signal energy was affected (as can be seen in Figure 4.17). As HOC analysis is based on frequency detection it would, therefore, be virtually unaffected by windowing.

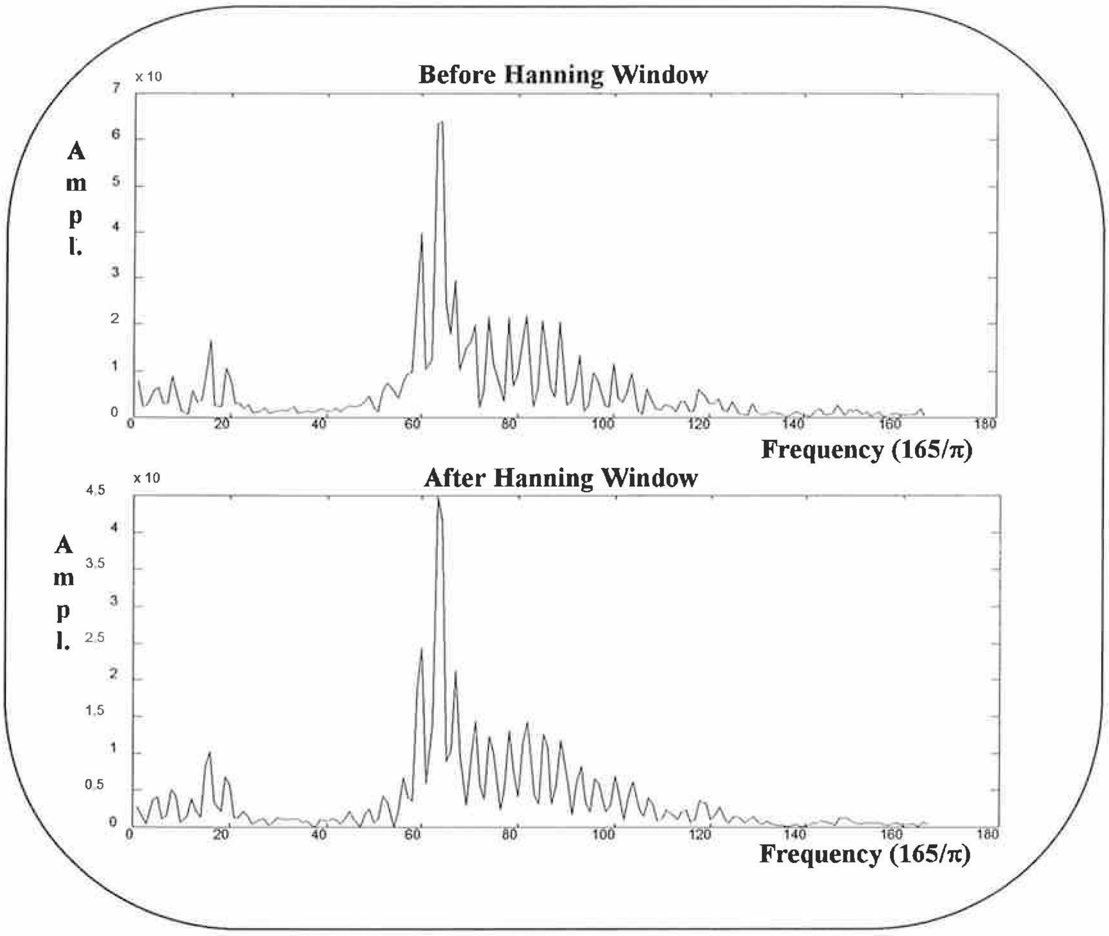


Figure 4.17: Spectra of the phoneme /æ/ before and after the application of a Hanning window.

4.2.3 Necessity to Pad Windowed Section.

As mentioned before, the speech signal is to be analysed section by section with a 50% overlapping window. It would, therefore, simplify matters if an integral number of window frames were present in the speech signal (i.e. the signal can be divided evenly by half the window length). This may be achieved by appending an appropriate number of samples to the end of the signal. The number of appended samples (S_A) may be calculated as follows:

$S_A = (W/2) - R$ where W is the Window Length and R is the Remainder Factor.

And, $R = \text{Rem}\,[N / (W/2)]$ where N is the Total Number of Samples in the Signal.

Initially, zeros were considered for the appended samples however, it seemed more practical to append samples that were in some way related to the signal itself. Hence, it was decided to append the last S_A samples of the signal onto itself (i.e. x_{N-S_A} to x_N). However, rather than attaching this set of samples directly to the end of the signal, a *mirrored image* of the samples was appended. This ensured that x_N and x_{N+1} were exactly the same, thus removing the possibility of discontinuity between the original signal and the appended samples and so safeguarding against the introduction of a click signal.

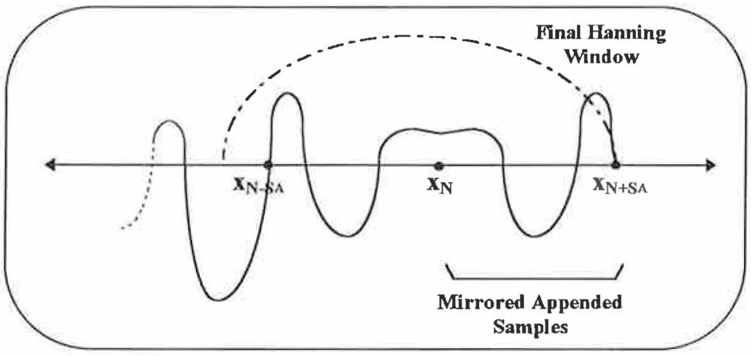


Figure 4.18: Appending mirrored samples to end of signal to allow integral number of windowed sections.

4.3 Testing the New HOC Algorithm.

The new HOC algorithm was tested on a variety of phonemes and the resulting HOC plots (Figure 4.19) were examined. At first sight, these plot seem conflicting with HOC theory. The convergence of the /δ/ sound (as in rod) to the same highest frequency as that of the /s/ sound does not seem to make sense. A low frequency signal should assume a position in the lower HOC plane and one of higher frequency in the upper HOC plane. Should they not converge to the maximum frequency present? Yes. So, why do they all seem to approach the same frequency? Well, this theory operates without problems for signals containing discrete frequencies. However, if the spectrum of a signal is examined (say for the /əʊ/ sound, as in low), it may be seen that although the frequency components above 1.1 kHz are of so little energy that they would normally be rendered negligible, *yet* they are still present. In a real continuous frequency signal the highest frequency present is always π, no matter how weak this component. As explained in [25], the HOC system tends to land on and detect discrete frequencies, but this system fails in the case of a continuous frequency signal. Why? Well, when the real signals (i.e. the sounds) were recorded there was always going to be white noise present in the recording environment. The best a low pass noise filter or the anti-aliasing filter could do was to limit this noise to pink (or band limited white noise). The highest frequency in this noise signal is the cut-off frequency of the anti-aliasing filter - i.e. π. Hence, as $j \to \infty$, $D_j \to \pi$. However, it can be seen that the peak amplitudes of each phoneme after each differencing stage are very different suggesting the different concentrations of energy at different frequencies (refer to Table 4.5). Kedem in [25] developed a new algorithm (the HK Algorithm) to overcome this problem. This algorithm would prove to be unsuitable for a speech signal owing to its extreme slowness and extraction of the same information of that from a FFT. Hence, it was decided that, rather than determine the highest frequency, remove it and so on, as is done in the HK algorithm, it would simplify matters greatly if it was possible to employ only the lower few orders of HOC data as speech recognition parameters.

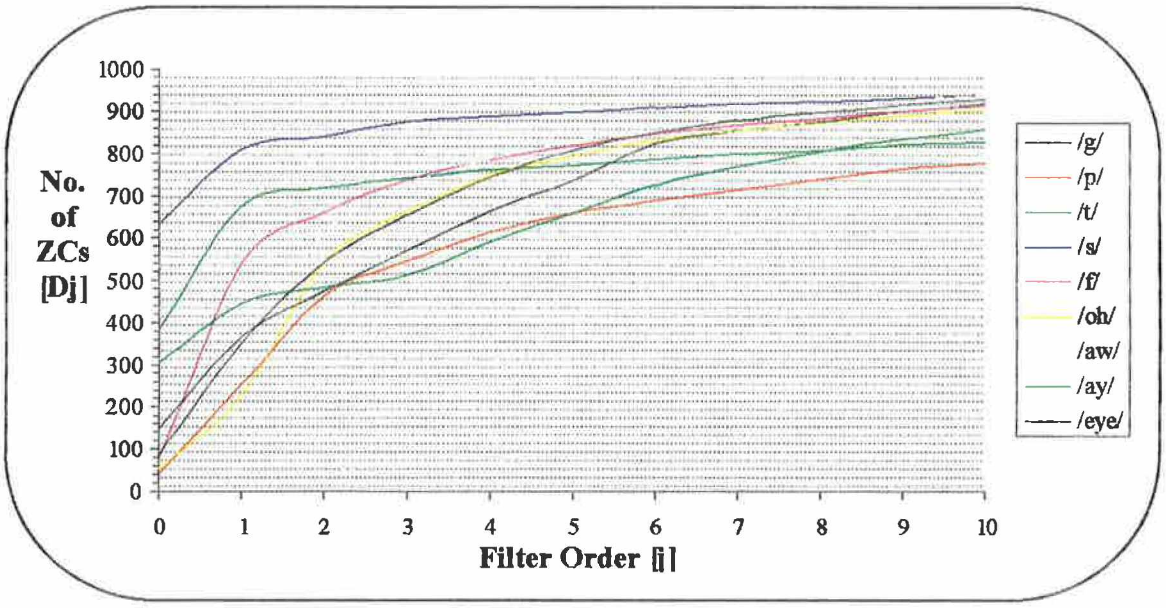


Figure 4.19: HOC plots for various phonemes converging to the same highest frequency. The signals have been normalised to 100msec.

Phoneme	Filter Order	Amplitude
/s/	0	3000
	5	1600
	10	1450
/oh/	0	8000
	5	45
	10	15

Table 4.5: The concentration of energy in different frequencies for different phonemes.

4.4 CHOC Plots.

Carrying out a windowed HOC analysis on a signal of considerable duration (~500ms) with a window length of 30ms and an overlap of 50% tends to yield a very large number of HOC data sets. The corresponding HOC plots prove to be of little or no use due to the high concentration of graph in the same HOC plane as shown in Figure 4.20. Hence, a new means of describing the HOC data had to be developed. A HOC plot by definition [25] is a plot of higher order zero-crossing counts against the corresponding filter order. The new representation, termed *CHOC Plot*, describes the number of higher order zero-crossings versus the corresponding window frame for each filter order. The 'C' in CHOC stands for 'columnised' referring to the 'column' data being plotted rather than the row data as in the case of a HOC plot. The resulting CHOC plots present the HOC data in a manner much easier to interpret. The CHOC plot may be compared to a TFR (Time-Frequency Representation) of the signal and this aspect will be discussed in detail in a later section. The curves in the CHOC plot are now termed '*Isofils*' as they connect HOC data points corresponding to the same filter order.

It should be noted here that as the filter order increases the isofils tend to level out and so do not contain as much useful information as isofils of lower orders. It may be seen from Figure 4.21 that as $j \to \infty$, $D_j \to 330$, which normalises to π - the maximum number of zero-crossings possible in a 30ms window, hence the maximum frequency present. Therefore, the greater the value of j, the less useful the information as a speech recognition parameter. After much examination of the CHOC plots, it was decide that only up to the second order of HOC data was of

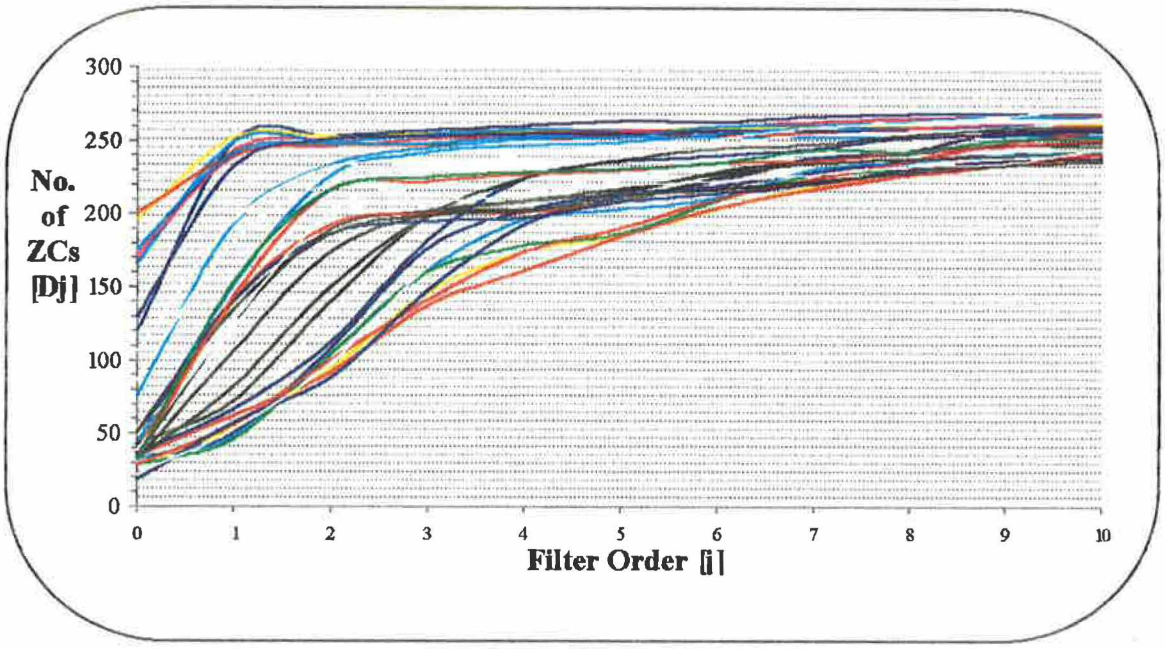


Figure 4.20: HOC plot for fully windowed speech signal: Stop.

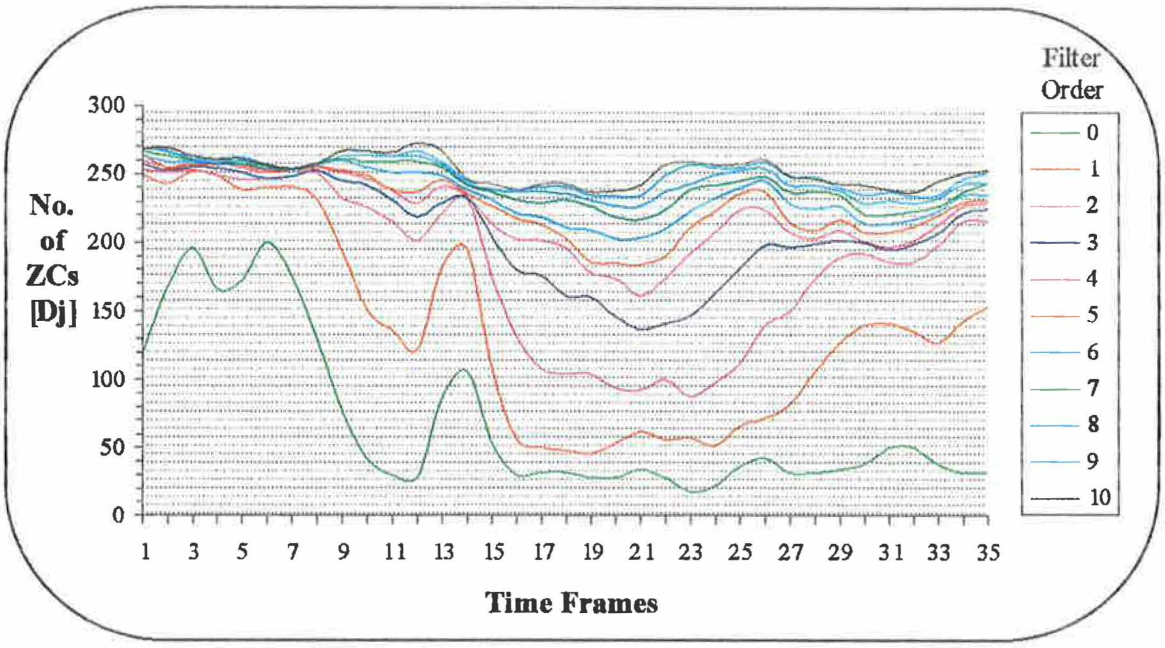


Figure 4.21: CHOC plot for the same fully windowed speech signal as in Figure 4.20. Note the clear phonemic structure of the signal.

considerable benefit. Above this proved too computationally expensive yielding information whose benefit was not proportional to the time taken to achieve it. Hence, D_0, D_1 and D_2 were assumed sufficient to allow adequate discrimination between signals.

4.5 Significance of HOC Values D_0, D_1, D_2.

In general, the sequence of values D_j produced during HOC analysis may be viewed as another type of spectrum, with $D_0/2(N-1)$ (the normalised identity HOC) corresponding to the fundamental frequency of the signal. However, these D_j values actually describe the signal in a simpler manner, owing to the synonymous nature of differencing and differentiation. Consider D_0 to D_2[†]:

- D_0 is simply the number of zero-crossings in the signal.
- D_1 is the number of peaks and troughs in the signal.
- D_2 is the number of points of inflection in the signal.

This suggests that only D_0 to perhaps D_3 are necessary to discriminate between signals. For example, the tuning fork note and the violin note in Figure 4.22 may be easily distinguished by considering the number of peaks and troughs present in the violin signal (i.e. although D_0 is the same, D_1 is extremely different).

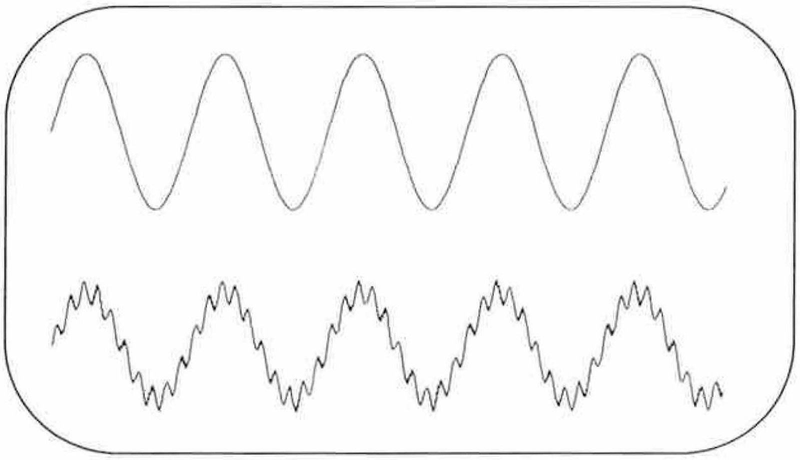


Figure 4.22: Acoustic signals for a tuning fork and violin [42] demonstrating similar zero-crossing information but different higher order crossing information.

4.6 Observations made on CHOC Plots.

4.6.1 Signal Repetition and Discrimination.

The CHOC plots in Figure 4.23 of each of the words from the Gr I vocabulary set demonstrate very well the power of signal discrimination contained within HOC analysis. As few as three parameters (D_0-D_2) appear to provide enough signal information to distinguish one utterance from another.

[†] A full explanation of how this is so is given in Appendix C2.

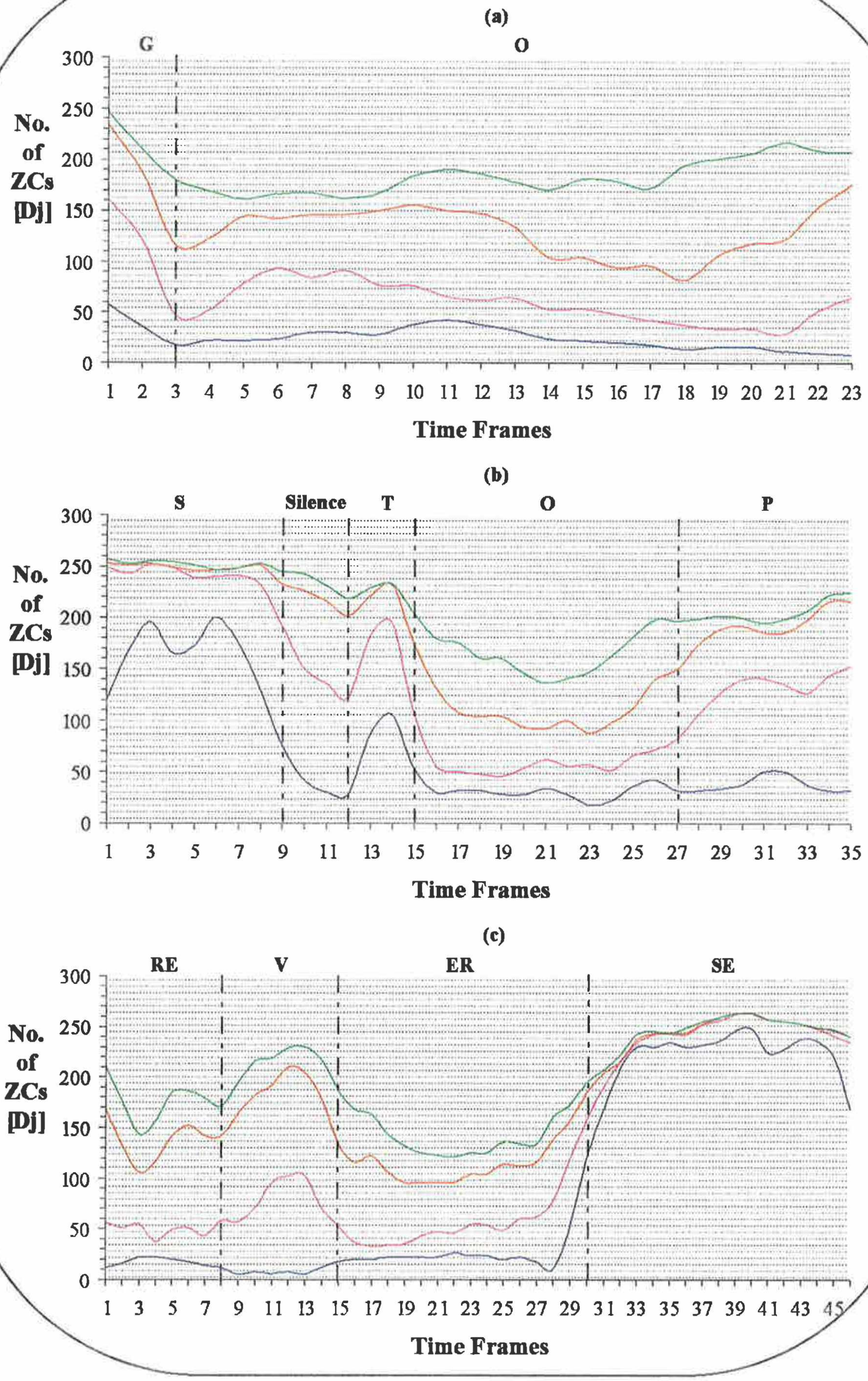


Figure 4.23: CHOC plots of Gr I vocabulary set with approximate phonemic boundaries: (a)Go, (b)Stop, (c)Reverse.

OVER➔

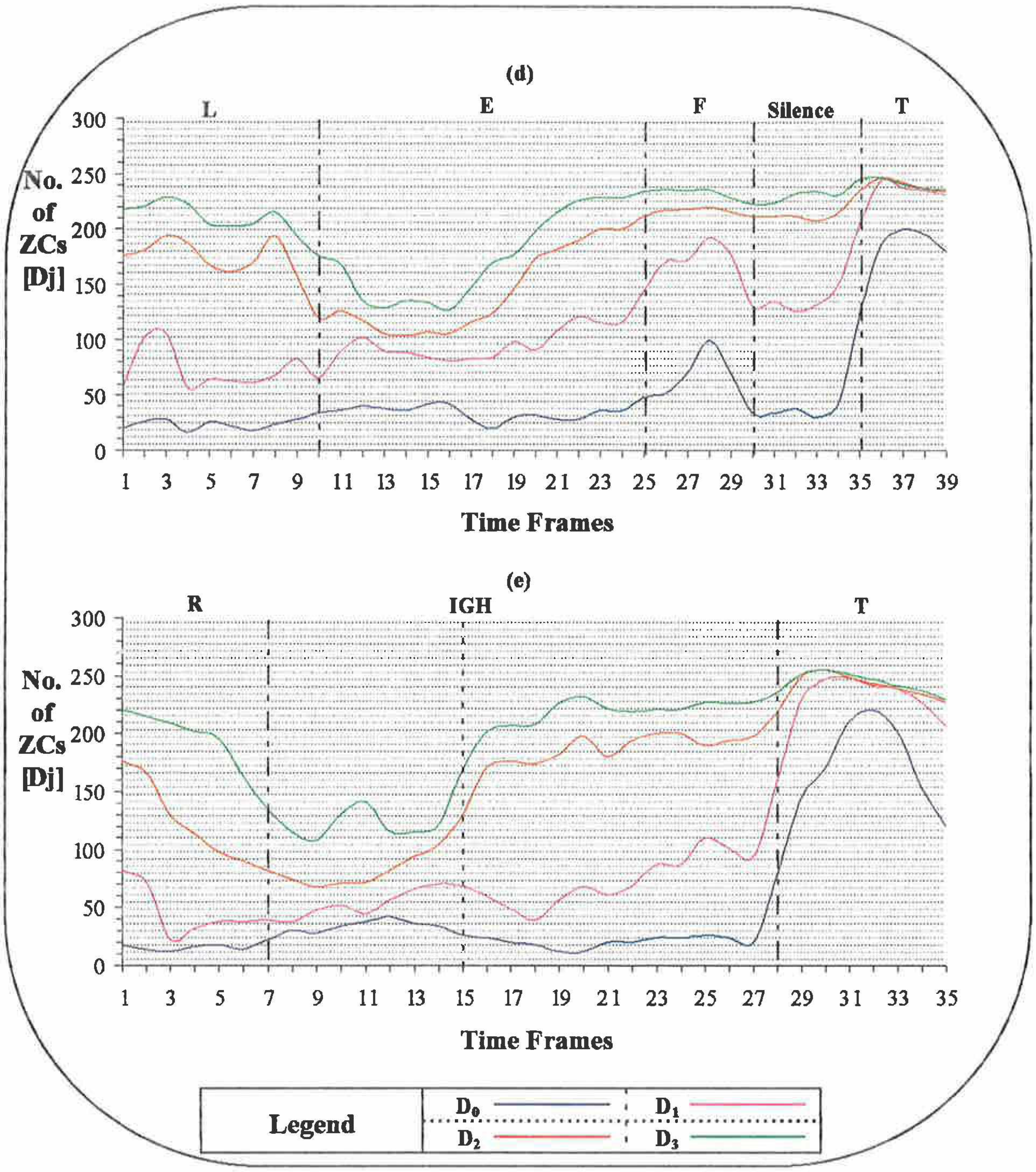
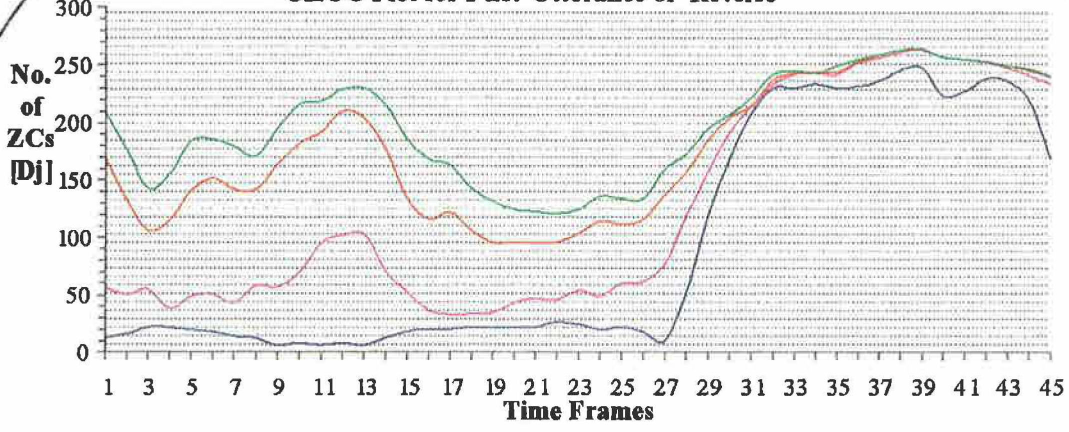
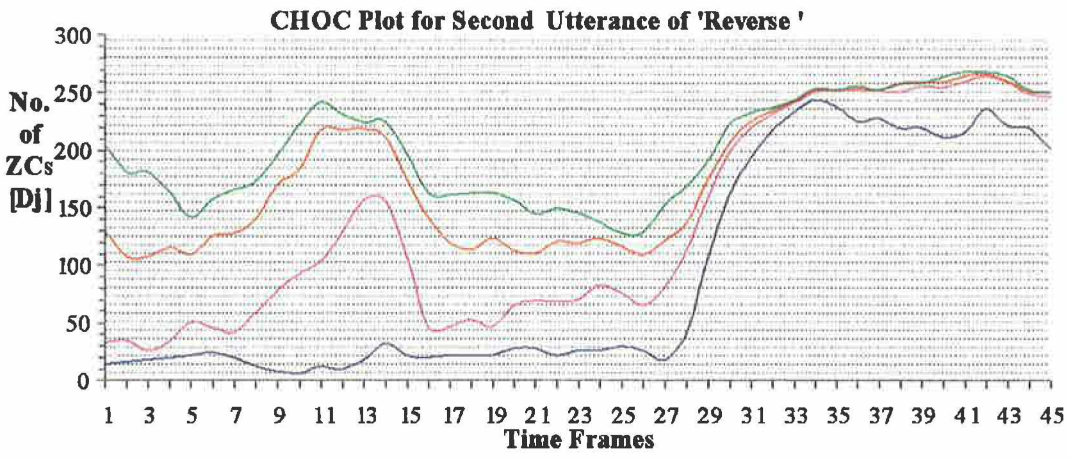


Figure 4.23 (Contd.): CHOC plots of Gr I vocabulary set with approximate phonemic boundaries: (d)Left, (e)Right.

However, a fundamental property required by a reliable feature extractor is repeatability: can the same set of parameters be obtained each time for different utterances of the same word? The CHOC plots given in Figure 4.24 are of three different utterances of the word 'Reverse' and demonstrate quite clearly that the system's repeatability is of high quality. The fact that HOC data is accurately repeatable adds to its viability in speech recognition.





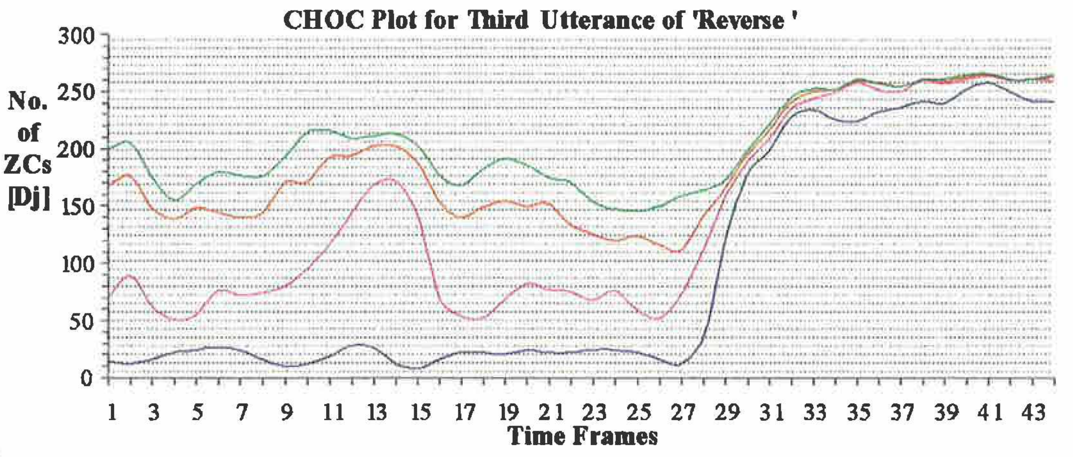


Figure 4.24: CHOC plots corresponding to three utterances of the word 'Reverse' clearly demonstrating the repeatability of HOC analysis.

4.6.2 Phonetic/Phonemic Detection and Discrimination.

The HOC technique has an advantage over other zero-crossing techniques in its power to detect the presence of phonemes that would have otherwise gone unnoticed. CHOC plots tend to complement voiced fricatives such as */v/* in the word '*Reverse*'. D_0 (the parameter used by most zero-crossing techniques) does not distinguish this phoneme from its surrounding vowels; however after just one differencing stage, the */v/* is highlighted. This is due to the removal of the voiced component leaving only the unvoiced sound, thus increasing the dominant frequency). The same can be observed in the case of the phoneme */p/* in the utterance '*Stop*'. The isofil D_0 shows little difference between this phoneme and */δ/* (the 'o' sound); however, D_1 and D_2 amplify the unvoiced components of */p/* thus distinguishing it from the vowel */δ/*.

4.6.3 Phonetic/Phonemic Segmentation.

The CHOC plots prove to be a powerful tool in phonemic segmentation. Clearly defined phonemes are visible in these plots with higher frequency signals such as */s/* yielding upper plane isofils and lower frequency signals such as */u/* yielding lower plane isofils. Mixed voiced-unvoiced signals such as */v/* are portrayed with low D_0 values and higher (even spiked) D_1 and D_2 values. This clear phonemic definition suggests the HOC method to be a suitable means to separate utterances into their phonemic segments.

4.6.4 Preservation of Time Domain Characteristics.

The CHOC plots tend to preserve certain characteristics present in the time domain signal. For example, in the phonetic structure of the word '*Stop*', there is a distinct period of silence before the burst of energy from the plosive */t/*. This may be attributed to the shifting of the tongue into position from the */s/* sound to the top-front of the palette and the intensifying of air pressure behind it before the sudden release of such energy. The silent period is clearly present in the CHOC plot of the utterance '*Stop*'. It is portrayed as a channel between the */s/* and */t/* at time frames 9-12. D_0 is very low and vowel-like, D_1 is too high for a vowel and D_2 almost suggests the signal to be of high frequency. This would imply that this part of the signal consists mainly of the broad band frequency nature of background noise, which in this case is assumed to be a silent period. The same phenomenon may be observed between the */f/* and */t/* of the utterance '*Left*'.

4.7 Performance of HOC Analysis.

To investigate how powerful the HOC technique may be at discriminating sounds a series of tests were performed on the system first using the vowel sounds in the Gr III vocabulary set and then the consonants in the Gr IV set. These tests would determine whether the system was capable of distinguishing between closely sounding syllables.

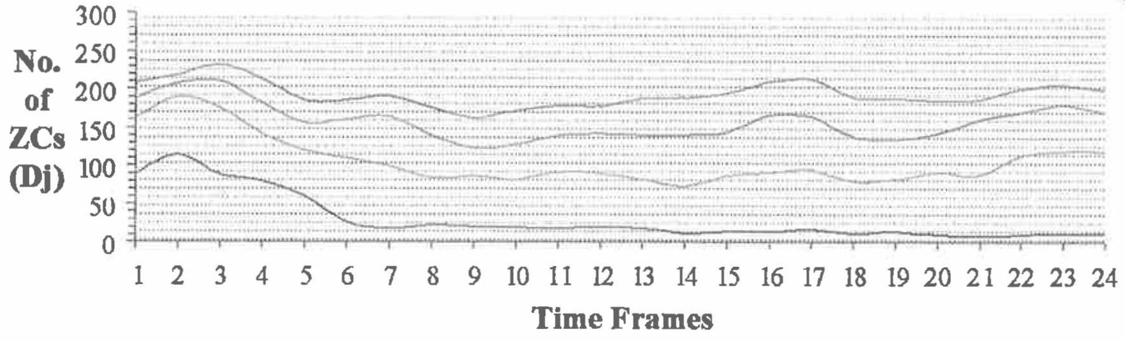
4.7.1 HOC Analysis of Vowels.

The Gr III vocabulary set, as described in Appendix A4, was subjected to HOC analysis up to D_3, using *Hocalg2.m* and the resulting CHOC plots were constructed as shown in Figure 4.25. It should be noted that each vowel was prefixed with the phoneme */k/* to allow similar entry into the vowel sounds. It was observed that in all the cases the vowel section of the syllable accommodated considerably longer duration than the preceding */k/* as expected and consisted of moderately flat isofils due to the quasi-stationarity of the sound.

Table 4.6 describes the average isofil levels for each vowel sound and the following conclusions may be drawn from this:

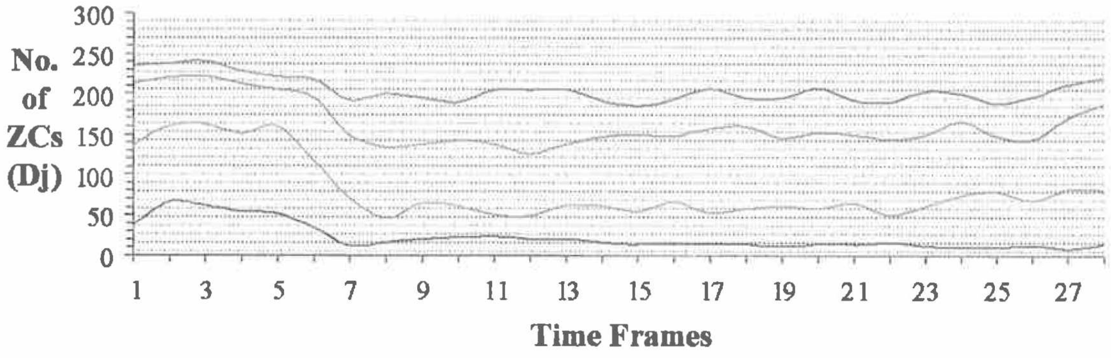
- D_0 proves of little use in differentiating one vowel sound to the next.
- D_3 is of little help, as the D_3 values for each vowel sound are quite similar. This was expected for consonants but not vowels.
- It may be observed that at the end of each CHOC plot the isofils tend to rise dramatically. This is merely due to the mouth closing, the sound becoming more whisper-like and thus an increase in the presence of higher frequencies. This phenomenon would not occur if the vowel was enclosed in a full word.
- Frame 22 onwards of */εi/* tends towards the isofil structure of the */i/* sound as the sound passes from the back of the mouth to the front. Again, this would not be very apparent in a whole utterance, but would prove to be an extra discrimination factor (e.g. an /ai/ in the utterance *Right*).

(a)



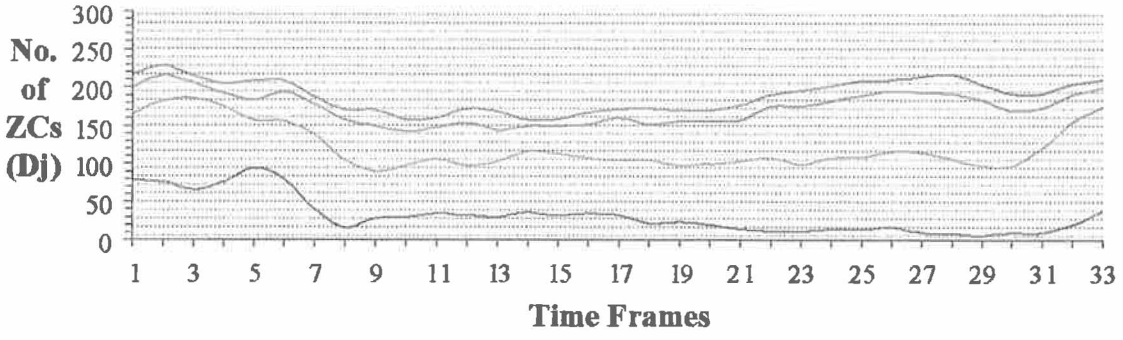
300
250
200
150
100
50
0
No. of ZCs (Dj)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Time Frames

(b)



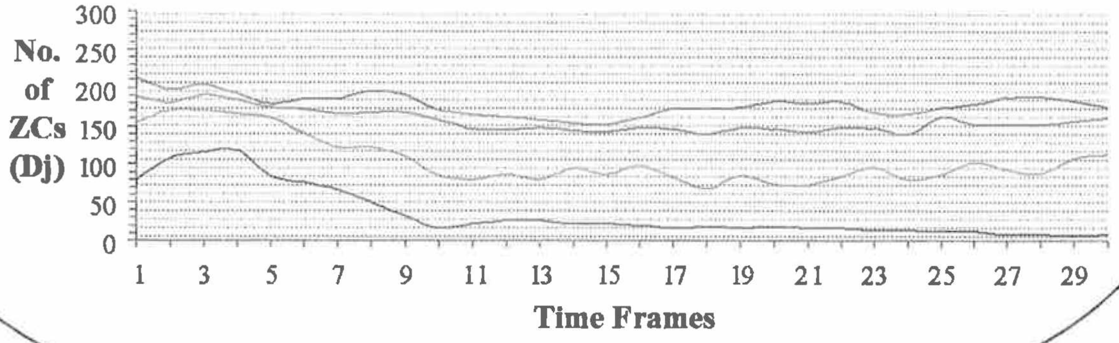
300
250
200
150
100
50
0
No. of ZCs (Dj)
1 3 5 7 9 11 13 15 17 19 21 23 25 27
Time Frames

(c)



300
250
200
150
100
50
0
No. of ZCs (Dj)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33
Time Frames

(d)



300
250
200
150
100
50
0
No. of ZCs (Dj)
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29
Time Frames

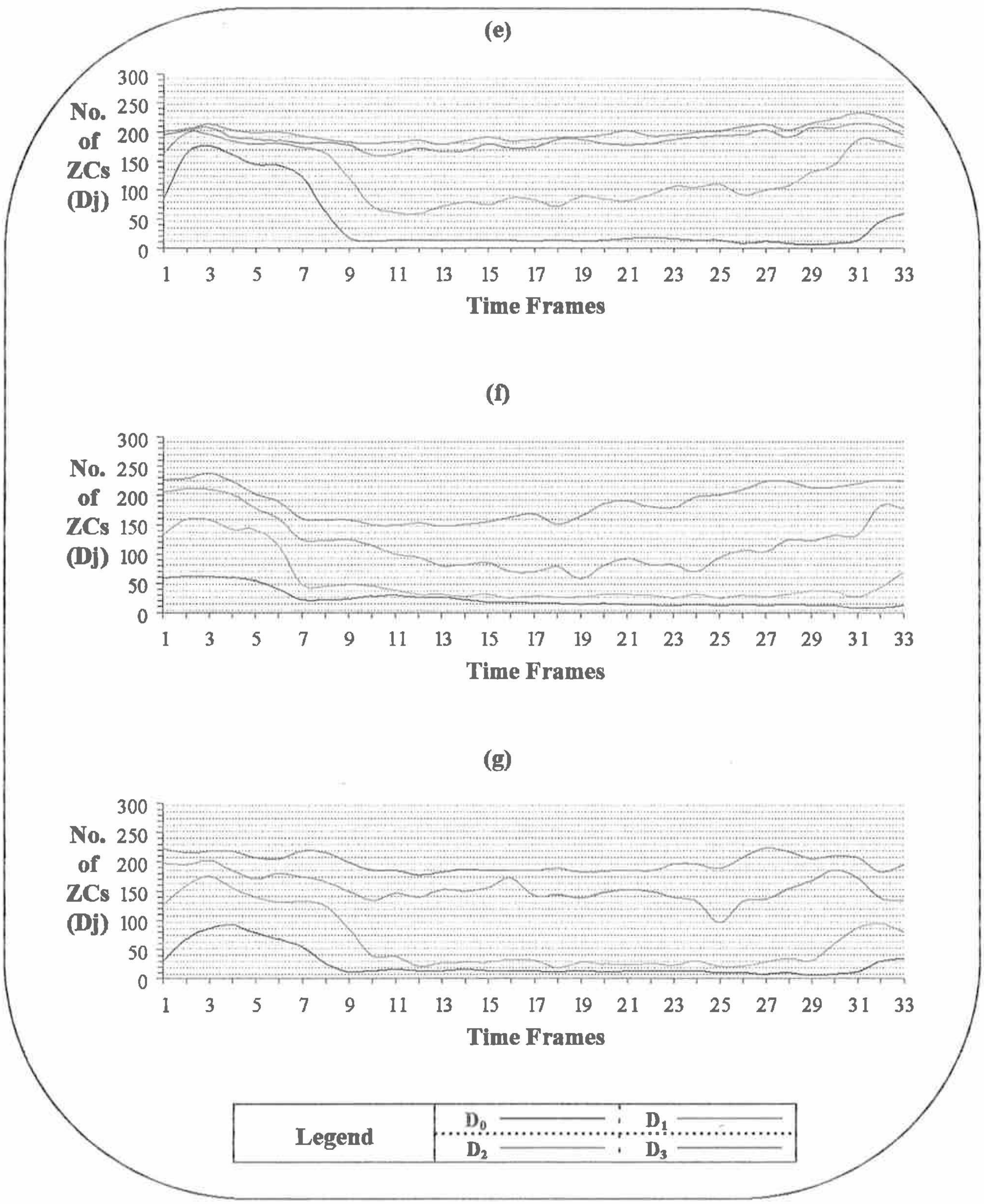


Figure 4.25: CHOC plots for Gr III vocabulary set ie. Vowel sounds (a) Kah, (b) Kaw, (c) Kay, (d) Ke, (e) Key, (f) Ko, (g) Koo.

- D_2 would prove sufficient to distinguish between */əʊ/* and */u/* even though D_0 and D_1 values are quite similar.
- The similarity between the CHOC plots of */ε/* and */æ/* would suggest that words such as *Pen* and *Pan* may be confused by the system.
- The amount of ripple in the isofils (as will be shown in a later section) will be taken into consideration by the Dynamic Time Warping stage and should prove to be an extra recognition factor.

Phoneme	D_{0Av}	D_{1Av}	D_{2Av}	D_{3Av}
/δ/ (kaw)	20	60	150	200
/əʊ/ (ko)	20	30	85	180
/u/ (koo)	15	30	140	200
/ε/ (ke)	20	90	150	180
/æ/ (kah)	20	90	145	190
/εi/ (kay)	30/15	105/125	155/175	170/210
/i/ (key)	15	90	175	200

Table 4.6: Average D_0-D_3 Values of Gr III Vocabulary Set after Hocalg2.m.

4.7.2 HOC Analysis of Consonants.

The words in vocabulary set Gr IV were then subjected to HOC analysis in order to test the technique's capability in distinguishing different consonants. The corresponding CHOC plots up to D_3 are shown in Figure 4.26 and Table 4.7 describes the average isofil level for each consonant. It should be noted that all the consonants were suffixed with the phoneme */æ/* to allow a similar exit from each sound. The following points were noted:

- D_0 is very low for */b/*, */d/* and frames 3-6 of */g/* illustrating the existence of a voiced component.
- The */g/* is broken into two sections, the first being a strong unvoiced almost */k/* sound and the second being voiced. This double section will help in the sound's recognition.
- The phonemes */b/* and */d/* may easily be distinguished by their D_1 and D_2 values.

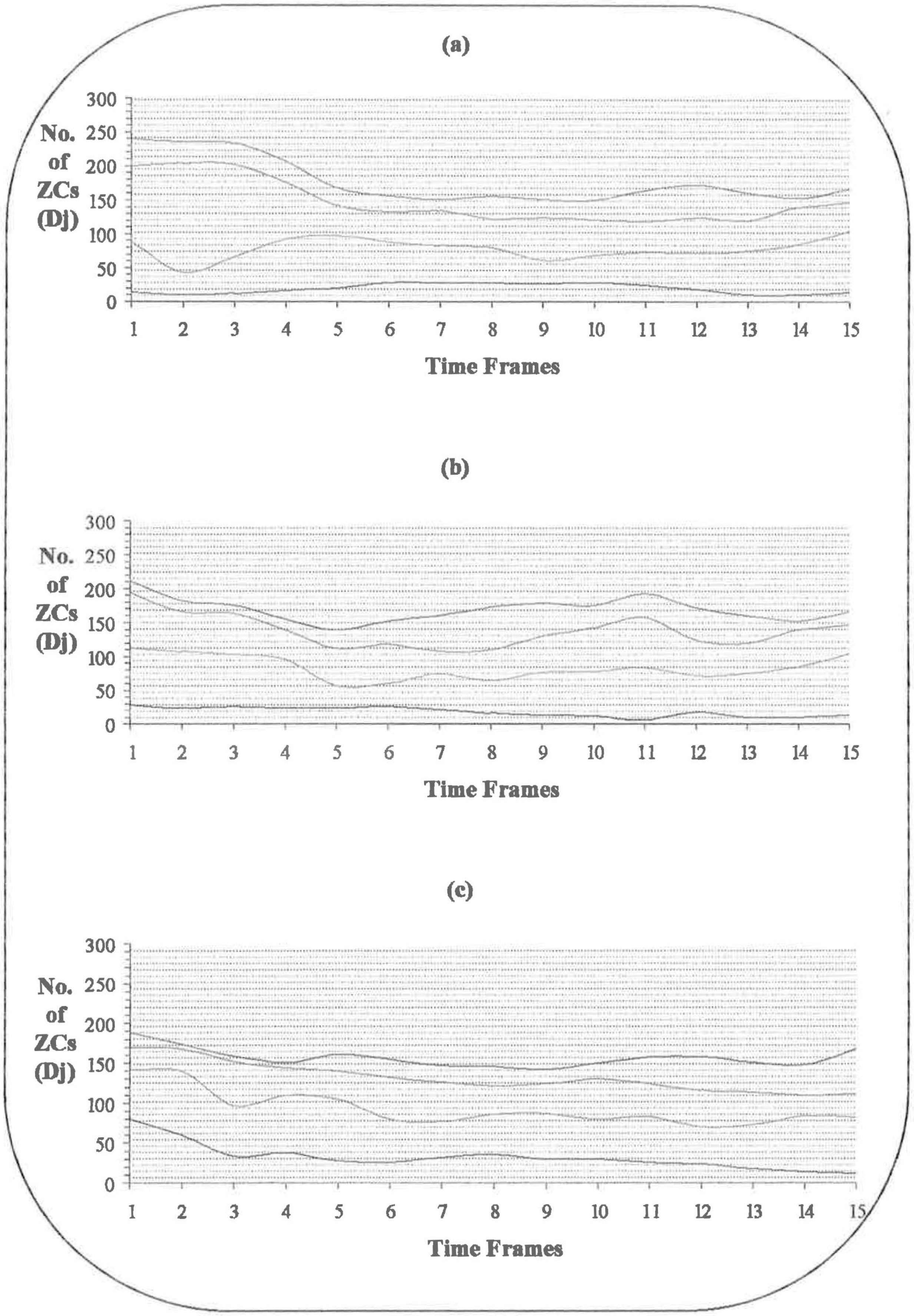
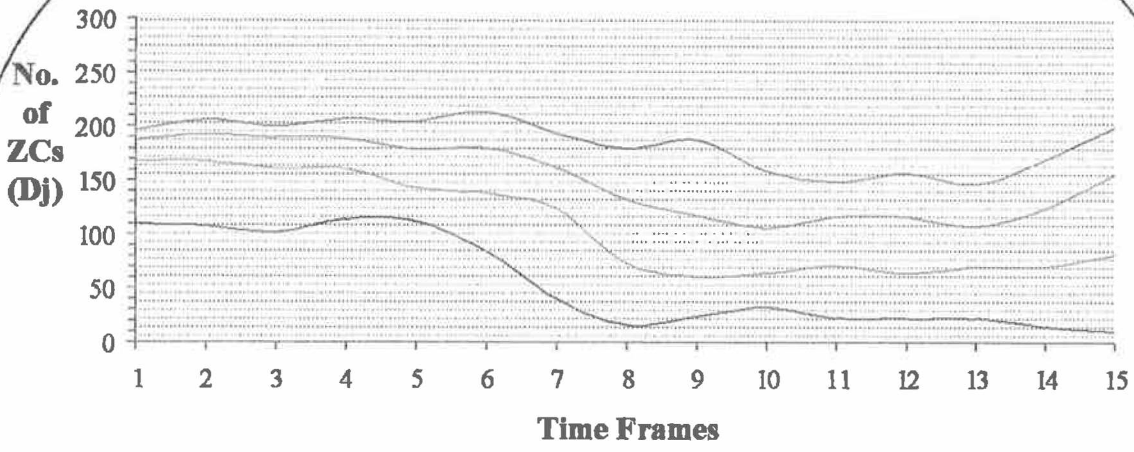


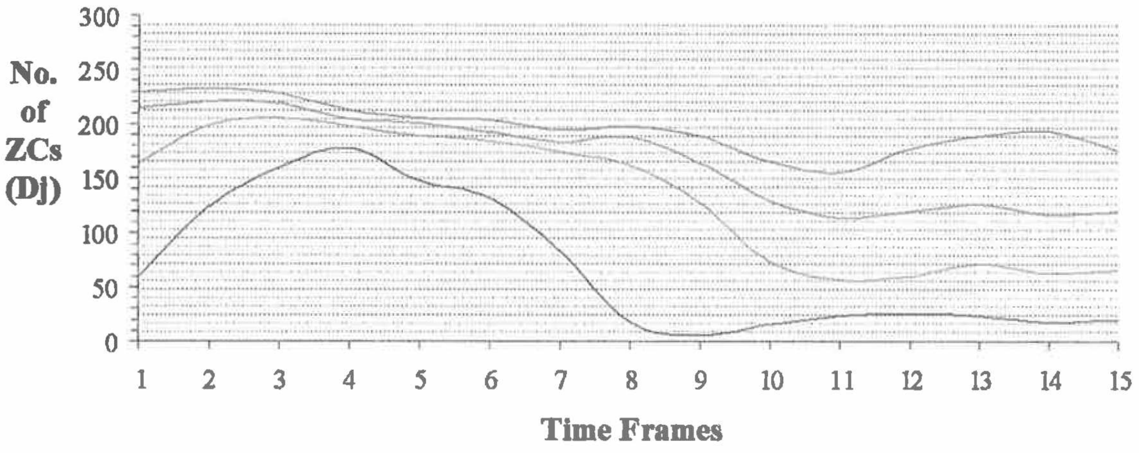
Figure 4.26: CHOC plots of Gr IV vocabulary set: (a)/b/, (b)/d/, (c)/g/.

OVER➔

(d)



(e)



(f)

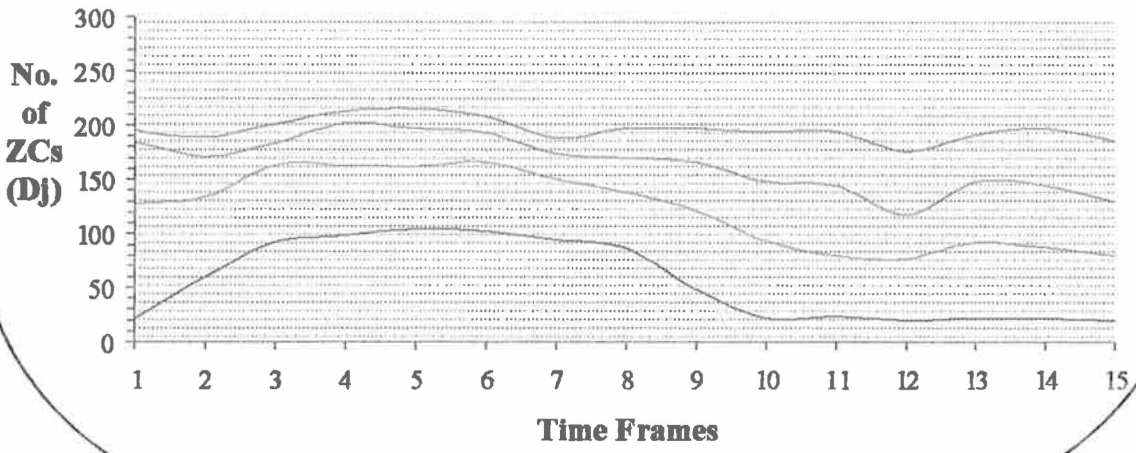
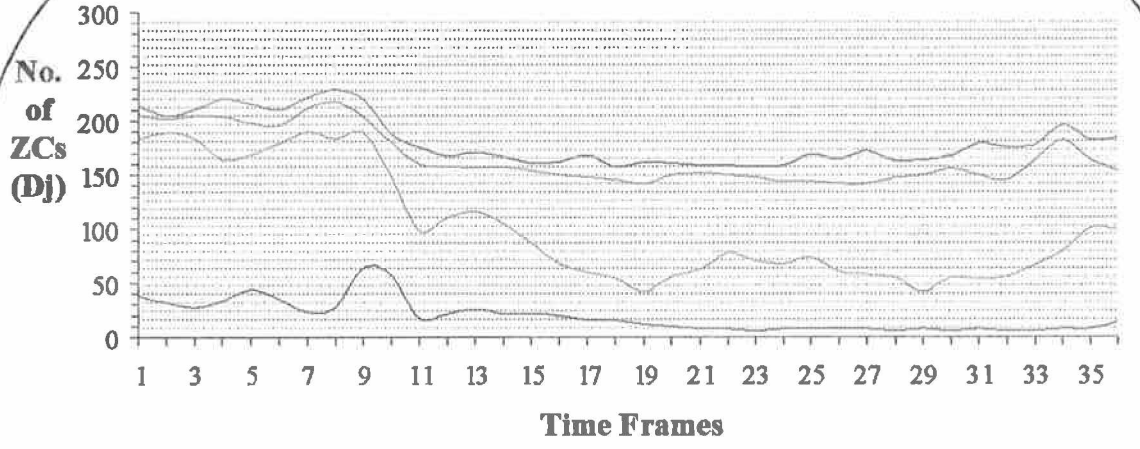


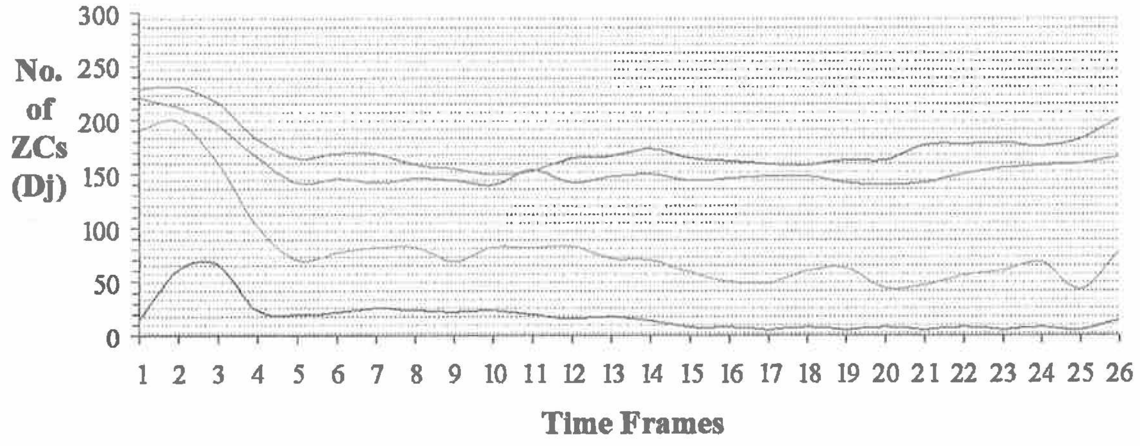
Figure 4.26 (Contd.): CHOC plots of Gr IV vocabulary set: (d)/k/, (e)/t/, (f)/p/.

OVER→

(g)



(h)



(i)

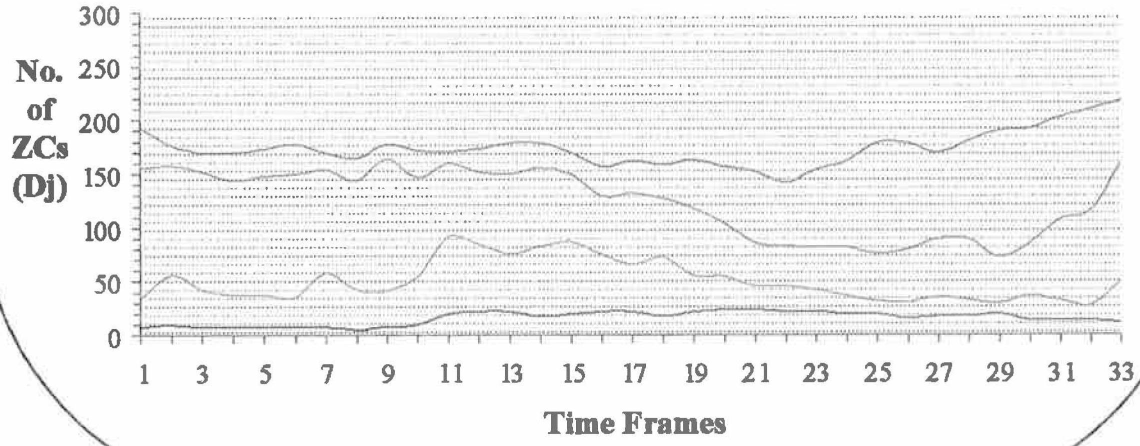


Figure 4.26 (Contd.): CHOC plots of Gr IV vocabulary set: (g)Fin, (h)Thin, (i)No.

OVER➔

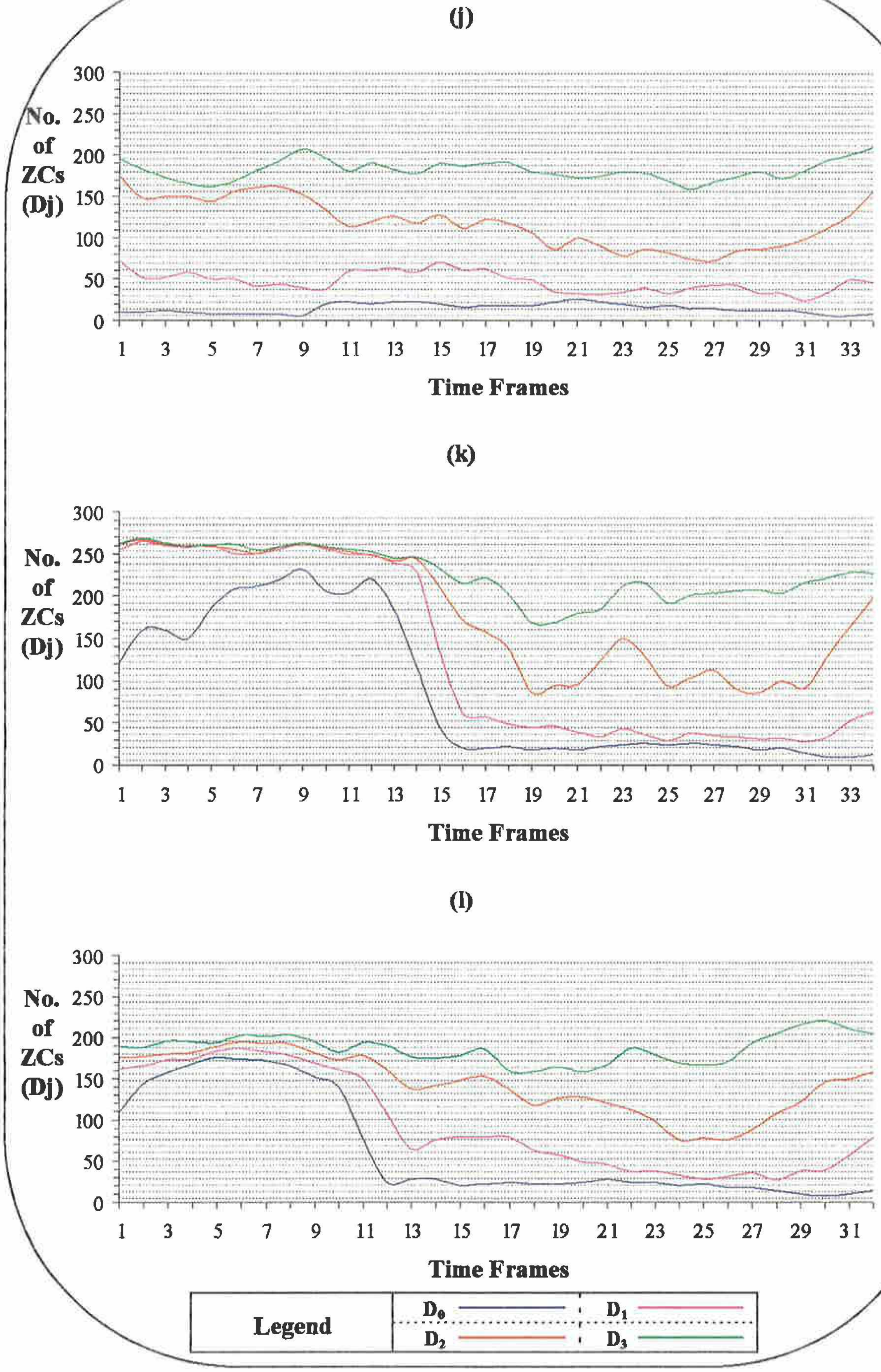


Figure 4.26 (Contd.): CHOC plots of Gr IV vocabulary set : (j)Mow, (k)So, (l)Show.

- The similarity between the $\bar{D}$ values of /*t*/ and /*k*/ does not cause a problem as there is a distinct peaking in the isofils of /*t*/, whereas the isofils of /*k*/ are relatively flat.
- *Fin* Vs. *Thin*: HOC analysis established very few differences between these utterances. The difference in the duration of the fricative parts is the only distinctively contrasting feature and, (as will be shown in a later section), this would be compensated against by the Dynamic Time Warping stage, thus causing confusion between these utterances. Sampling at a higher frequency may help but would result in slower analysis.
- *No* Vs. *Mow*: Both nasals are of similar duration with virtually the same D_0 values. Both D_1 and D_2 for the two utterances are of the same level but differ slightly in the amount of ripple. This would suggest that confusion may occur between these words.
- *Show* Vs. *So*: The /ʃ/ sound exhibits much lower D values than /*s*/, with D_3 of /ʃ/ being ~190 and D_0 of /s/ being ~200. Also, the isofils of /ʃ/ are more distributed over the CHOC plane than those of /*s*/which are closely packed. The broader band frequency content of /ʃ/ and higher frequency content of /*s*/ would explain this phenomenon. This would suggest a high possibility of distinguishing these phonemes.

Phoneme	D_{0Av}	D_{1Av}	D_{2Av}	D_{3Av}
/t/	110	170	200	210
/k/	110	165	190	205
/p/	70	155	185	195
/g/	60/30	145/115	170/155	180/165
/d/	25	110	170	180
/b/	15	60	200	235

Table 4.7: Average D_0-D_3 Values of Gr IV Vocabulary Set after Hocalg2.m.

Chapter 5

5. Recognition Stage for CHOC.

In this chapter various recognition and decision stages are studied and the most suitable one is chosen. A number of tests are performed on the system including success rate tests, the system's sensitivity to noise, non-linear phase changes and intonation changes. A comparison of the CHOC system to the Spectrogram is also given.

In the last chapter it was shown how suitable a speech analyser the HOC technique was. The next step is to embed this analyser into a recognition system, calculate its percentage success rates and test it in different scenarios. Three recognition techniques were considered:

- Simple Voiced-Unvoiced Decision.
- Phonetic Categorisation.
- Cluster Analysis/Pattern Matching.

The first technique may have been quite easily implemented. However, since HOC analysis yielded considerably high quality information from the speech signal, it was felt that to make a simple voiced-unvoiced decision would be a great waste of valuable information. Also, low signal discrimination and low success rates would result unnecessarily from a voiced-unvoiced decision, spoiling the purpose of the HOC technique. Henceforth, this method was abandoned.

The second method was considered to retain more of the important signal information extracted by HOC analysis than a voiced-unvoiced decision and so was further investigated. A group of ten phonetic categories was established as follows:

Cat1 : Voiced Stops (/b/,/d/,/g/)	Cat6 : Nasal (/n/,/m/)
Cat2 : Voiced Fricatives (/v/,/z/,/ʒ/,/ð/)	Cat7 : Unvoiced Stops (/p/,/t/,/k/)
Cat3 : Vowel-like (/l/,/r/)	Cat8 : Unvoiced Fricatives-Strong (/s/,/ʃ/)
Cat4 : Vowel-Front (/i/,/I/,/æ/,/ε/)	Cat9 : Unvoiced Fricatives-Weak (/θ/,/*f*/,/h/)
Cat5 : Vowel-Back (/u/,/ʌ/,/ʊ/)	Cat10 : Transition Period.

Each word in the stored vocabulary was then roughly categorised using this set (e.g. Reverse : 34238). Then, each set of CHOC data corresponding to each word was studied to establish a direct association between D_0-D_2 and each category (e.g.

If $(a \le D_0 \le b)$ & $(c \le D_1 \le d)$ & $(e \le D_2 \le f)$[†] , then this time frame may be categorised as CatX).

Word	Phoneme	D_0	D_1	D_2
Go	/g/	30-70	110-180	180-240
	/əv/	8-45	22-95	70-180
Stop	/s/	120-210	220-260	240-260
	/t/	70-110	160-200	200-235
	/δ/	14-38	40-78	80-155
	/p/	20-90	60-180	110-220
Right	/r/	5-20	50-100	150-200
	/ai/	20-50	40-70	70-110
		10-30	50-90	160-210
	/t/	180-200	210-220	215-225
Left	/l/	10-30	20-106	120-200
	/ε/	15-50	50-100	100-160
	/f/	30-110	110-215	190-240
	/t/	135-220	210-245	220-250
Reverse	/rε/	5-30	20-100	100-180
	/v/	5-35	70-175	170-225
	/εə/	15-35	30-85	90-160
	/s/	190-255	200-270	210-275

Table 5.1: CHOC Ranges for GrI Vocabulary.

The category ranges in Table 5.1 were established after studying the CHOC data for GrI vocabulary frame by frame. Now, after HOC analysis the unknown utterance may be categorised frame by frame to give a long string of category characters (e.g. 11111334446444777). All that is required from this string are the unlike-terms; hence, the like-terms may be removed, (using *Rlktrms.m*), leaving 134647 in this case. This character string may then be compared to the stored strings in the vocabulary. If a match is found, then the word is recognised. However, it must be noted that by removing time information in this way, outlier categories (such as 6 in this case) gain equal representation in the category string.

[†] a,b,c,d,e,f represent maximum and minimum D_j values for each phoneme category.

Upon testing, this method did not prove to be as successful as first hoped. The need to categorise the vocabulary proved difficult at times. The main problem arose when the CHOC data was in transition from one phoneme to the next[‡]: sloping isofils within such a period resulted in the detection of many categories that were not even present. To overcome this, it was necessary to reduce the number of categories to three or four. This coarsening of categories, (as in the case of the voiced-unvoiced decision), tended to neglect valuable signal information within the CHOC data. Hence, this technique was also rejected and so the third option (which is studied in detail in the following section) was examined and eventually selected as the most suitable.

5.1 Cluster Analysis and Pattern Matching.

The third recognition technique considered is pattern matching. This involves the comparison of test patterns with reference patterns to find a *best fit* utterance and may be described in three stages:

- Clustering/Training.
- Dynamic Time Warping/Pattern Matching.
- Decision Rule.

These stages are described in Sections 5.3-5.5, but since Dynamic Time Warping is applied to both the training and pattern matching stages, it is important first to understand what exactly is happening when process is conducted.

5.2 Theory behind Dynamic Time Warping (DTW) [45].

As shown in Figure 5.1, after front-line processing it is necessary to have some type of pattern matching process present in order to recognise the signal. In the case of CHOC data, the simplest way would be to compare frame by frame the test data to a stored template of data.

[‡] The sliding window is covering a section of two different phonemes, hence the number of zero-crossings is averaged out .

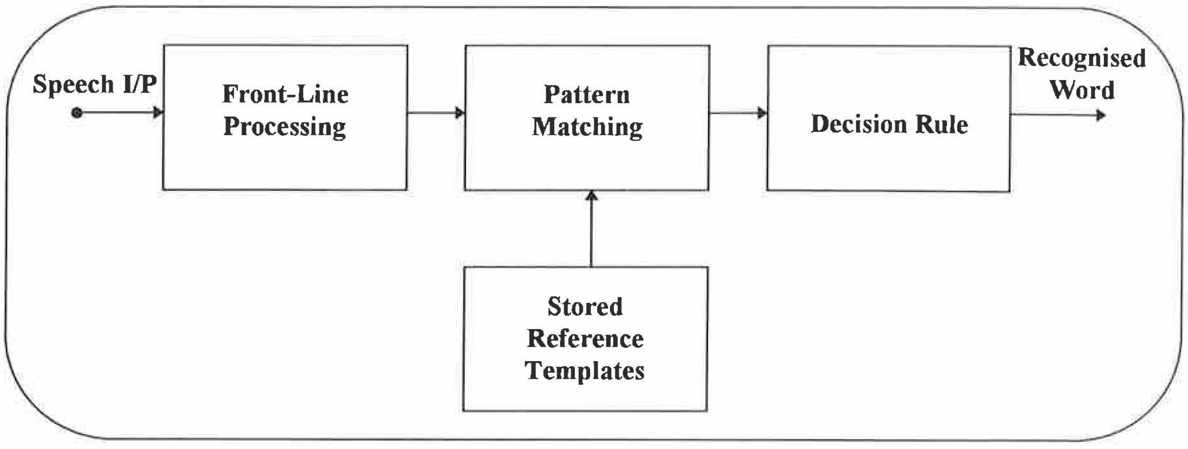


Figure 5.1: A Typical Speech Recognition System.

However, in general words are spoken at different rates even when the same speaker repeats the same word. Hence, globally the test and reference signals would be of different duration. A process called *Linear Time Warping (LTW)* employs frame duration normalisation to force each template to consist of an equal number of frames. However, as demonstrated in Figure 5.2(a) local variation in duration also occurs causing LTW to yield a mismatch in the internal frames of the word.

This local variation is due to the fact that vowels and stressed syllables tend to expand and contract more than consonants and unstressed syllables. Hence, words spoken with differently stressed vowels (intentional or not) may not be recognised as the same word.

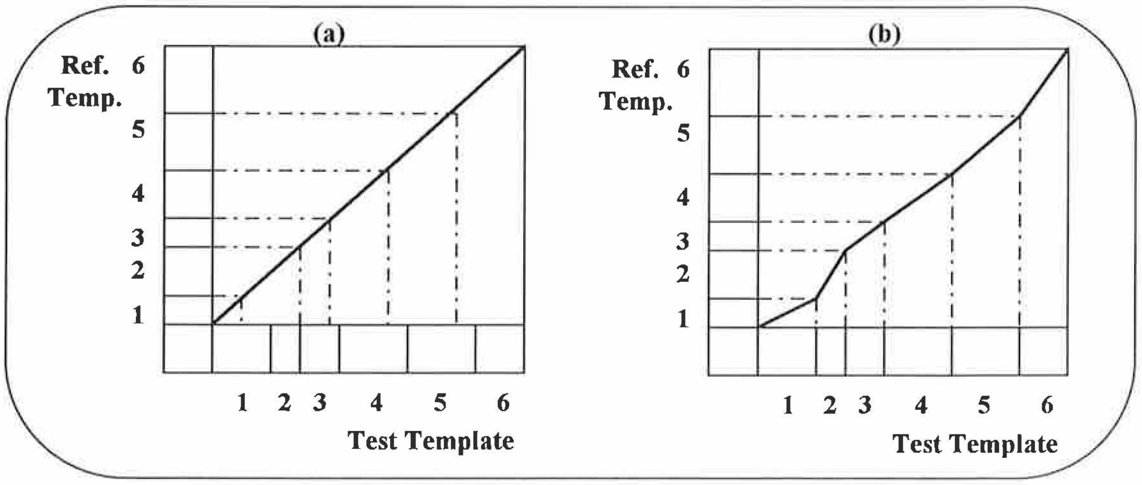


Figure 5.2: **(a)** Mismatching of frames by LTW and **(b)** Non-linear time alignment of frames by DTW.

The method known as *Dynamic Time Warping (DTW)* [54] was developed to overcome the problems of variable phonemic duration, by imposing a *non-linear time alignment* on the test and reference signals as shown in Figure 5.2(b).
DTW operates using a mathematical concept known as *Dynamic Programming*. Simplified, it may be described as follows:

- At each time interval the 'best fit' frame is decided under certain constraints.
- This is repeated until all the time frames have been compared resulting in a non-linear time alignment or *Minimum Path* which best maps the test signal onto the reference template.

The process is best described by example 5.1 given in Appendix D1.

5.3 Clustering and Training of System.

A cluster is simply a group of reference templates associated with the same utterance. Instead of only one template per word, clusters of L templates for each word are created and stored to allow comparison with an unknown test signal. In the case of this thesis, ten tagged[†] CHOC templates were first created for each word using *Hocalg5.m* (described in Figure 5.3). A training process was then executed (in the form of Trainer.m and Train.c - described by Figures 5.4 and 5.6) to find the five most similar templates from the group of ten. These five data sets are stored giving a cluster of five reference templates. In total there exists N×5 reference templates, where N is the number of words in the vocabulary set.
Clustering allows a greater robustness as there now exists clusters of five similar templates, thus increasing the possibility of a match. Cluster analysis is especially effective when the test signal is not as close to the reference signal as expected (e.g. an accented or stressed utterance). A reduction of confusion between two similar sounding words is also apparent. The only disadvantage is that the recognition system may now run slower as there are more reference templates to be compared with the test signal (N×5).

[†] The issue of tagging CHOC data is discussed in Section 5.3.2.

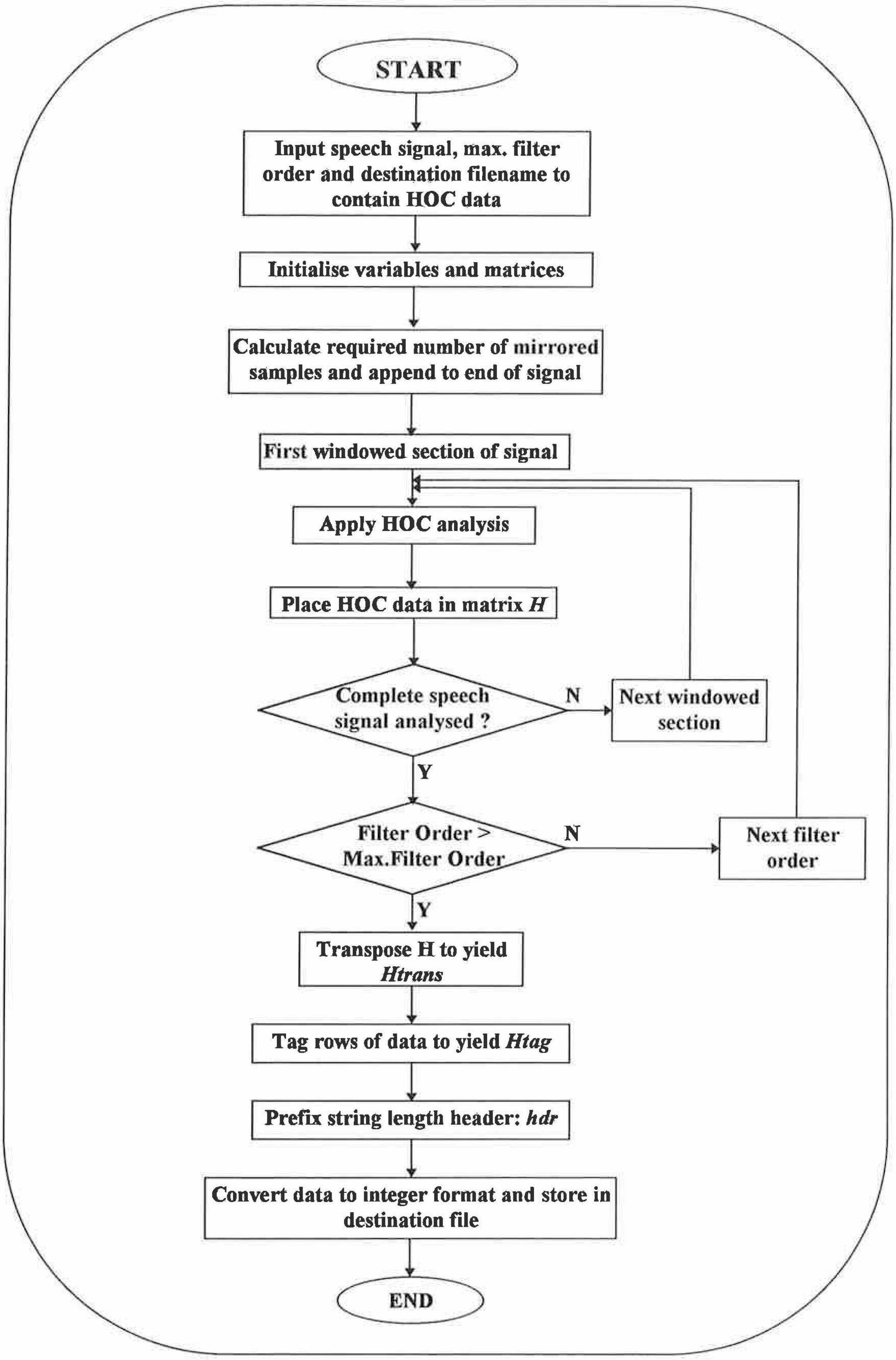

START
Input speech signal, max. filter order and destination filename to contain HOC data
Initialise variables and matrices
Calculate required number of mirrored samples and append to end of signal
First windowed section of signal
Apply HOC analysis
Place HOC data in matrix *H*
Complete speech signal analysed ?
N
Next windowed section
Y
Filter Order > Max.Filter Order
N
Next filter order
Y
Transpose H to yield *Htrans*
Tag rows of data to yield *Htag*
Prefix string length header: *hdr*
Convert data to integer format and store in destination file
END

Figure 5.3: Flowchart explaining the action of Hocalg5.m.

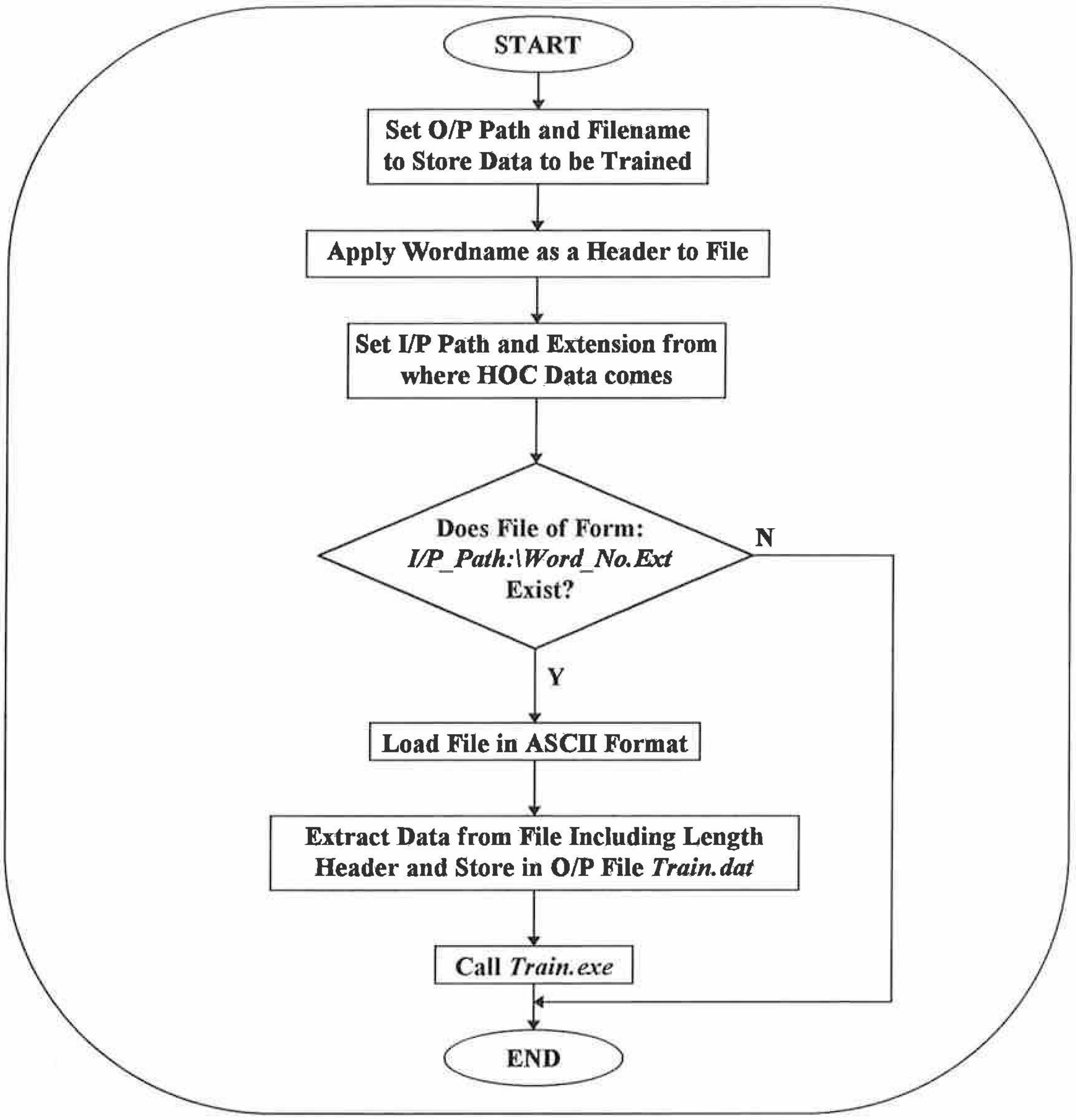

START
Set O/P Path and Filename to Store Data to be Trained
Apply Wordname as a Header to File
Set I/P Path and Extension from where HOC Data comes
Does File of Form: I/P_Path:\Word_No.Ext Exist?
N
Y
Load File in ASCII Format
Extract Data from File Including Length Header and Store in O/P File Train.dat
Call Train.exe
END

Figure 5.4: Flowchart explaining the formating of data by Trainer.m before training of the system.

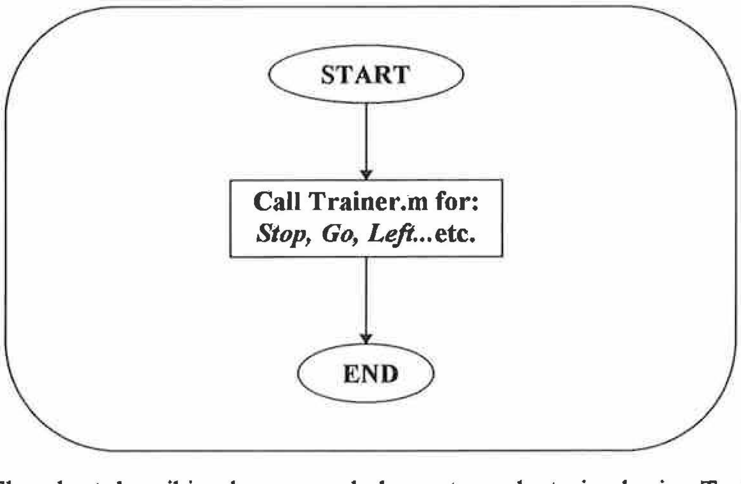


Figure 5.5: Flowchart describing how a vocabulary set may be trained using Trainall.m.

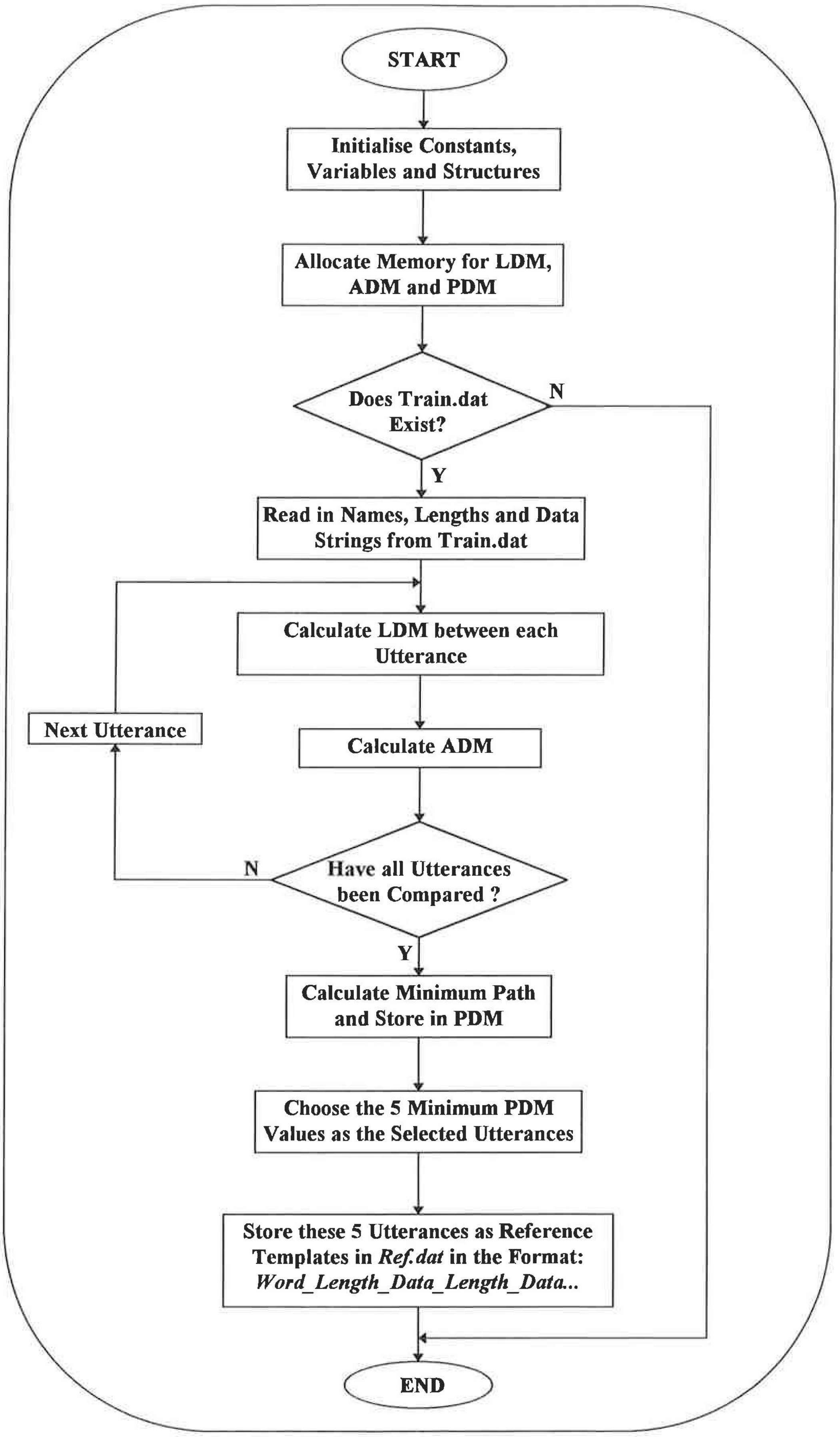

START
Initialise Constants, Variables and Structures
Allocate Memory for LDM, ADM and PDM
Does Train.dat Exist?
N
Y
Read in Names, Lengths and Data Strings from Train.dat
Calculate LDM between each Utterance
Next Utterance
Calculate ADM
Have all Utterances been Compared ?
N
Y
Calculate Minimum Path and Store in PDM
Choose the 5 Minimum PDM Values as the Selected Utterances
Store these 5 Utterances as Reference Templates in Ref.dat in the Format: Word_Length_Data_Length_Data...
END

Figure 5.6: Flowchart showing how Train.c determines the best five templates for each word.

5.3.1 Trainer.m, Train.c and Trainall.m.

The system was trained as described in Section 5.3, by selecting the five closest matching utterances from a group of ten and storing these in memory as reference templates. The C-program *Train.c* (a slightly modified version of that described in [44]) was written to achieve this. However, as described in greater detail in Appendix D2, the m-file *Trainer.m* had to be written to overcome data type incompatibility. *Trainall.m* simply executes *Trainer.m* for every word in the chosen vocabulary set.

5.3.2 Process of Tagging CHOC Data.

It is important to note that the data obtained through HOC analysis is in the form of a W×3 matrix, where W is the number of time frames in the data set. Each row in the matrix describes the D_0, D_1 and D_2 zero-crossing content of a particular time frame. Rather than having three separate patterns to examine and match for each utterance, it would simplify matters, (by avoiding multi-dimensional DTW), if there was only one pattern per utterance. Hence, each D_j data string was tagged onto its predecessor as described in Appendix D3.

5.4 Pattern Matching Stage.

Once the system is trained to recognise a particular set of vocabulary, the next stage is to develop the system to recognise unknown utterances (presented in the form of CHOC data). *Dtwclus6.c* (a slightly modified version of *Dtwclus5.c* as described in [44]) was chosen to conduct this task. As before, there were data compatibility problems and, as described in Appendix D4, *Wordrec.m* was written to overcome such difficulties.

Using DTW, the minimum paths are calculated and displayed and the decision rule (which is described in the next section) is applied. If the chosen utterance has a minimum path value of greater than 4000, the system informs the user that an appropriate match has not been made. Otherwise, the recognised word is displayed. (Note: flowcharts summarising the calculation of the LDM and ADM in both the training and matching stages are shown in Figures 5.9 and 5.10).

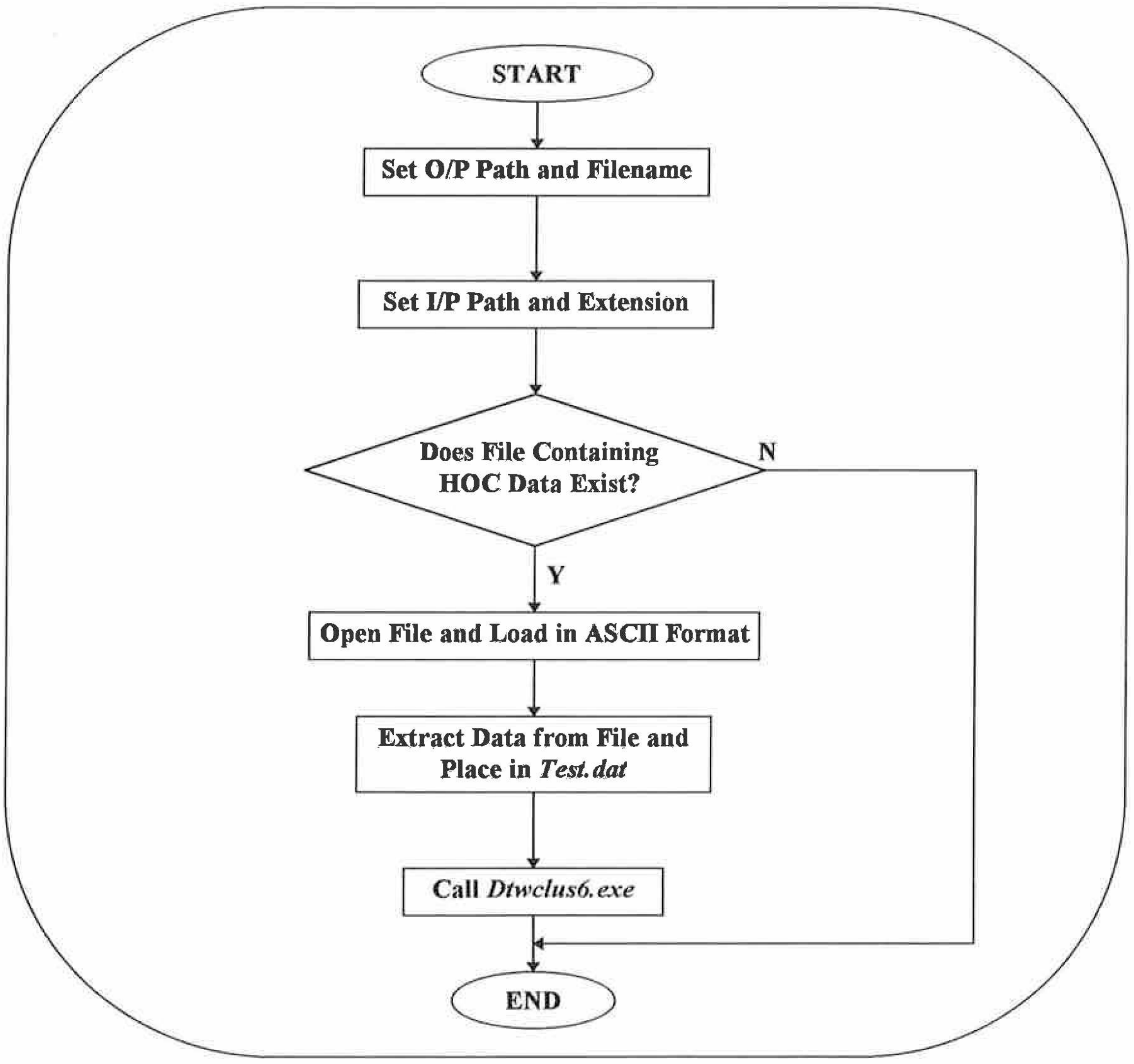

START
Set O/P Path and Filename
Set I/P Path and Extension
Does File Containing HOC Data Exist?
N
Y
Open File and Load in ASCII Format
Extract Data from File and Place in Test.dat
Call Dtwclus6.exe
END

Figure 5.7: Flowchart explaining how Wordrec.m formats the CHOC data before recognition.

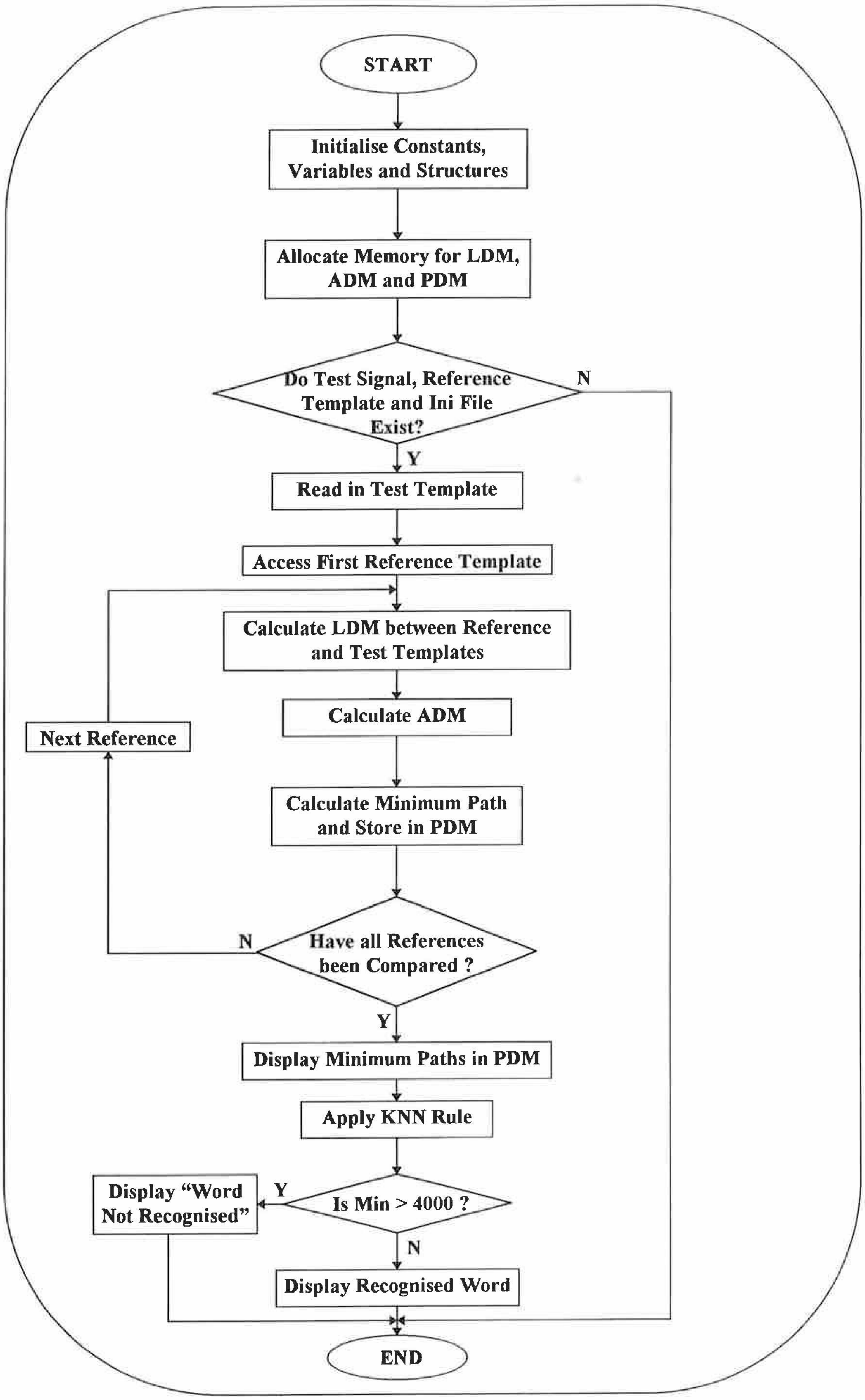

START
Initialise Constants, Variables and Structures
Allocate Memory for LDM, ADM and PDM
Do Test Signal, Reference Template and Ini File Exist?
N
Y
Read in Test Template
Access First Reference Template
Calculate LDM between Reference and Test Templates
Calculate ADM
Next Reference
Calculate Minimum Path and Store in PDM
N
Have all References been Compared ?
Y
Display Minimum Paths in PDM
Apply KNN Rule
Display "Word Not Recognised"
Y
Is Min > 4000 ?
N
Display Recognised Word
END

Figure 5.8: Flowchart expaining the operation of Dtwclus6.c.

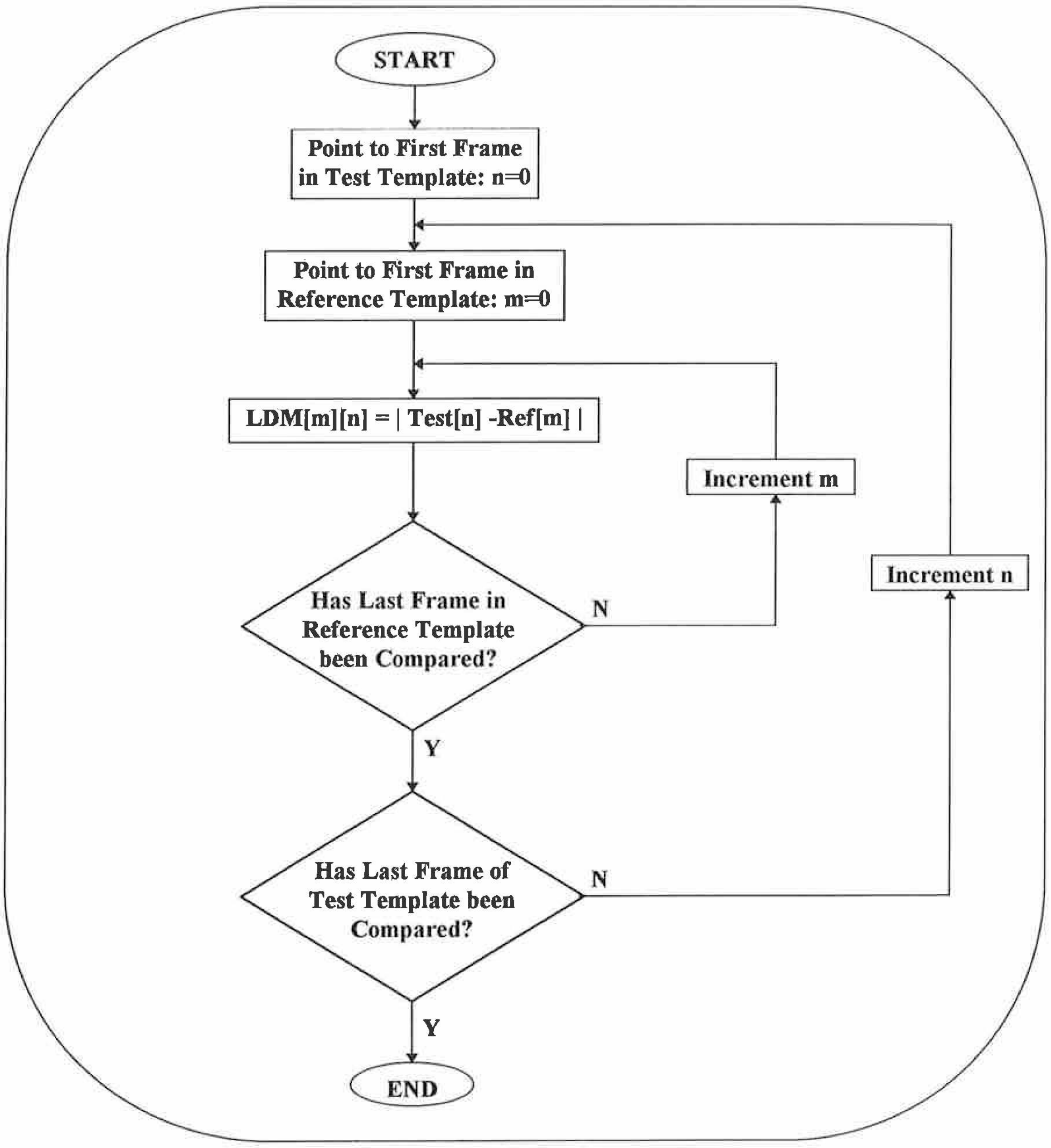

START
Point to First Frame in Test Template: n=0
Point to First Frame in Reference Template: m=0
LDM[m][n] = | Test[n] -Ref[m] |
Increment m
Increment n
Has Last Frame in Reference Template been Compared?
N
Y
Has Last Frame of Test Template been Compared?
N
Y
END

Figure 5.9: Flowchart describing the calculation of the Local Distance Matrix (LDM) in DTW.

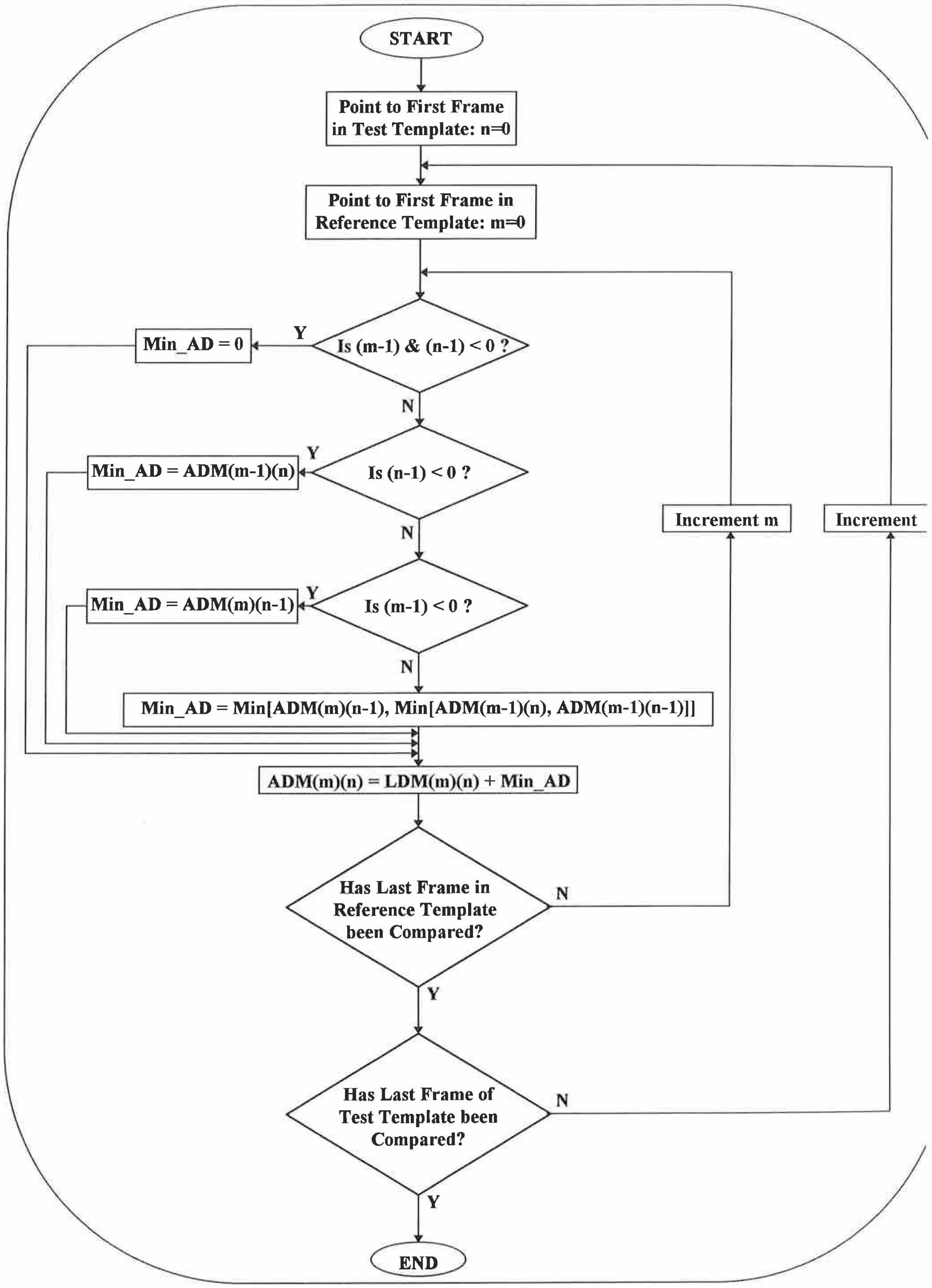

START
Point to First Frame in Test Template: n=0
Point to First Frame in Reference Template: m=0
Is (m-1) & (n-1) < 0 ?
Y
Min_AD = 0
N
Is (n-1) < 0 ?
Y
Min_AD = ADM(m-1)(n)
N
Is (m-1) < 0 ?
Y
Min_AD = ADM(m)(n-1)
N
Min_AD = Min[ADM(m)(n-1), Min[ADM(m-1)(n), ADM(m-1)(n-1)]]
ADM(m)(n) = LDM(m)(n) + Min_AD
Has Last Frame in Reference Template been Compared?
N
Increment m
Y
Has Last Frame of Test Template been Compared?
N
Increment
Y
END

Figure 5.10: Flowchart describing the calculation of the Accumulated Distance Matrix (ADM) in DTW.

5.5 Decision Rule.

After the test template has been compared to all the reference templates using DTW, the next step is to decide which reference template best fits the test signal. The reference template having the smallest minimum path is the one with the highest probability of matching the test signal. The *Nearest Neighbour (NN) Rule* is based on this idea, where $\mathcal{D}_{min\_i}$ (the minimum paths associated with the comparison of the ith reference template) are calculated. From these, the template with the smallest $\mathcal{D}_{min}$ value is chosen as the best fit to the unknown word.

A more robust decision is the *K-Nearest Neighbour (KNN) Rule*. In this case, the K minimum $\mathcal{D}_{min}$ values are extracted from each cluster of words and averaged to give a set of L $\bar{D}_{min}$ values, where L is the vocabulary size. The reference template with the smallest $\bar{D}_{min}$ value is then chosen to be the closest match to the unknown utterance. This technique was chosen for the system due to its ability to reduce the chances of selecting an outlier. In this case, K was given a value of 3 (i.e. The 3NN rule was implemented); 3 was deemed the most suitable K-value for a cluster size of 5. This decision rule is explained by the flowchart in Figure 5.11.

5.6 Success Rates Resulting from HOC Analysis of Gr I Vocabulary Set.

Initially, the system was trained for a vocabulary size of 5, using the Gr I set, to ensure the recognition software was functioning correctly. As can be observed in Table 5.2 extremely high recognition success rates were achieved from the system. These results, however, may only be used to suggest the potential of the HOC technique as a speech recognition system. They cannot be considered as valid success rates, as the size of the vocabulary set was too small, unless of course, the system is only required to recognise five words. However, most very small vocabulary speech recognition systems require the recognition of ~15 words. Of course, as the vocabulary size increases, the % success rates for this system will decrease.

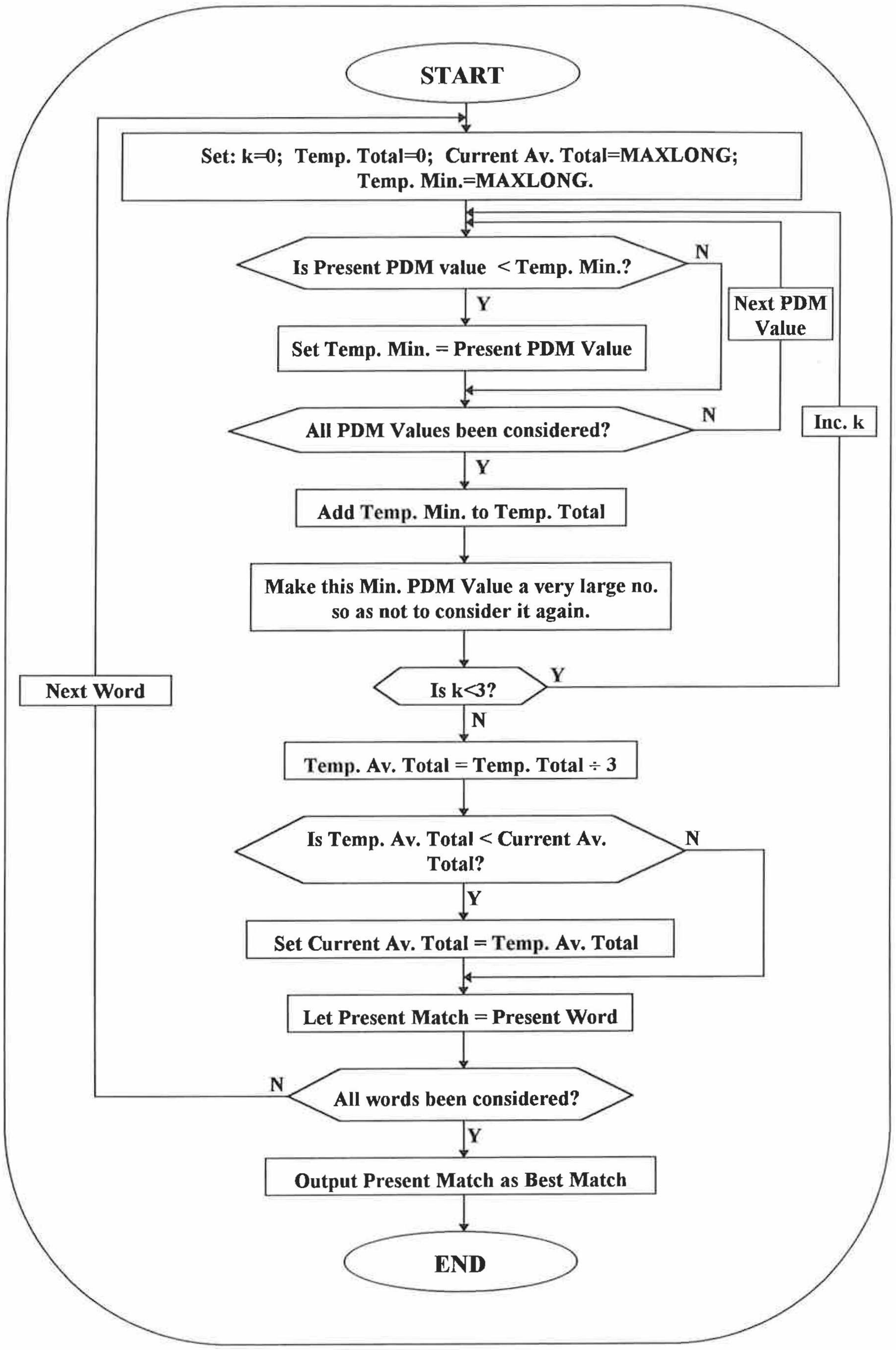

START
Set: k=0; Temp. Total=0; Current Av. Total=MAXLONG; Temp. Min.=MAXLONG.
Is Present PDM value < Temp. Min.?
N
Y
Set Temp. Min. = Present PDM Value
Next PDM Value
All PDM Values been considered?
N
Inc. k
Y
Add Temp. Min. to Temp. Total
Make this Min. PDM Value a very large no. so as not to consider it again.
Next Word
Is k<3?
Y
N
Temp. Av. Total = Temp. Total ÷ 3
Is Temp. Av. Total < Current Av. Total?
N
Y
Set Current Av. Total = Temp. Av. Total
Let Present Match = Present Word
N
All words been considered?
Y
Output Present Match as Best Match
END

Figure 5.11: Flowchart describing the operation of the K-Nearest Neighbour Rule as a decision rule.

Word	% Success Rate
Go	100
Stop	100
Left	91
Right	100
Reverse	100

Table 5.2: The % success rates for the Gr I vocabulary set.

5.7 Success Rates Resulting from HOC Analysis of Gr II Vocabulary Set.

In order to achieve valid % success rates, the vocabulary size was increased to 14 words in the form of the Gr II vocabulary set. Keeping to the theme of a control vocabulary, some similarly sounding words were deliberately selected to increase the difficulty of recognition (e.g. *No* vs. *Go*, *Start* vs. *Stop*). Typical CHOC plots for these words are shown in Figure 5.12.

The system was retrained for a vocabulary size of 14 with these words and using *Wordrec.m* and *Dtwclus6.c* a new set of % success rates was achieved. These success rates (shown in Table 5.3) were not as attractive as those for the Gr I set, yet were still extremely satisfactory.

A '*Confusion Matrix*' was established to explain between which words the system found difficult to distinguish. This matrix is shown in Table 5.4 and consists the fractional confusion between the different utterances.

Word	% Success Rate
Go	100
Stop	55
Left	91
Right	100
Reverse	91
Cancel	100
On	100
Off	100
Yes	91
No	100
Fast	100
Slow	100
Start	100
End	100

Table 5.3: The % success rates for the Gr II vocabulary set.

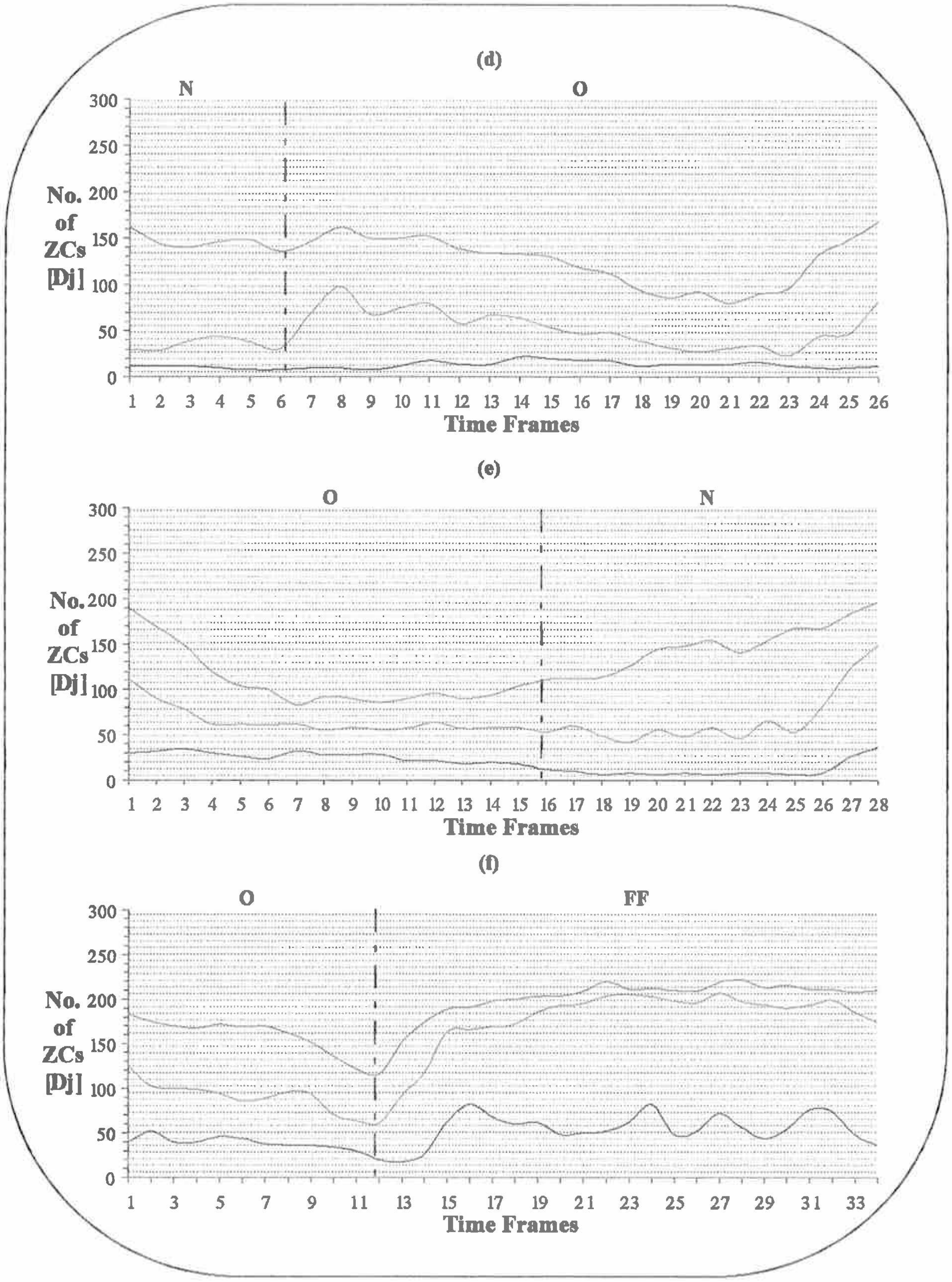


Figure 5.12 (Contd.): CHOC plots for Gr II vocabulary set with approximate phoneme segments: (d) No, (e) On, (f) Off.

Over➔

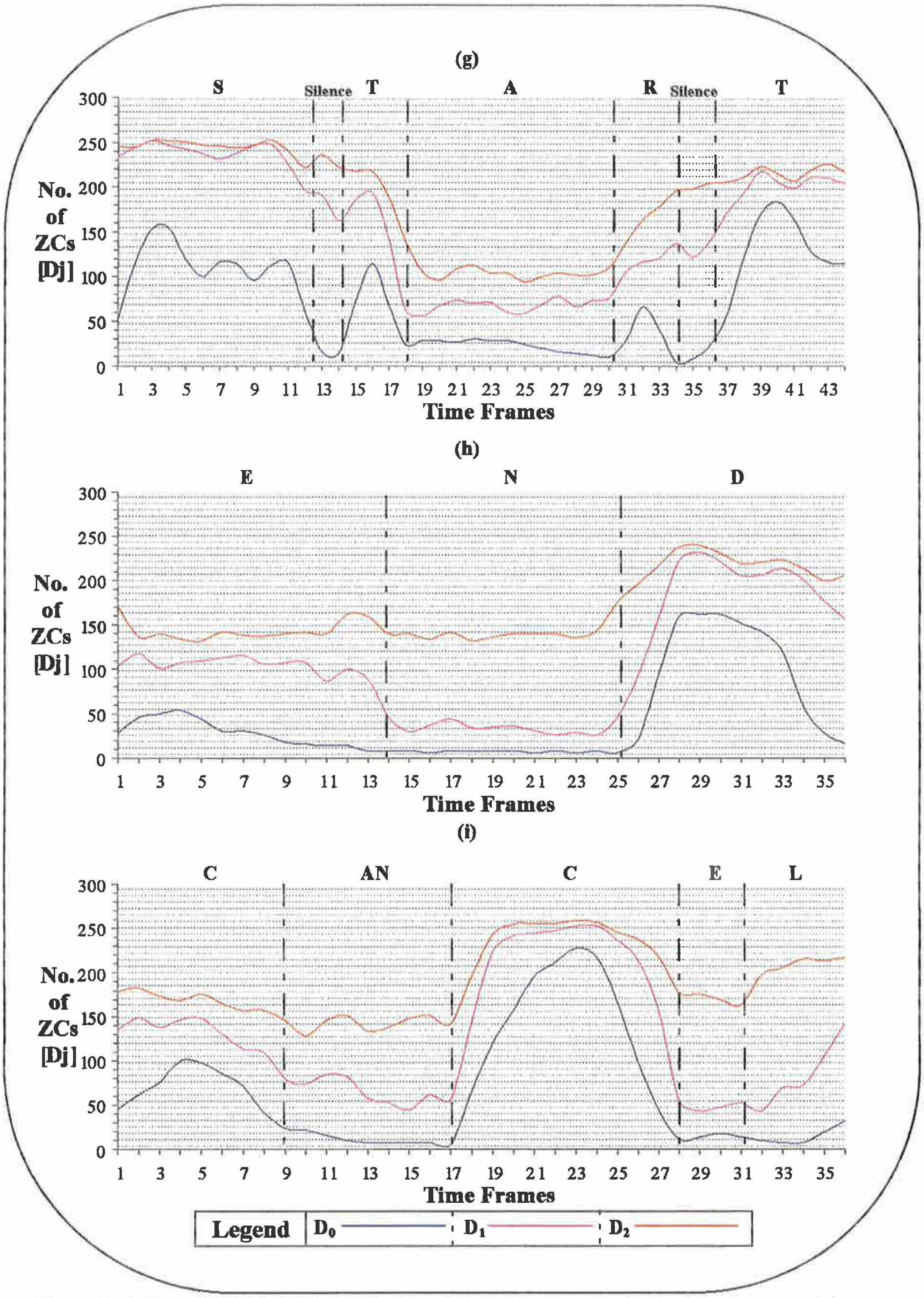


Figure 5.12 (Contd.): CHOC plots for Gr II vocabulary set with approximate phoneme segments: (g) Start, (h) End, (i) Cancel.

Confusion between the utterances of *Stop* and *Start* is very apparent with *Stop* being recognised as *Start* 45% of the time. This is the only significant mis-recognition present in this vocabulary set and is probably due to the identical beginning and similar middle and ending phonemes in each word. Reference to the corresponding CHOC plots will clarify this similarity.

	Go	Stop	Left	Right	Reverse	Cancel	On	Off	Yes	No	Fast	Slow	Start	End
Go	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Stop	0	0.55	0	0	0	0	0	0	0	0	0	0	0.45	0
Left	0	0	0.91	0.09	0	0	0	0	0	0	0	0	0	0
Right	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Reverse	0	0	0	0.09	0.91	0	0	0	0	0	0	0	0	0
Cancel	0	0	0	0	0	1	0	0	0	0	0	0	0	0
On	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Off	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Yes	0	0	0	0	0	0	0	0	0.91	0	0	0	0.09	0
No	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Fast	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Slow	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Start	0	0	0	0	0	0	0	0	0	0	0	0	1	0
End	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 5.4: The *Confusion Matrix* for the Gr II vocabulary set. This may be interpreted as the i^{th} row was recognised as the j^{th} column n number of times.

Henceforth, HOC analysis has now shown to be successful in recognising and distinguishing words in a small vocabulary set with very attractive success rates.

5.8 Testing the System's Vulnerability to Intonation Changes.

At this stage the system has proven to operate successfully with the user speaking in a clear and orderly toned voice. However, how would the system cope if the speaker was to utter the command word in a totally different tone of voice? This phenomenon was studied in relation to the HOC technique using the Gr V vocabulary set. A recording of the word *Stop* was made in an orderly mood and then in an *insistently cranky mood* (this consisted of a dramatic change in intonation). HOC analysis was performed on both words resulting in the CHOC plots shown in Figure 5.13. The similarity between the two plots is still reasonably satisfactory with the main differences occurring in the vowel section and the overall increase in duration. The /s/ and /t/ were unaffected as expected since the

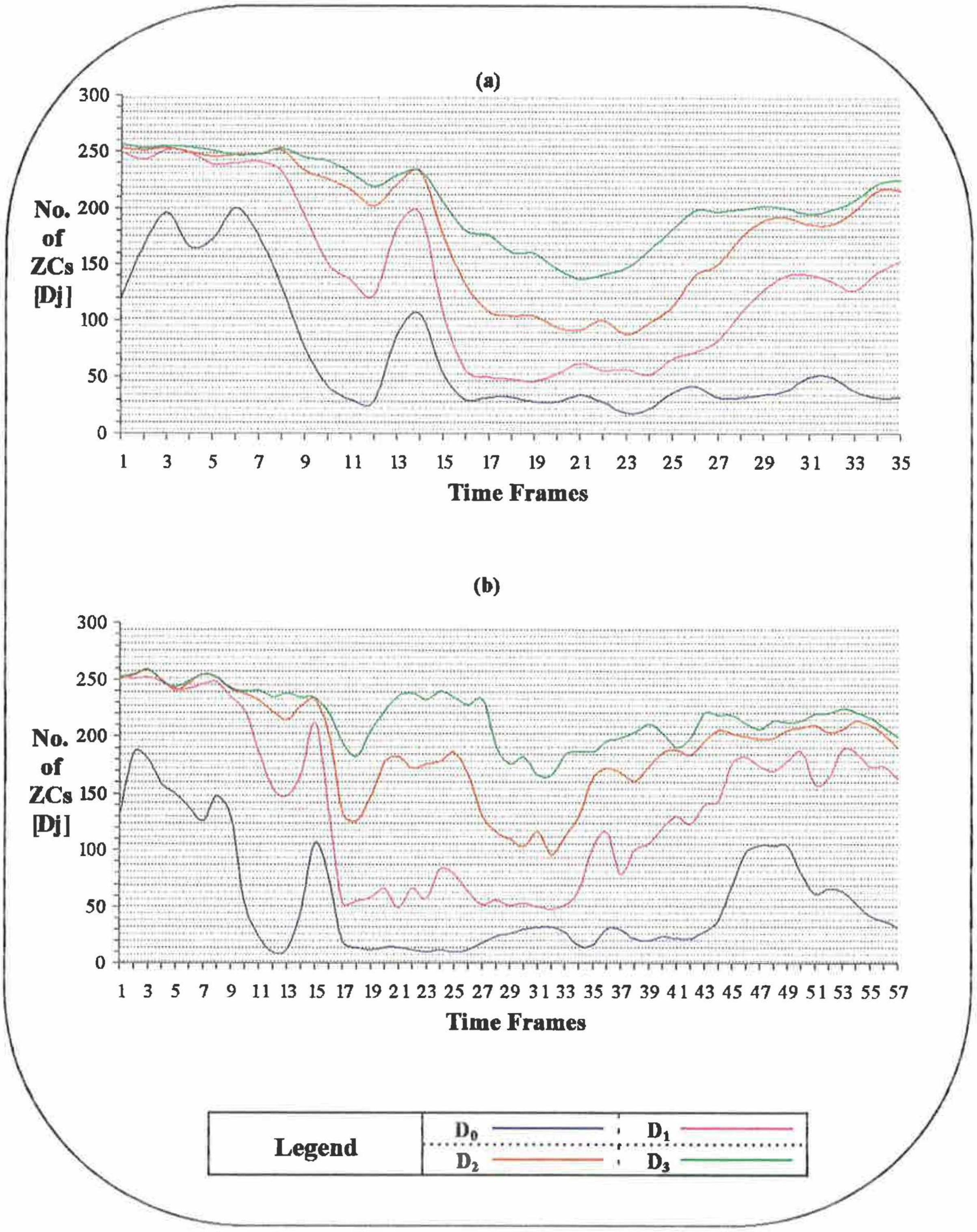


Figure 5.13: Comparison of CHOC plots of utterances with different intonation (a) 'Stop-Orderly' and (b) 'Stop-Cranky'.

intonation change took place towards the end of the utterance. The */o/* breaks into two parts: the initial low frequency and the final rise in pitch. Much ripple is now also present in the */o/* section where the D_2 level has also changed significantly. The */p/* has become more pronounced: this showing even in the D_0 isofil. However, this word was recognised as *Start*, although the $\bar{D}_{min}$ values for both *Stop* and *Start* were extremely close (a difference of ~100). Hence, so long as the user maintains the tone of their voice relatively similar to that which the system was trained, no serious intonation problems should arise.

5.9 Testing the System's Vulnerability to White Noise.

It is in the nature of the differencing filter to amplify higher frequencies, after all it is a high pass filter. Therefore, it was very important to test how vulnerable the system was to noise - the fear being that the HOC technique may amplify any high frequency noise present and so, *drown* the signal information and affect recognition. Initially, tests were performed on a pseudo-speech signal and the '*clean*' HOC data was compared to the '*dirty*' HOC data. A 90ms pseudo-speech signal was synthesised in Matlab by summing three sine waves, the frequencies and amplitudes of which were obtained from a set of formant frequencies and relative amplitudes after Ainsworth (1974) [46]. The signal data chosen was that for the phoneme */u/*:

f_1 = 250Hz	A_1 = 51dB
f_2 = 880Hz	A_2 = 38dB
f_3 = 2080Hz	A_3 = 17dB

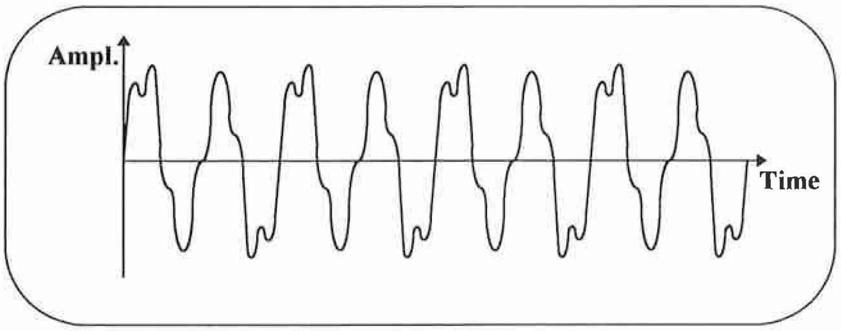


Figure 5.14: The synthesised /u/ sound using the first three formants and their relative amplitudes.

An m-file (*Sndsynt.m* - App. G) was written to synthesise any sound given the first three formant frequencies, their relative amplitudes and the sampling frequency (usually 11kHz). This synthesised sound may be played back using the Matlab command '*sound*'. Using Hocalg.m the pure /*u*/ signal (shown in Figure 5.14) was analysed up to the third filter order resulting in the following HOC data:

Time Frame\Filter Order	0	1	2	3
1	14	31	52	98
2	16	31	56	93
3	16	32	60	102
4	16	32	56	101
5	16	31	54	96

Table 5.5: HOC data for 90ms of '*clean*' synthesised /*u*/. The figures indicate the number of zero-crossings in each time frame corresponding to each filter order.

Next, a *white noise* signal was generated using the '*rand*' function in Matlab, which was then added to the synthesised sound. The m-file *Addnoise.m* (App. G) was written to generate and add this noise signal.

HOC analysis was now performed on the '*dirty*' signal with S/N ratios of 40dB, 30dB, 20dB and 10dB. An error matrix was established for each set of data to describe the severity of the noise (described in Table 5.6).

These results suggest that the system is quite immune to white noise levels with a S/N ratio as low as 20dB only if the maximum filter order is less than three.

5.10 Testing the System's Vulnerability to Pink Noise.

If this system were to be implemented in hardware/μprocessor an anti-aliasing filter would be present for the usual reasons. This filter would have a cut-off frequency of some value around half the sampling frequency (5.5kHz in this case). Thus, it makes sense to test the system against '*Bandlimited White Noise*' or '*Pink Noise*' rather than white noise, as any ambient noise would be filtered to some extent by the anti-aliasing filter.

An m-file (*Addpkns.m* - App. G) was written to achieve the addition of bandlimited white noise and the results in Table 5.7 were obtained as before.

S/N	HOC Data				Error Matrix			
40	14	31	52	98	0	0	0	0
	16	31	56	93	0	0	0	0
	16	32	58	106	0	0	-2	4
	16	32	56	101	0	0	0	0
	16	31	54	96	0	0	0	0
30	14	31	52	96	0	0	0	-2
	16	31	58	95	0	0	2	2
	16	32	56	106	0	0	4	4
	16	32	58	103	0	0	2	2
	16	31	54	100	0	0	0	4
20	14	33	66	136	0	2	14	38
	16	31	70	127	0	0	14	34
	16	34	74	132	0	2	14	30
	16	32	70	135	0	0	14	34
	16	33	60	120	0	2	16	24
10	14	65	190	253	0	34	138	155
	16	57	192	238	0	26	136	145
	18	70	194	246	2	38	134	144
	18	74	196	259	2	42	140	158
	16	77	198	248	0	46	144	152

Table 5.6: HOC data and error matrices for different levels of *white noise* superimposed on a synthesised /u/ sound.

S/N	HOC Data				Error Matrix			
40	14	31	52	98	0	0	0	0
	16	31	56	93	0	0	0	0
	16	32	60	102	0	0	0	0
	16	32	56	101	0	0	0	0
	16	31	54	96	0	0	0	0
30	14	31	52	98	0	0	0	0
	16	31	56	97	0	0	0	4
	16	32	60	104	0	0	0	2
	16	32	56	101	0	0	0	0
	16	31	54	96	0	0	0	0
20	14	33	52	98	0	2	0	0
	16	33	58	99	0	2	2	6
	16	34	62	94	0	2	2	-8
	16	30	58	95	0	-2	-2	-6
	16	33	54	98	0	2	0	2
10	14	33	72	124	0	2	20	26
	16	37	82	119	0	6	26	26
	16	38	86	128	0	6	26	26
	16	36	84	131	0	4	28	30
	16	39	70	118	0	7	16	22

Table 5.7: HOC data and error matrices for different levels of *bandlimited white noise* superimposed on a synthesised /u/ sound.

These results demonstrate how the simple addition of a low pass filter can improve the robustness of the system by increasing its tolerance to noise. A S/N ratio as low as 10dB still yields HOC data with a such a low distortion factor that the probability of a mis-recognition would be minimal.

This low pass filter may either be in the form of an anti-aliasing filter or a filter at a later stage in the system (yet before the differencing stage).

5.11 Noise - Not just a Matter of Filtering.

The direct effects of noise on the system (i.e. the differencing filter amplifying high frequency noise and so dramatically altering the zero-crossing information) is not the only problem when the system is used in a noisy environment. Research carried out by Lombard [76] in 1911 discovered indirect effects on a speech signal due to noise.

Lombard noted that a speaker's voice systematically changes when the speaker is subjected to a noisy environment. This dynamic relationship between speech and hearing is known as the *Lombard Effect.* The indirect effects of noise on a a speech signal are described in Appendix D5.

2W

As this appendix explains, these effects should be taken into consideration when training the system and teaching the user how to use it (hence the need to test the system's vulnerability to intonation changes - Section 5.8).

5.12 The System's Handling of Real Noise.

To examine how well the system can manage with actual ambient noise, (without the Lombard Effect), two utterances were analysed as described by the Gr VI vocabulary set. The first was a '*clean*' utterance of the word '*Stop*' whose corresponding CHOC plot is shown in Figure 5.15(a). To create a '*dirty*' counterpart, a section of loud music† was superimposed on the '*clean*' utterance to allow a S/N ratio of 5dB. Its corresponding CHOC plot is shown in Figure 5.15(b). In the */s/* segment, only the D_0 isofil was affected by the noise. The */t/* remained unaffected. A dramatic increase of levels in the isofils for the */o/* section (frames 16-32) is apparent, while D_1-D_3 remain unaffected. However, a large ripple component

† A section of 'We're So Pretty' by the Sex Pistols was used. The short duration (~700ms), however, would have captured a certain band of frequencies depending on the music at that instant, thus enhancing certain parts of the utterance more than others.

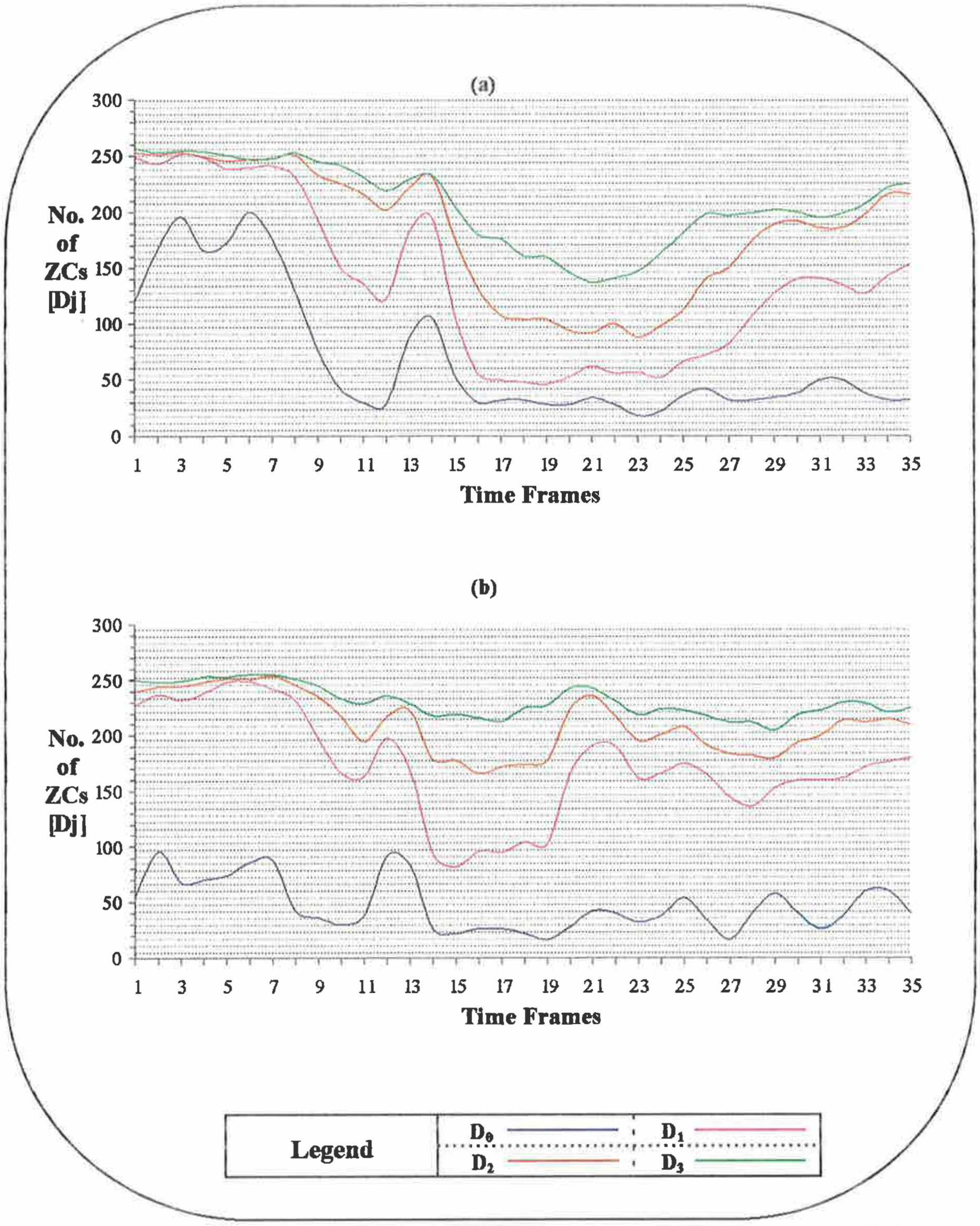


Figure 5.15: Comparison of CHOC plots of the utterance 'Stop' (a) without and (b) with an added noise component with S/N ratio of 5dB.

present in the entire D_0 isofil appears to be the result of a presence of low frequency background noise. This utterance was actually recognise as '*Left*' due to the overall signal distortion.

5.13 Effects of Non-Linear Phase Shift on the System.

Phase shift occurs in a signal when the components are shifted in time nT seconds (equivalent to multiplying by e^{-nTs} in the frequency domain or by z^{-n} in the z-domain). Ideally, all phase shift should be *linear* (i.e. all the signal components are shifted equally) as this is easy to correct. However, in reality the phase nature of certain processes[‡] gives rise to different components being shifted to a different degree. This phenomenon is known as *Non-Linear Phase Shift* and is illustrated in Figure 5.16. The signal in Figure 5.16(a) comprises the sum of two sinusoids. It is clear that by applying a phase change to the higher frequency *only*, the resultant waveform becomes distorted (Figure 5.16(b)).

This phenomenon often proves to be a problem in certain signal analysis techniques where analysis of one signal yields very different results to that of a similar signal due to subjection to different processes prior to analysis. For example, if the speech recognition system was trained in a controlled environment or with a specific pre-filtering module and it was required to be used over a telephone line or where the exact pre-filtering block was not available, there would be no guarantee that these added processes would affect the phase in the same manner as the original system. Hence, the phase would be affected non-linearly and this may upset the results from the system. The need for designing *phase compensators*, for within each environment the recognition system is to operate, would be eliminated if the system could be proven to be relatively insensitive to such phase changes.

Hence, it was necessary to test how non-linear phase changes would affect the HOC data produced during HOC analysis.

[‡] Non-linear phases are often present in analogue transmission lines, telephone networks, etc.

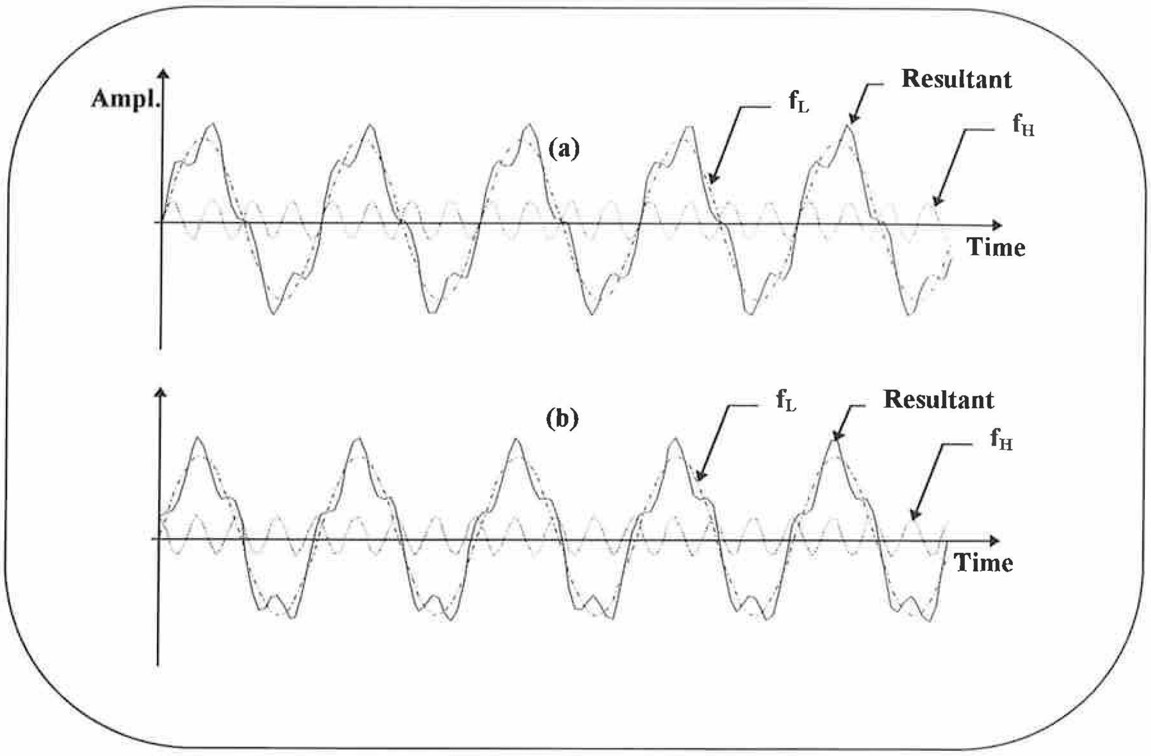


Figure 5.16: The effect of non-linear phase change on a signal comprising a high and low frequency component.

The m-file *Sndsynps.m* (Appendix G) was written to synthesise a sound from its first three formants allowing a phase shift to be added to any of these frequencies. The /*u*/ sound was synthesised again, initially with no phase changes and HOC analysis was performed on the signal yielding the data in Table 5.5[†]. Next each formant was shifted in turn by 90° applying HOC analysis each time. Table 5.8 shows the resulting HOC data and error matrices after the non-linear phase changes. As expected, no difference was detected by the human ear in the sound after the phase change was applied.

From these results, it can be seen that non-linear phase changes will affect HOC data but only very slightly. Such satisfactory results would suggest that the HOC system is relatively insensitive to non-linear phase changes and would create no serious problems in signal recognition.

[†] Refer to Section 5.9.

Frequency out of phase by 90°	HOC Data				Error Matrix			
f_1	16	32	58	103	2	1	6	5
	14	31	58	102	-2	0	2	9
	16	31	56	97	0	-1	-4	-5
	16	34	50	98	0	2	-6	-3
	16	28	54	95	0	-3	0	-1
f_2	14	31	52	112	0	0	0	4
	16	35	54	110	0	4	-2	7
	16	32	50	106	0	0	-10	4
	16	32	54	105	0	0	-2	4
	16	31	54	107	0	0	0	11
f_3	14	29	52	105	0	-2	0	7
	16	31	54	101	0	0	-2	8
	16	30	54	102	0	-2	-6	0
	16	32	54	109	0	0	-2	8
	16	33	54	110	0	2	0	14

Table 5.8: HOC data and error matrices corresponding to different non-linear phase changes applied to a synthesised /*u*/ sound.

5.14 The Effects of Non-Linear Phase Change on Real Speech Data.

To test how the HOC analysis routine was affected by a non-linear phase change to a real signal, the syllable /*ka*/ from the utterance '*Can*' (obtained from the Sheffield Signals [17]) was analysed by *Hocalg.m* yielding the HOC data in Table 5.9. The phase response of this syllable is shown in Figure 5.17(a).

Time\Order	**0**	**1**	**2**	**3**
1	10	120	156	201
2	20	57	134	179
3	16	57	134	183
4	6	87	144	196
5	6	71	116	179
6	14	57	84	157
7	18	43	98	153
8	10	23	90	153
9	8	24	82	148
10	8	28	70	138
11	12	30	46	133

Table 5.9: HOC data for the */ka/* syllable from the utterance '*Can*'.

The m-file *Chgphs.m* was written to impose a non-linear phase change on the speech signal. A delay term of *Cos(10t/l)+Sin(2t/l)*† was used to achieve the extreme non-linear phase response shown in Figure 5.17(b). The magnitude remained at a constant value of 1 to ensure all effects were merely from the phase change.

HOC Data				**Error Matrix**			
16	130	178	203	6	10	22	2
14	46	128	166	-6	-9	-6	-13
10	57	110	166	-6	0	-24	-17
10	73	126	166	4	-14	-18	-30
6	67	106	153	0	-4	-10	-26
12	57	82	139	-2	0	-2	-18
16	43	78	133	-2	0	-20	-20
8	24	72	134	2	1	-18	-19
10	20	74	124	-2	4	-8	-24
12	28	64	101	4	0	-6	-37
10	27	48	106	-2	-3	2	-27

Table 5.10: HOC data and error matrix for the syllable */ka/* after the application of the non-linear phase change: *Cos(10t/l)+Sin(2t/l)*.

† That is, a non-linear phase of $e^{Cos(10t/l)+Sin(2t/l)}$ was applied to the speech signal in the form of a unity gain filter.

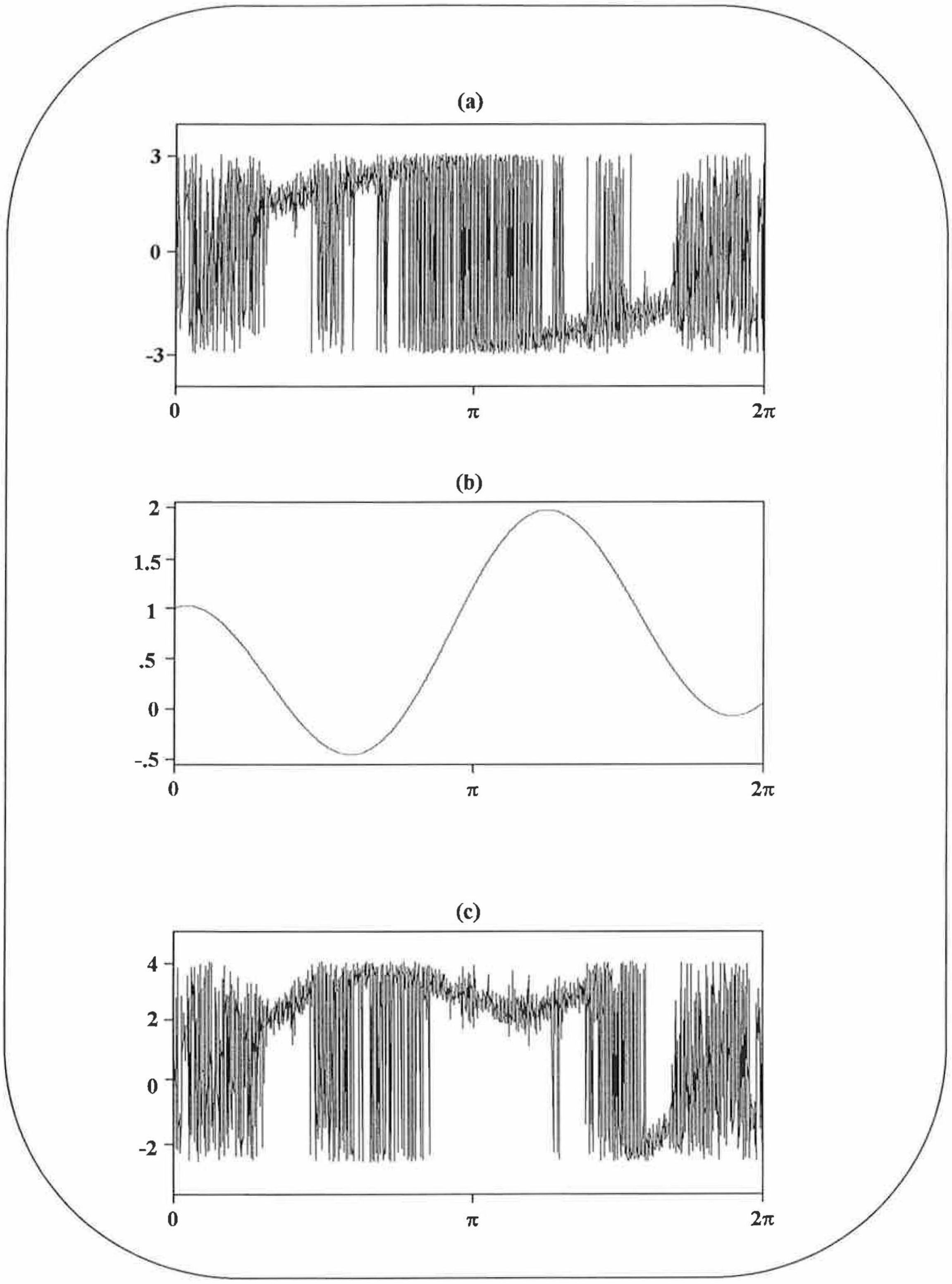


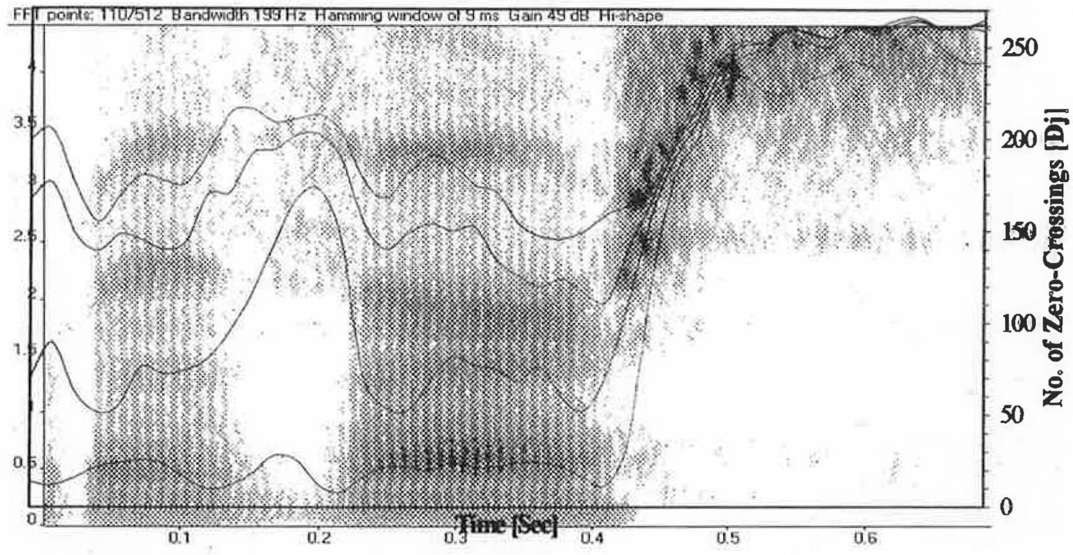
Figure 5.17: Phase plots of (a) */ka/* before any non-linear phase change, (b) an extremely non-linear phase (*Cos(10t/l)+Sin(2t/l)*), (c) */ka/* after this non-linear phase change.

The phase response of the */ka/* sound altered to that in Figure 5.17(c) as a result of the phase change. This *new* sound, although sounding the same to the ear, was again analysed by *Hocalg2.m.* The resulting HOC data and error matrix may be seen in Table 5.10. It is clear that although this non-linear phase change *did* affect the HOC data, it did not distort the data to such an extent that the word would not be recognised (even with such an extreme phase change). Hence, it may be concluded that the CHOC recognition system is practically insensitive to non-linear phase changes.

5.15 Comparison to Spectrogram - Formant Tracking Nature of CHOC Plots.

A spectrogram of the speech signals '*Reverse*' and '*Stop*' were obtained in order to demonstrate any comparisons between this TFR and CHOC plots. The spectrograms are shown in Figure 5.18 with the corresponding CHOC plots superimposed on top. It is clear that the isofils in the CHOC plots display similar time-frequency information as the spectrograms. Where the spectrograms show high frequency content (e.g. the /*s*/ in *Reverse*), the isofils tend to follow. In the higher frequencies, the more powerful the component (i.e. the darker spots in the spectrogram) the closer the isofils. In the lower frequencies (the vowels) the isofil tend to fall *near* and sometimes *on* the more powerful components, thus demonstrating a *formant tracking nature*. This is not merely a coincidence; the *Dominant Frequency Principle*, as discussed in Section 3.2.1, states clearly that HOC analysis results in normalised zero-crossing counts that tend to fall on or near the most dominant frequency present (in this case the formants). As previously stated, when isofils are evenly spread throughout the CHOC plane the nature of the signal is noisy and broadband; this is demonstrated in the /*p*/ of *Stop*. The voiced and unvoiced nature of the phoneme /*v*/ may be seen in the utterance *Reverse*. The isofil D_0 tracks the lower frequency in the spectrogram, while D_1 jumps up to the higher frequency components (not detecting any intermediate frequencies) as does the spectrogram. There is a slight timing difference between the two techniques in the case of /*t*/ in *Stop*; however, this may only be a consequence of different Hamming window durations (30ms for CHOC and 9ms for the Spectrogram).

(a)



(b)

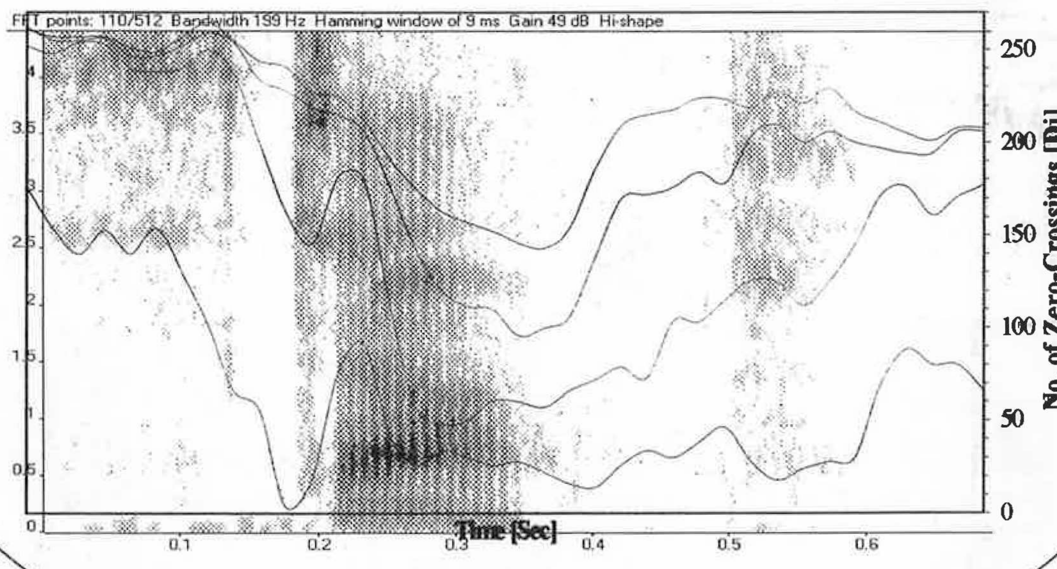


Figure 5.18: Comparison of CHOC plots and spectrogram for the utterances of (a)*Reverse* and (b) *Stop*.

Chapter 6

6. Concluding Remarks and Way Forward.

The suitability (or insuitability) of both HOC analysis and the SIFE system in the development of a low cost, high quality speech recognition system has now been illustrated.

The SIFE system, although initially appearing to possess considerable potential [12], was shown to be of little use as a speech recognition system. The output waveforms proved to be too inconsistent, with the system producing unreliably repeatable SIFE signals displaying weak signal discriminatory powers. While very successful in its original implementation, (a voice training instrument [14]) the SIFE technique proved unsuitable in the development of a speech recognition system.

The HOC system, on the other hand, proved to be a very suitable signal analysis technique. Its output signals (in the form of CHOC plots) were remarkably repeatable and displayed an extremely significant capacity in discriminating signals. When tested, this system produced very satisfactory % success rates (in the order of 90%) even with a larger vocabulary size. The only problematic aspects were speed (the HOC analysis routine ran in ~2.5 minutes on a 486 (33MHz))[†] and the system's tolerance to high frequency noise due to the differencing filter stages. Nevertheless, this system possesses great potential at being further developed as a speech recognition system.

6.1 HOC Versus SIFE.

As have been noted in previous sections in this thesis, there exist certain advantages of the HOC system over (among other zero-crossing techniques) the SIFE. The following is a description of some of those advantages:

- The HOC technique tends to enhance voiced fricatives such as */v/* in '*Reverse*', whereas most zero-crossing analysis systems show little differences between these and vowels.
- The same may be said about */p/* in '*Stop*', which demonstrates a very similar zero-crossing rate to that of the preceding vowel; however, even after only one filtering stage the */p/* becomes more pronounced.

[†] This was reduced to ~1sec/word using the TMS320C26, which suggests that the possibility of real-time implementation is promising.

- Periods of silence before plosives (which are characteristic of this category of speech) are clearly defined by the HOC method allowing an extra parameter of recognition[†].
- The SIFE also demonstrates this but to a lesser degree; however many zero-crossing systems do not persevere this characteristic during analysis.
- The fine quality of output data from the HOC technique allows the use of a number of decision/recognition techniques:
    - ♦ Dynamic Time Warping with Nearest Neighbour Rule.
    - ♦ Phonetic Categorisation.
    - ♦ Voiced-Unvoiced Decision.
- This high quality signal output has allowed % success rates to be determined, which in turn have proven to be very satisfactory (91-100% with the exception of the utterance '*Stop*' - 55%).
- The poorer quality of output data from the SIFE technique suggests the Voiced-Unvoiced Decision would be the only suitable recognition technique.
- The HOC technique requires no post-filtering (unlike the SIFE), and produces a more reliable and repeatable output, therefore suggesting to be a more attractive system.
- The HOC technique tends to take '*non-zero-crossing signal variation*' into consideration unlike the SIFE which can only detect a zero-crossing. For example, consider the signals shown in Figures 6.1 and 6.2. A SIFE analysis of the waveform in Figure 6.1 would result in the detection of the lower frequency ($1/T_2$) only, whereas the waveform actually consists of the summation of two frequencies ($1/T_1$ and $1/T_2$). However, by repeated filtering, the HOC method will eventually detect frequency $1/T_1$. A SIFE analysis of the waveform in Figure 6.2 would result in the recognition of a sinusoidal signal of frequency $f=1/T_p$, but would totally ignore the fact that the waveform also consists of a ripple signal (p_1-p_5 and t_1-t_5). However, after HOC analysis, up to only D_2, all of the peaks and troughs (p_1-p_5,t_1-t_5) and all points of inflection present are

[†] Silence on a CHOC plot is portrayed as isofils evenly distributed in the CHOC plane due to the broadband nature of background noise.

considered in the recognition of the signal. Hence, it can be seen that the HOC method extracts higher quality information from the speech signal than the SIFE method.

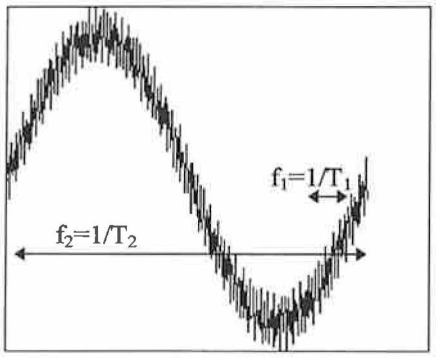


Figure 6.1: Summation of two sinusoids.

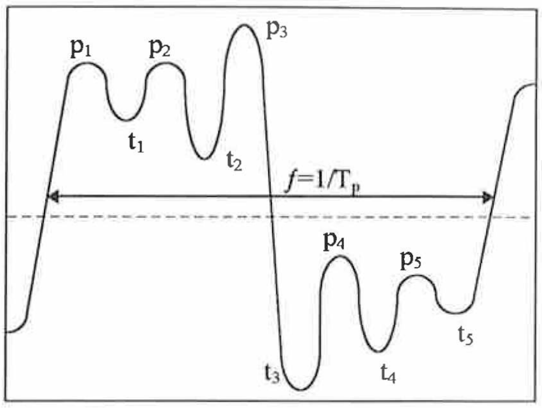


Figure 6.2: A pseudo-speech signal.

6.2 Hardware Implementation.

The HOC system may be implemented in hardware in either of two ways:

1. Appropriate zero-crossing detector circuitry and switched capacitor filters to operate as differencing filters.
2. Application of software routine on a microprocessor/DSP chip such as the TMS320C26 [65] (a software routine, called *Hocana.asm*, has been written to apply HOC analysis to any signal using this DSP chip.

Although slightly more expensive, the latter may be faster, less complicated and more reliable due to the low number of components (only the DSP board and the software routine). The only significant problem with this machine code program is that no appropriate triggering routine has been written for the system. Presently, the analysis routine may be activated by any sound; it has no distinction between background sounds and actual speech signals as the DSP chip is constantly polling for a signal. The designing of such a triggering routine and further development of the system could prove to be suitable work for a short-term project in the future.

6.3 Noise Tolerance.

As previously stated, the high pass filtering nature of the differencing stages in HOC analysis has a tendency to amplify high frequency noise. Initially, this proved to be a worrying aspect, but as the noise tests in Sections 5.9-5.12 suggest, the system operates reasonably well so long as the maximum filter order remains below three. However, there is no guarantee that ambient noise levels will remain below that threshold value so as to allow a maximum filter order of two. Henceforth, it would be recommended to develop some type of filtering circuitry to reduce the effect of ambient high frequency noise *without* affecting the high frequency content of unvoiced speech signals. Once again, this could prove to be useful research work in the future.

6.4 Tolerance to Non-linear Phase Changes.

It has been shown how although the HOC system is affected by non-linear phase changes, the consequences are so insignificant to render the system practically insensitive to such signal variations. This is important as it removes any trepidation about the phase nature of any equipment (such as analogue transmission lines, recording equipment, amplifiers, telephone systems, etc.) to be used before the HOC analysis stage. Hence, the use of the system in any phase environment is permitted without the need to retrain the system.

6.5 Speed Considerations.

At present, the HOC analysis program (Hocalg5.m) executes at an average speed of 2.5 minutes per word on Matlab, using a 486 @ 33MHz. Running on Matlab (a Windows [41] driven package), the program is expected to be slow. However, the training and recognition stages (Train.c and Dtwclus6.c) are faster (30 seconds - 1 minute) running on MS-DOS [41], yet still require greater speed before being implemented in a practical application. An assembly program has been written to implement the analysis stage of the HOC system on a TMS320C26 and executes dramatically faster than the Matlab version (~1sec/word). Hence, it would be a good idea to implement the whole recognition system on a chip such as the TMS320C26 to improve processing time.

Another method to increase speed would be to reduce the number of stored reference templates of each word (presently five) which would in turn reduce the number of pattern comparisons to be made for each test signal.

6.6 Limitations on Vocabulary Size.

It has been shown how the HOC system worked almost perfectly as a speech recognition system for a vocabulary size of five. This however could not really validate the system, unless the application only required the recognition of five words. The vocabulary was then extended to fourteen words and the results were extremely satisfactory. The only word that caused trouble was '*Stop*' which seemed to be confused with '*Start*' 45% of the time due to their similarity in phonetic structure. The converse confusion however did not result.

Such high success rates of recognition would suggest that the system could be retrained for anything up to 30 words and still function satisfactorily. Care should be taken, however, in choosing the vocabulary, avoiding words that sound too similar, to reduce the possibility of a mis-recognition.

6.7 Implementation for Continuous Speech.

The success of the HOC technique as an *isolated word* speech recognition system would suggest that an investigation into its ability to recognise *continuous* or *connected speech* or *phrases* would be recommended. The fact that HOC data (in the form of CHOC plots) contains such powerful signal distinguishing capabilities suggests that there would be no problems in using the system in the recognition of phrases (as long as it has been trained with the appropriate phrases). The only problem is that the size of the stored templates would be significantly larger and so, both HOC analysis and pattern matching would be slower, most probably resulting in the need for a more powerful DSP chip.

However, the system could not be expected to operate satisfactorily on connected speech when only trained for each isolated word in the sentence (due to the changing nature of each word when uttered successively). As with other connected speech recognisers, it would be necessary to develop an appropriate *connected speech training routine* if the system was to be used in the recognition of continuous speech.

6.8 Possibility of Speaker Independence.

The intention of this project was to design a *speaker dependent* speech recognition system (one requiring training for each individual user). The system developed is speaker dependent although it may be extended to be more independent. Training the system with utterances of each word made by a number of speakers (preferably with similar accents and pitch) will improve its ability to recognise a greater number of speakers[†]. The clustering technique implemented in the training of this system is particularly suited for this purpose.

6.9 Implementation as Formant Tracking System.

It has been shown how the isofils of a CHOC plot almost track the formants of a speech signal and display similar frequency information to that of a spectrogram. The appropriate study into the nature of the different filters that may be used in HOC analysis (e.g. the use of a more precise HPF instead of a differencing filter or the use of both HPFs and LPFs) may result in the development of a simple yet very effective TFR system or formant tracking system.

6.10 Practical Applications.

The developed HOC system could prove useful in a number of practical applications. The low cost nature and simple mathematical theory behind the system make it desirable product to be implement practically. Among the possible applications is the *motion control for a wheelchair* as described in [12]. This would be of great benefit to quadriplegics, (and other users for whom joystick control is not suitable), allowing full control of their wheelchairs and perhaps reducing the cost of the wheelchair.

The system may also be implemented (with the appropriate interfacing), to allow *cursor/menu control* in a software application such as Windows. This would serve both as a convenience to a user as well as a useful package for a paraplegic computer operator.

[†] As the number of users the system is trained for increases, the more users it will recognise but lower percentage success rates for each individual user will result.

A paraplegic may also benefit from the system being implemented as a *telephone dialling instrument*, where the user would merely be required to activate the system and dictate the phone number. Of course, the receiver would not be in the usual *mobile* form; instead a fixed receiver would be present with a high quality speaker and microphone. This could be used in all telephone systems to reduce the number of moving parts (i.e. the dial/keypad).

Staying with the telephone theme, the system may also be used as a *voice activated telephone information service* (e.g. train timetables).

Finally, the HOC system may be used as a *voice operated remote control system* for household appliances such as televisions, hi-fis, lights, curtains, etc.

Each of these applications would be very suited to the HOC system as they only require small vocabularies of words to be recognised.

6.11 Continuing Liaisons with the Project Sponsor.

The initial sponsorship by ***Ogden Atlantic*** (formerly Logitech Ireland) made towards the Advanced Research Programme (ARP) application (Forbairt) was further increased near the end of the project which was extremely appreciated. These funds helped in expenses, etc. and the extra sponsorship allowed the final work necessary to develop the system to the stage it is at now.

Links with Mr. Willie Brien (Engineering Manager, Ogden Atlantic (Ireland)) have been kept throughout the duration of the project with regular meetings to assess progress, etc. taking place. Mr. Brien and his company are very interested in the system's development and are very satisfied with the progress made to date. (Logitech were hoping to release a speech recognition system onto the market).

The hope is to maintain a liaison with Ogden Atlantic in order to further develop the system in the near future.

Appendix A

A1. Materials.

The following section describes the recording procedures, the equipment used in signal recording and analysis and the various speech signals used in the testing of the speech recognition techniques. A brief description of any processing required by the recorded speech signals before analysis could begin is also presented.

A2. Recording Details.

- All the speech signals mentioned in this thesis were recorded using a Sound Vision 16 Gold sound card [36] in a 486 DX2 66MHz PC, (with the exception of the Sheffield Signals [17]).
- Voices were sampled at 11025Hz (16 bit data).
- Sounds were recorded in a 'quiet' (only the sound of the PC was present) environment.
- A standard unidirectional microphone, positioned approximately 15cm from the source, was used.
- All recorded subjects were adult males except where specified.

A3. Signal Analysis Materials.

- All software simulations of hardware systems were implemented in Simulink Ver. 2.1 [40].
- Any other analysis of signals took place in Matlab Ver. 4.0 [40] with the appropriate m-files being written in Notepad [41].
- Conversion of data from *.wav* format to *.mat* format was achieved using either Goldwave [13] or Notomat [9].

A4. Speech Databases.

The following sets of speech data were used to test the performance of the speech analysis/recognition system described in this thesis:

- Gr I [Basic Control Vocabulary]: *Stop*, *Go*, *Left*, *Right*, *Reverse.*
- Gr II [Advanced Control Vocabulary]: *Stop*, *Go*, *Left*, *Right*, *Reverse*, *Fast*, *Slow*, *On*, *Off*, *Yes*, *No*, *Start*, *End*, *Cancel.*
- Gr III [Vowel Comparison Set]: /æ/, /ɛ/, /ɘv/, /δ/, /i/, /ɛi/, /u/.
  (All prefixed with /k/).
  (Refer to Figure 1.9 for pronunciation).
- Gr IV [Consonant Comparison Set]: /b/, /d/, /g/, /p/, /t/, /k/, *Fin*, *Thin*, *Mow*, *No*, *So*, *Show.*
- Gr V [Intonation Set]: *Stop* (orderly), *Stop* (cranky).
- Gr VI[Noisy Set]: *Stop* (clean), *Stop* (dirty).

A5. Necessary Processing of Recorded Data before Analysis.

- As soon as the speech signals were recorded, they were normalised using the '*normalise*' function in the sound card software. This amplifies the sound to its maximum possible volume without distortion. It improves the definition/resolution of the signal especially in sections of the sound containing low energy (eg. /s/, /f/).
- After converting *.wav* files to *.mat* files using Goldwave [13], the resulting *.mat* files must be transposed and saved once again in Matlab to convert each data matrix into the standard Matlab *.mat* format.

Appendix B

B1. Difficulties with SIFE Hardware.

The SIFE system described in [12] yields a SIFE signal from an utterance received in real-time at a microphone connected to the hardware. This signal is then downloaded to a PC via an *Intel 8255 Programmable Peripheral Interface* [15] where the two parameters (mean square energy and absolute mean) are calculated using a C-program.

In order to improve distinction between different SIFE signals, a number of statistical algorithms were written in C-code. Among these new parameters were *mean, variance, probability density function, area under the SIFE signal.*

After testing the system in [12], it was discovered that it was extremely sensitive to background noise and so very irregular results were achieved. The software was also problematic with the PC crashing on numerous occasions. Hence, it was decided that it was necessary to (i) record the speech signals in a quieter environment (overcoming the need for an expensive microphone) and (ii) rewrite the software so that a more user-friendly and reliable package was available for testing.

B2. Description of SIFESIM Simulink Model.

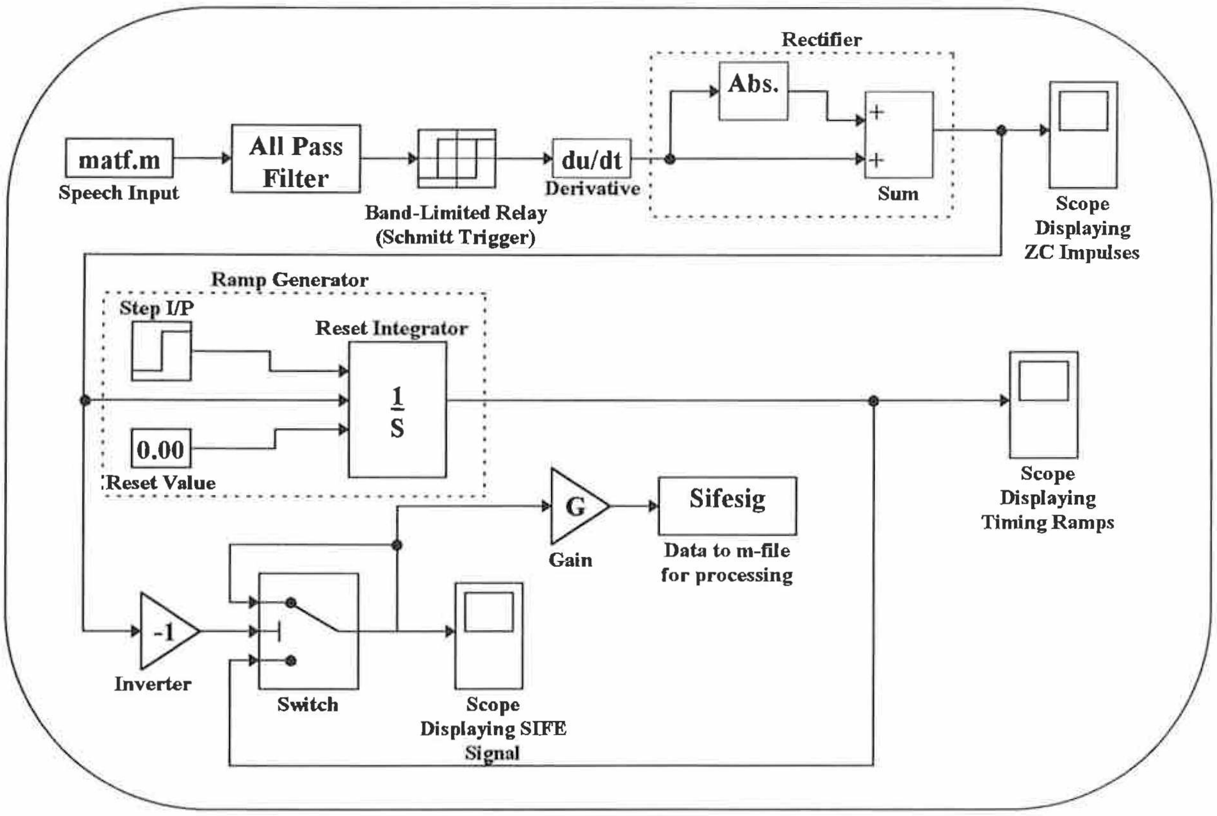
The three section of the SIFESIM model may be described as follows:

- The Zero-Crossing Impulse Stage: The original speech signal is stored in the Matlab file named '*matf.m*'[†]. A *Relay* block mimics the operation of the Schmitt trigger. The *Derivative* block simply differentiates the ICS signal and the combination of the *Absolute* and *Sum* blocks extract only the positive-going zero-crossing impulses. This pulse train may be observed in the appropriate scope.
- The Reset Integrator Stage: A ramp generator (consisting of a *Reset Integrator*, *Step Function*, *Reset Value* and *Reset Trigger*) produces the timing ramps displayed in the corresponding scope. The step function is continually integrated until a zero-crossing (reset trigger) is detected when the output is

[†] Refer to Appendix B3 for signal format.

reset to the reset value. This value had to be set to 0.001 to compensate for the slow resetting of the ramp.

- The Sample & Hold Stage: Initially, a '*Zero-Order Hold*' block was implemented in conjunction with the *switch* block to achieve the sample and hold operation, but it was later observed that these blocks had the same effect on a signal. The switch block was easier controlled and so chosen over the ZOH. The sampling rate was too high to allow accurate resetting of the integrator with this block removed and so, an *All Pass Filter* block with an appropriate sampling frequency (11kHz) was inserted at the initial stage of the system. This allowed the integrator to reset correctly. An *Invertor* block is necessary to cope with the negative triggering of the switch block. The output data is then amplified and stored in a file called '*Sifesig.mat*' for analysis by the C-program.



Rectifier
Abs.
matf.m
Speech Input
All Pass Filter
Band-Limited Relay (Schmitt Trigger)
du/dt
Derivative
Sum
Scope Displaying ZC Impulses
Ramp Generator
Step I/P
Reset Integrator
1/s
0.00
Reset Value
Scope Displaying Timing Ramps
G
Gain
Sifesig
Data to m-file for processing
-1
Inverter
Switch
Scope Displaying SIFE Signal

Figure B1: Simulink block diagram for SIFESIM.

B3. Recording and Formatting of Matf.m.

Five utterances of each word in vocabulary set Gr I were recorded as described in Appendix A4 resulting in *.wav* files of speech data. These were saved in *.raw* format and then converted to *.mat* format using Notomat.c [9] to allow Matlab compatibility.

In order to implement the '*From File*' block in Simulink, this *.mat* format (which is in a single column of data) had to be further converted into a matrix of one *time row* and one *data row*. The following Matlab code was implemented to achieve this:

```
t = [0 : 1/fs : N/fs];        % fs= Sampling Frequency, N=No. of Samples.
load word;                    % word is the utterance to be converted.
wordf = [t ; word(1:N)'];     % wordf is the converted utterance.
Save wordf wordf;             % Save variable to file.
```

The output data is saved in ASCII format to allow compatibility with the C-program using the command:

```
save sifesigc.txt sifesig - ascii;
```

B4. Possible Recognition Routine using SIFE Data.

A simple routine may be:

1. Examine each sample of the SIFE signal and categorise into:

Voiced (V)	if	Sample $\geq X_{Th}$
Unvoiced (U)	if	Sample $< X_{Th}$.

This results in a string of Voiced-Unvoiced sections coded into V and U characters (or 0's and 1's).

2. Remove all consecutive *like-terms* using *Rlktrms.m* leaving a small string of V's and U's describing the voiced-unvoiced nature of the utterance.

3. After each V and U character (or 0 and 1) append the number of times this character was detected. This retains important timing information which may prove necessary in the recognition of the utterance.

3. Compare this string to the template strings (corresponding to the chosen vocabulary) stored in memory. Select the closest match as the recognised word.

Although this recognition routine is easy to implement, it is quite obvious that it would prove an unsuccessful means of speech recognition with the SIFE system for the following reasons:

- There exists poor repeatability in the SIFE signal.
- The discriminatory powers of the SIFE signal have not proven to be satisfactory.
- As described in Section 1.3.2, there is a considerable amount of overlap between voiced and unvoiced zero-crossing data. Hence, the probability of a successful recognition being made using a voiced-unvoiced decision on a poor quality signal is slim.
- The SIFE signal tends to display certain sounds misleadingly. For example, the /v/ in the utterance '*Reverse*' as in Figure 2.6 is portrayed as a purely voiced sound and the high frequency content (the unvoiced component) is neglected (a problem with most zero-crossing analysers).

Appendix C

C1. Description of Diffil.m and Sumfil.m.

The *Pascal* function in Matlab was used to determine the coefficients of each filter stage: the diagonal of the (n+1)th order Pascal matrix yields the coefficients of the nth order summation filter and $(-1)^n$ gives the appropriate signs for the differencing filter coefficients.

eg. 3rd Order Filter $\Rightarrow$ Pascal(4)

$$\begin{matrix} 1 & 1 & 1 & \mathbf{1} \\ 1 & 2 & \mathbf{3} & 4 \\ 1 & \mathbf{3} & 6 & 10 \\ \mathbf{1} & 4 & 10 & 20 \end{matrix}$$

The delay operator was achieved using *Roll.m,* which simply rotates the data sequence n positions to the left or right depending on whether a β^{-n} or β^{n} is required. This significance of this m-file is explained in detail in Section 4.2.1. *Zcount.m* was employed to count the number of zero-crossings after each filter stage and *Hocplot.m* plots a HOC plot corresponding to the signal.

C2. Explanation of the Significance of D_0, D_1, D_2.

- D_0 is simply the number of zero-crossings in the signal.
- D_1 is the number of peaks and troughs in the signal.
- D_2 is the number of points of inflection in the signal.

How is this possible? Well, differencing a signal x_t once yields a new signal $y_t=dx_t/dt$ which is in fact a description of the slope in the original signal. Knowing that the slope at a maximum or minimum is always zero, then a zero-crossing in y_t corresponds to a maximum or minimum x_t. The same holds true when a second difference is applied (ie. if d^2x_t/dt^2 is zero a point of inflection is present). D_3 and higher orders are a little more difficult to understand.

Appendix D

D1. Example 5.1 - Demonstrating How Dynamic Time Warping Operates.

The test signal T={3, 5, 9, 2, 1} is to be mapped onto the reference template R={2, 4, 5, 8, 3, 1} for comparison.

The first step is to create what is called a *Local Distance Matrix (LDM)* which describes the local distances between each sample. This local distance may be calculated as: $d(T_i,R_j) = | T_i - R_j |$ and is termed the *absolute city block distance*. The following LDM results:

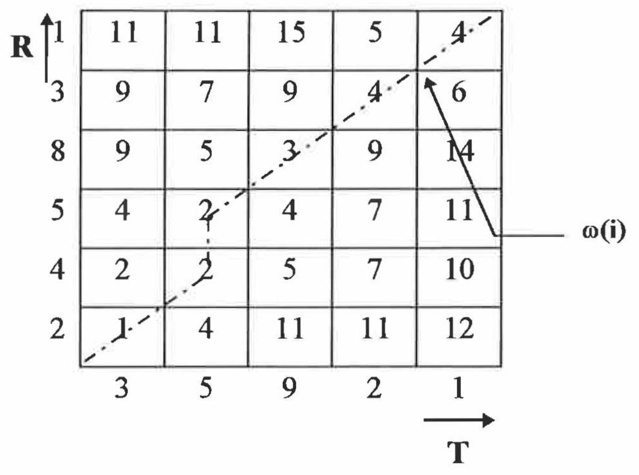
R ↑					
1	2	4	8	1	0
3	0	2	6	1	2
8	5	3	1	6	7
5	2	0	2	3	4
4	1	1	3	2	3
2	1	3	7	0	1
	3	5	9	2	1

T →

Next an *Accumulated Distance Matrix (ADM)* is established using the equation:

$$\mathcal{D}_A(T_i,R_j) = d(T_i,R_j) + \text{Min}[\mathcal{D}_A(T_i,R_{j-1}), \mathcal{D}_A(T_{i-1},R_{j-1}), \mathcal{D}_A(T_{i-1},R_j)]$$

This yields an ADM as follows:



R ↑					
1	11	11	15	5	4
3	9	7	9	4	6
8	9	5	3	9	14
5	4	2	4	7	11
4	2	2	5	7	10
2	1	4	11	11	12
	3	5	9	2	1

T →

The minimum path is then found under certain constraints. Such constraints may include that the path may not be horizontal for more than one time interval (preventing excessive expansion or compression of the time axis) or that the path my be bounded by a parallelogram with sides of slopes 2 and ½ (Itakura constraint). It is important to ensure the path does not have a negative slope at any time (ie.

Time order must be preserved) and so, the mapping function ω(i) is always monotonically increasing. ω(i) is shown in the ADM above and now maps the test and reference templates in the following manner:

i	j = ω(i)
1	1
2	2,3
3	4
4	5
5	6

This may be described by the mapping in Figure D1.

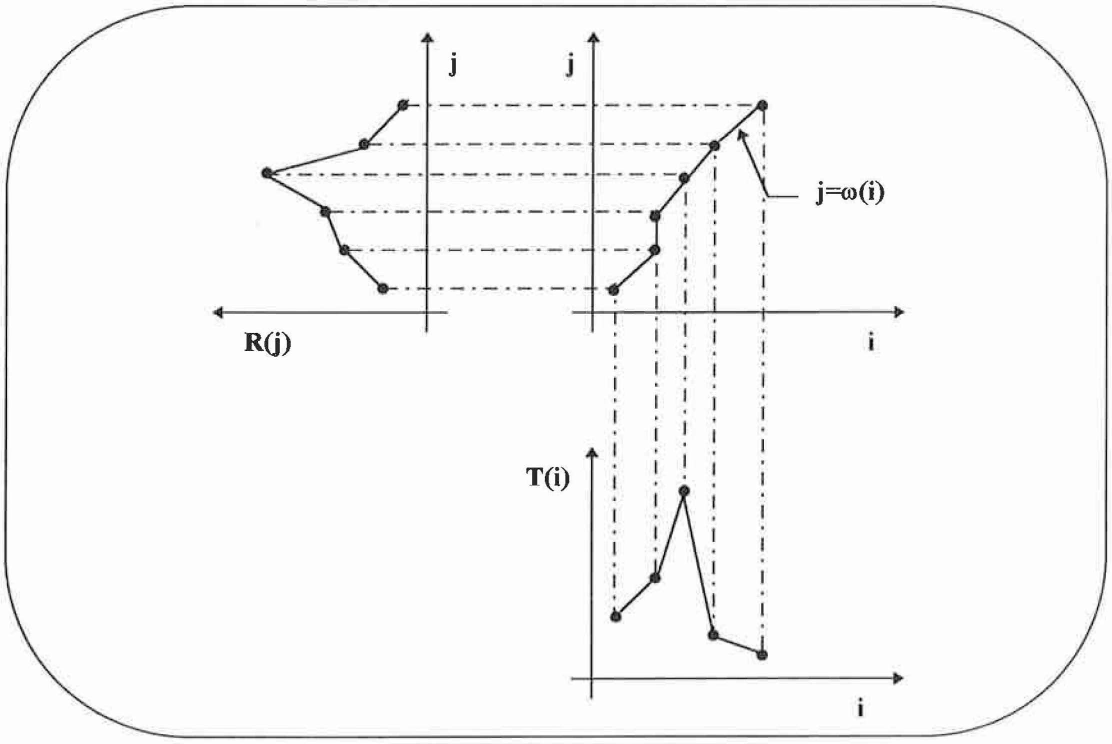


Figure D1: Mapping of Test and Reference Templates after DTW.

D2. Necessary Formatting in the Training Stage.

The program used to train the system was *Train.c* - a slightly modified version of that described in [44]. However, this program was not sufficient in itself due to problems of data type incompatibility and different data string structures. *Trainer.m* was written to overcome these problems by converting the data into the correct type (ie. ASCII to binary) and arranging the data strings into the proper format (ie. Appending the name of the word to the beginning of the file); this newly formatted data is stored in the file called *Train.dat*. *Trainer.m* also runs the executable file of *Train.c* eliminating the need to leave the Matlab environment and enter the Turbo C compiler. *Trainall.m* simply executes *Trainer.m* for every word in the chosen vocabulary set.

Train.c operates as described by the flowchart in Figure 5.6. The data to be trained (ie. the CHOC data in *Train.dat*) is entered into a structure of the form: *[Word_Name].[Word_Length].[CHOC_Data]*. DTW is performed on each of the ten utterances for a particular word resulting in 10×10 minimum paths. The matrix displaying these paths is symmetrical about a diagonal of zero values - a resultant of comparing like utterances. The columns of this matrix are added to give a string of values that describe the similarity between each utterance. The utterances with the five minimum values are chosen as the most similar and stored in memory as reference templates under the filename *Ref.dat*.

D3. Tagging of CHOC Data.

For simplicity the CHOC data was tagged to form a single data string avoiding the need to use multi-dimensional DTW. To achieve this the CHOC data matrix *H* was first transposed in *Hocalg5.m* resulting in the matrix *Htrans* as shown in Figure D2(b)).

$$\text{(a)} \quad \mathbf{H} = \begin{bmatrix} \mathbf{D_0} & \mathbf{D_1} & \mathbf{D_2} \\ \mathbf{D_0} & \mathbf{D_1} & \mathbf{D_2} \\ \mathbf{D_0} & \mathbf{D_1} & \mathbf{D_2} \\ \vdots & \vdots & \vdots \\ \mathbf{D_0} & \mathbf{D_1} & \mathbf{D_2} \end{bmatrix} \qquad \text{(b)} \quad \mathbf{Htrans} = \begin{bmatrix} \mathbf{D_0} & \mathbf{D_0} & \mathbf{D_0} & \dots & \mathbf{D_0} \\ \mathbf{D_1} & \mathbf{D_1} & \mathbf{D_1} & \dots & \mathbf{D_1} \\ \mathbf{D_2} & \mathbf{D_2} & \mathbf{D_2} & \dots & \mathbf{D_2} \end{bmatrix}$$

$$\text{(c)} \quad \mathbf{Htag} = [\, \mathbf{D_0}\ \mathbf{D_0}\ \mathbf{D_0}\ \dots\ \mathbf{D_0}\ \mathbf{D_1}\ \mathbf{D_1}\ \mathbf{D_1}\ \dots\ \mathbf{D_1}\ \mathbf{D_2}\ \mathbf{D_2}\ \mathbf{D_2}\ \dots\ \mathbf{D_2}\,]$$

Figure D2: The three stages taken by *Hocalg5.m* to tag CHOC data.

Finally, each row was *tagged* onto the end of its preceding row to give the data string shown in Figure D2(c). Now, there exist only one pattern, in the form of a tagged CHOC data set, to be matched. These tagged data sets were saved in files of the form: *t_name.dat*. A typical plot of a tagged CHOC data set is shown in Figure D3.

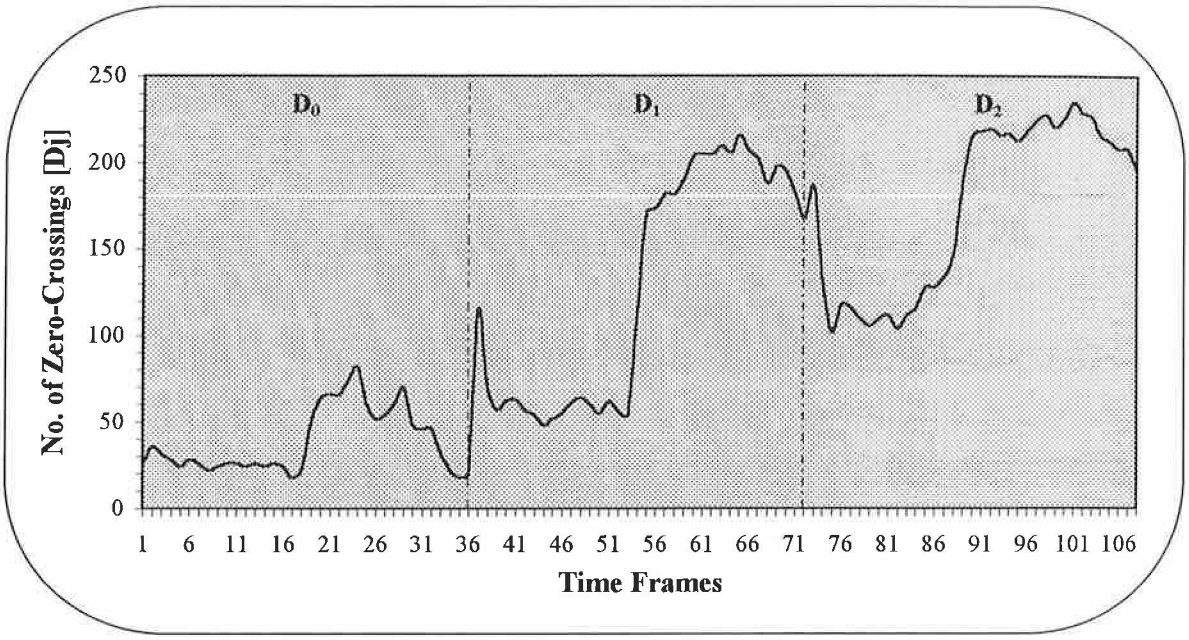


Figure D3: Tagged CHOC plot for the utterance '*Off*' up to the second order.

D4. Necessary Formatting in the Recognition Stage.

As in the training stage, *Dtwclus6.c* was not compatible with the data produced by *Hocalg5.m* and so, *Wordrec.m* (Figure 5.7) was written to overcome these difficulties. This m-file converts the data from ASCII to binary and places it in a file called *Test.dat*. It also runs *Dtwclus6.exe* to avoid the need of leaving the Matlab environment.

Dtwclus6.c functions as described by the flowchart in Figure 5.8. The CHOC data of the unknown test signal (*Test.dat*) is read into the structure: *[Length].[CHOC_Data]* and compared to each stored reference template in *Train.dat* (acquired from *Train.c*) using DTW. The minimum paths are then displayed and the decision rule (which is described in the next section) is applied. If the chosen utterance has a minimum path value of greater than 4000, the system informs the user that an appropriate match has not been made. Otherwise, the recognised word is displayed. (Note: flowcharts summarising the calculation of the LDM and ADM in both the training and matching stages are shown in Figures 5.9 and 5.10).

D5. Discussion on the Significance of the Lombard Effect.

The following indirect effects appear to occur when the speaker is subjected to noise:

1. Volume increase.
2. Vocal effort increase.
3. Change in pitch (generally an increase).
4. Narrower range in pitch variation.
5. Falling intonation.
6. Length and quality of vowels change.
7. Sounds tend to become nasalised.
8. Duration of segments change: -vowels lengthen;
   -consonants shorten;
   -overall duration increases.

Hence, even if the noise component could be removed by filtering, the characteristics of the speech signal would still be different. Young explains in [76] that a similar effect may be observed if an echo is detected by the speaker resulting in a decreasing of rate and lengthening of vowel duration.

These phenomena prove problematic when the speech recognition system is located in a noisy environment, such as a fighter plane cockpit, or an echoed environment, such as a long-distance telephone information service.

These effects should be taken into consideration when training the system and teaching the user how to use it (hence the need to test the system's vulnerability to intonation changes - Section 5.8).

Appendix E

```
/************************************* SIFE.H ********************************/
                    /* This is the header file to be included in all the modules */
/* Macros */
#define CONTROL   0x303
#define PORTA    0x300
#define PORTB    0x301
#define PORTC    0x302
#define CONTROL2  0x323
#define PORTA2    0x320
#define PORTB2    0x321
#define PORTC2    0x322
#define ERROR     0x00

#define GO       0x01
#define LEFT      0x02
#define RIGHT     0x03
#define STOP      0x04
#define REV       0x05
#define pi        3.14159

#define MAX_NUM_SAMPLES  500
#define TRIG_LEV         3000
#define TRUE 1
#define FALSE 0

/* Functions */
void init(void);
void titlepage(void);
void waitdis(void);
void getdat(void);
void datfin(void);
void portset(void);
float digitise(void);
int valsig(int,float *);
void pltdis(int,float *);
void mtrsig(int);
void prnt(void);
int msqren(int,float *,float);
int mnab(int,float *,float);
int mean(int,float *,float);
int varc(int,float,float *,float);
int coefvar(int,float *,float,float,float);
/*void samper(int);*/
int area(float,int,float *);
int probden(int,float *);
int gauspro(int,int,float *);
int acor(int,float *);
int autocross(int,float *);
int minpath(int,float *);
int hmsqr(int,float *);
int habs(int,float *,int);
int hprob(int,float *);
int freqcnt(int,float *);
int ngeomean(int,float *);
int tgeomean(int,float *);
int ngm(int,float *);
int lgm(int,float *);
/****************************************************************************/
```

```
/************************************ SIFEMAIN.C ********************************/
    /*    The main c-program to obtain a SIFE signal from recorded speech utterances        */

#include <time.h>
#include <stdio.h>
#include <dos.h>
#include "c:\tc\keith\sife.h"

main()
{
float data[MAX_NUM_SAMPLES+2],waitdata;
int input[2*(MAX_NUM_SAMPLES+1)];
float T=0, Tdum=0.0;
int t,i,value,valid,command,finish;


time_t start, end;

FILE *ptr;

   clrscr();
   init();                  /* initialisation of graphics */
   titlepage();               /* displays titlepage */


        for(t=0;t<=MAX_NUM_SAMPLES;t++)       /* initialisation of array */
          {                  /*  containing samples of  */
           data[t]=0;            /*     input signal      */
          }



        portset();              /* initialises ports */


/* Get data from A/D using function digitise */

finish = FALSE;
while(!finish)
     {
        waitdis();       /* allows user know when system */
                            /*  is ready for speech signal  */

        while(digitise() < TRIG_LEV); /* waiting for a valid signal */
                                          /* ie. sample with amplitude  */
                                          /* greater than about 3/4 of  */
                                          /* the maximum amplitude which*/
                                          /*    is 4096 (ie.12 bits)    */

        getdat();      /* displays screen while getting data */

        for(t=1;t<=MAX_NUM_SAMPLES;t++)
          { start = time('\0');
           input[t] = digitise();    /* fills array with 1000 samples */
           end = time('\0');
           Tdum+=(end-start);
          }
        T=Tdum/MAX_NUM_SAMPLES;        /* calculates sample interval */
```

```
          for(t=1;t<=MAX_NUM_SAMPLES;t++)
            {                     /* converts time interval information */
             input[t] = 1/input[t];  /*     into frequency information    */
            }

          datfin();    /* allows user to know when all data has been entered */

   printf("\n Sampling Interval : %f seconds",T);



          for(t=1;t<=MAX_NUM_SAMPLES;t++)    /* centres samples around zero */
            {
             data[t]= (((float)input[t]*10)/4096.0)-5;
            }

      valid = 9999;   /*     sets valid to an 'errored' value      */
                            /* before calculation of real value of valid */

      valid=valsig(valid,data);       /* selects valid portion of signal */

      if((valid < 0) || (valid > MAX_NUM_SAMPLES))     /*if valid remains */
            {                                     /*  at 9999 then   */
            printf("Error has occured\nvalid = %d",valid); /*error has occured*/
            exit(0);
            }
      if(valid==0)
            {
            printf("\n\n\n\n\n\n\n                  SORRY,");
            printf("\n\n\n           NO SPEECH SIGNAL HAS BEEN RECEIVED...");
            printf("\n\n\n                  TRY AGAIN!!!\n\n\n\n\n\n\n");
            }
      else
            {
             msqren(valid,data,T);/*calculates mean square energy of samples*/
            }

/*      ptr=fopen("a:\\data.dat","w");    /* writes array of data */
/*      for(i=0;i<valid;i++)            /* to file in 'a-drive' */
/*      fprintf(ptr," %f \n",data[i]);   /* to allow examination */
/*      fclose(ptr);                  /*      of samples      */


      printf("\n        ENTER 'C' TO CONTINUE OR ANYTHING ELSE TO END...");
      if (tolower(getch()) == 'c')
             { finish = FALSE;
               clrscr();
               cleardevice();
             }
      else finish = TRUE;

      cleardevice();
      }
      closegraph();

} /* END OF MAIN */
/*****************************************************************************/
```

```
/*********************************** GRAFUNC.C *****************************/
        /* Any graphic functions necessary for the SIFE program are included in this module */
#include <graphics.h>

/* ***FUNCTION TO INITIALISE GRAPHICS*** */
void init(void)
{
int g_driver, g_mod, errorcode;
        setcbrk(1);
        g_driver = DETECT;

        initgraph(&g_driver, &g_mod, "c:\\tc");
        if((errorcode=graphresult()) != grOk)
          {
           printf("BGI flie not in c:\\tc or \n");
           printf("graphics error : %s\n", grapherrormsg(errorcode));
           exit(1);
          }
        cleardevice();
        settextstyle(TRIPLEX_FONT,0,3);
} /* End of init */

/* ***FUNCTION TO DISPLAY TITLEPAGE*** */
void titlepage(void)
{
int cm,xm,ym;

   cm=getmaxcolor();
   xm=getmaxx();
   ym=getmaxy();

   cleardevice();
   setbkcolor(RED);
   setfillstyle(SOLID_FILL,GREEN);                    /* creates border */
   bar(xm/40,ym/40,(int)((long)(xm*39)/40),(int)((long)(ym*39)/40));
   setfillstyle(SOLID_FILL,BLUE);                  /* creates background */
   bar(xm*3/40,ym*3/40,(int)((long)(xm*37)/40),(int)((long)(ym*37)/40));
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
   setcolor(YELLOW);                                                  /*    sets     */
   outtextxy(xm/4,ym/4-50,"SPEECH RECOGNITION SYSTEM");               /*             */
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);                            /*             */
   outtextxy(xm/4+15,ym/3-50,"FOR AN ELECTRIC WHEELCHAIR ");          /*  textstyle  */
   settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);                            /*             */
   setcolor(WHITE);                                                   /*    and      */
   outtextxy(xm/3+60,ym/3+50,"DCU/DIT");                              /*             */
   outtextxy(xm/4+40,ym/2,"Master's Degree Project");                 /*   colour    */
   settextstyle(SANS_SERIF_FONT,HORIZ_DIR,1);                         /*             */
   setcolor(CYAN);                                                    /*             */
   outtextxy(xm*4.5/12+10,ym*5/8,"K.NEWSOME");                        /*    and      */
   setcolor(LIGHTRED);                                                /*             */
   outtextxy(xm*7/24,ym*7/9,"Supervisors: Dr. R. Scaife");            /*             */
   outtextxy(xm/4+139,ym*7.35/9,"Dr. E. Coyle");                      /*   prints    */
   setcolor(WHITE);                                                   /*             */
   settextstyle(SMALL_FONT,HORIZ_DIR,5);                              /*             */
   outtextxy(xm*2/3,ym*39/40,"Hit a key...");                         /*   "..."     */
   getch();
} /* End of titlepage */
```

```
/* ***FUNCTION TO DISPLAY 'WAITING' SCREEN*** */
void waitdis(void)
{
int cm,xm,ym;

  cm=getmaxcolor();
  xm=getmaxx();
  ym=getmaxy();

  cleardevice();
  setbkcolor(BLUE);
  setfillstyle(SOLID_FILL,CYAN);
  bar(xm/4,ym/4,(int)((long)(xm*30)/40),(int)((long)(ym*30)/40));
  settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
  setcolor(YELLOW);
  outtextxy(xm/50+160,ym/50+120,"Waiting for Voice Trigger...");
} /* End of waitdis */

/* ***FUNCTION TO DISPLAY 'GETTING DATA' SCREEN*** */
void getdat(void)
{
int cm,xm,ym;

cm=getmaxcolor();
xm=getmaxx();
ym=getmaxy();

cleardevice();
setbkcolor(BLUE);                                                    /*   sets    */
setfillstyle(SOLID_FILL,CYAN);                                       /* colour,   */
bar(xm/4,ym/4,(int)((long)(xm*30)/40),(int)((long)(ym*30)/40));      /* border    */
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);                              /*   and     */
setcolor(YELLOW);                                                    /* textstyle */
outtextxy(xm/50+160,ym/50+120,"GETTING DATA...");                    /*   and     */
outtextxy(xm/50+330,ym/50+320,"PLEASE WAIT...");                     /*  prints   */
} /* End of getdat */                                                /*   "..."   */

/* ***FUNCTION TO DISPLAY 'FINISHED' SCREEN*** */
void datfin(void)
{
int cm,xm,ym;

cm=getmaxcolor();
xm=getmaxx();
ym=getmaxy();

cleardevice();
setbkcolor(BLUE);                                                    /*   sets    */
setfillstyle(SOLID_FILL,CYAN);                                       /* colour,   */
bar(xm/4,ym/4,(int)((long)(xm*30)/40),(int)((long)(ym*30)/40));      /* border    */
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);                              /*   and     */
setcolor(LIGHTGREEN);                                                /* textstyle */
outtextxy(xm/50+160,ym/50+120,"FINISHED...");                        /*   and     */
outtextxy(xm/50+330,ym/50+320,"HIT ANY KEY...");                     /*  prints   */
getch();                                                             /*   "..."   */
cleardevice();
} /* End of datfin */
/*****************************************************************************/
```

```
/********************************** SIGANA.C **********************************/
/*          This module deals with the signal analysis necessary to obtain the SIFE signal       */
/*                                  and output the recognised command                            */

#include <graphics.h>
#include <stdio.h>
#include "c:\tc\keith\sife.h"

int command=0,offset=0;

/* ***FUNCTION TO INITIALISE PORTS*** */
void portset(void)
{
/* Setup first DIO card (address 300hex) to expect A/D data IN at PORTA  */
/* and PORTC (LOWER) and data OUT on PORTB and PORTC (UPPER)  i.e. 91HEX */
     outportb(CONTROL,0x91);

/* Setup second DIO card (address 320hex) to expect data OUT at PORTA */
/* and PORTC (UPPER) the control registers of the D/A's   i.e. 80HEX */
     outportb(CONTROL2,0x80);
     outportb(PORTC2,0xFF);

/* Setup MUX we want channel 1 therefore */
/* write out 001 binary to MUX on PORTB  */
     outportb(PORTB,0x01);
} /* End of portset */

/* ***FUNCTION TO READ-IN SAMPLES*** */
float digitise(void)
{
int i,test=0;
int aadat, ccdat, ddata;

   /*    Sampling Frequency set by the delay loop       */
   /*   and the constant ie. MAX_NUM_SAMPLES   */
          outportb(PORTC,0xFF);
     i=0;
     while (i<500)
     {
         test=test*2;
      i++;
     }

          outportb(PORTC,0x00);
     i=0;
     while (i<500)
     {
         test=test*2;
     i++;
     }

                  outportb(PORTC,0xFF);
     i=0;
     while (i<500)
     {
     test=test*2;
     i++;
     }
```

```
            aadat=inportb(PORTA);
            ccdat=inportb(PORTC);

        ddata = ((ccdat & 0x000F)*256) + aadat;        /* adding PORTA to the 4 */
                                                       /* LSB of PORTC shifted  */
                                                       /*    8 bits to the left to   */
                                                       /*       form a 12 bit        */
                                                       /*  representation of the     */
                                                       /*           sample           */

        return(ddata);

    } /* End of digitise */

    /* ***FUNCTION TO VALIDATE SIGNAL*** */
    int valsig(int valid,float data[])
    {
    int i, t, ab, temp1, offset;


       for (i=2;i<MAX_NUM_SAMPLES;i++)                 /* finds beginning of signal */
         {
            if(( (data[i]) - (data[i+1]) )!=0)
         break;
         }
           offset=i;

           for(i=MAX_NUM_SAMPLES;i>0;i--)              /* finds end of signal */
           {
           ab = data[i] - data[i-1];
           if((ab>.5)||(ab<-.5)) break;
           }
        temp1=i;

        valid=temp1-offset;                      /* calculates no. of valid samples */
        if(valid<0) valid=0;

        t=0;

        do
           { data[t] = data[t+offset];           /* removes invalid samples */
          t++;                                   /*  at beginning of signal   */
           } while(t<valid);                     /*       from data array      */

           return(valid);
    } /* End of valsig */
    /**************************************************************************/
```

```
/*********************************** ALGRTMS.C ******************************/
/*                  This module contains all the statistical algoritms required in the          */
/*                                   recognition of the SIFE signal                               */

#include <graphics.h>
#include <math.h>
#include "c:\tc\keith\sife.h"

float T;

/* ***MEAN SQUARE ENERGY FUNCTION*** */
int msqren(int valid,float data[],float T)
{
 int t=0;
 float dummy=0,dataave=0;

    for(t=1;t<=valid;t++)                    /* calculates mean */
         {                               /*  square value   */
         dummy += ((data[t])*(data[t]));        /*   of samples    */
         }
    dataave = dummy/valid;
     printf("\n Mean square energy for speech : %f", dataave);
     mnab(valid,data,T);

} /* End of msqren */

/* ***MEAN ABSOLUTE FUNCTION*** */
int mnab(int valid,float data[],float T)
{
int t = 0;
float dumm=0.0,dataav=0.0;

    for (t = 1;t<=valid;t++)        /* calculates average of */
    {                          /*    absolute values    */
     dumm += fabs(data[t]);         /*      of samples       */
    }
    dataav = dumm/valid;
    printf("\n Abs value : %f",dataav);
    mean(valid,data,T);
} /* End of mnabs */



/* ***MEAN FUNCTION*** */
int mean(int valid,float data[],float T)
{
int t=0;
float meanval=0,dummy=0;

  for (t=1;t<=valid;t++)        /* calculates average */
    {
    dummy += (data[t]);       /*     of samples     */
    }
  meanval=dummy/valid;
  printf("\n Mean Value : %f",meanval);
  varc(valid,meanval,data,T);
}
```

```
/* ***VARIANCE FUNCTION*** */
int varc(int valid,float meanval,float data[],float T)
{
int t=0;
float varval=0,dummy=0;

  for (t=1;t<=valid;t++)
     {
     dummy += ((data[t]-meanval)*(data[t]-meanval));
     }
  varval=dummy/valid;          /* calculates variance of samples */
  printf("\n Variance : %f",varval);
  coefvar(valid,data,T,varval,meanval);
} /* End of varc */

/* ***COEFFICIENT OF VARIATION FUNCTION*** */
int coefvar(int valid,float data[],float T,float varval,float meanval)
{
float cv=0;

cv=(varval/meanval)*100;
printf("\n Coefficient Of Variation : %f",cv);

area(T,valid,data);
}

/* ***FUNCTION TO FIND SAMPLING INTERVAL*** */
/*float samper()
{
time_t start,end;
int j;
float Tdum=0,T=0;

  while(j<1000)
     { start=time('\0');
      digitise();
      end=time('\0');
      Tdum=Tdum+(end-start);
      j++;
     }
  T=Tdum/1000;
  printf("\n Sampling Interval T :%f seconds",T);
  return(T);
} */

/* ***AREA FUNCTION*** */
int area(float T,int valid,float data[])
{
int t=0;
float curarea,dummy=0;

  for (t=1;t<=valid;t++)       /* calculates area between signal and x-axis */
     {dummy += (data[t]);     /* using rectangular approximation (ie. sum  */
     }                  /*  of sample amplitude by sample interval   */
  curarea=dummy*T;
  printf("\n Area Under Curve : %f",curarea);
  probden(valid,data);
} /* End of area */
```

```
/*  ***PROBABILITY DENSITY FUNCTION***   */
/*     ***FROM FIRST PRINCIPLES***       */
/*                                       */
/*       ie.  P(y) = No. of samples in range   */
/*                   Total no. of samples      */
/*                                       */
int probden(int valid,float data[])
{
int t=0;
float pdf,num=0;
float AR1=3.0,AR2=2.0;

 for (t=1;t<=valid;t++)
    {if ((data[t] <= AR1) && (data[t] >= AR2))
    num++;
    }          /* calculates probability that a sample will fall */
 pdf=num/valid;    /*      within the range AR1 to AR2         */

 printf("\n Probability Density Function From First Principles For The");
 printf("\n    Range Of Amplitudes %f To %f : %f",AR1,AR2,pdf);

 gauspro(num,valid,data);
 } /* End of probden */

/*  ***PROBABILITY DENSITY FUNCTION***   */
/*     ***FROM GAUSSIAN FORMULA***       */
/*                                       */
/*                  1     exp[(-y-ý)²]   */
/*       ie. P(y) = σ√2π      [ 2σ²  ]   */
/*                                       */
/*                                       */
int gauspro(int num,int valid,float data[])
{
int t=0;
float gpdf=0,dum1=0,dum2=0,dumean=0,duvar=0,dugpdf=0;
float AR1=3.0,AR2=2.0;

 if(num!=0)
 {
  for (t=1;t<=valid;t++)
    {if ((data[t] <= AR1) && (data[t] >= AR2))
    dum1 += (data[t]);
    }
  dumean=dum1/num;
  for (t=1;t<=valid;t++)
    {if ((data[t] <= AR1) && (data[t] >= AR2))
    dum2 += ((data[t]-dumean)*(data[t]-dumean));
    }
  duvar=dum2/num;
  for (t=1;t<=valid;t++)
    {if ((data[t] <= AR1) && (data[t] >= AR2))
    dugpdf=(1/(sqrt(duvar*2*pi)))*exp((-((data[t]-dumean)*(data[t]-dumean))/(2*duvar)));
    }
                              /* finds Gaussian probability sample */
  gpdf += dugpdf;      /* will fall within range AR1 to AR2 */
  }
 else gpdf=0;
```

```
  printf("\n Probability Density Function From Gaussian Formula For The");
  printf("\n     Range Of Amplitudes %f To %f : %f",AR1,AR2,gpdf);

  acor(valid,data);
 } /* End of gauspro */

/* ***FUNCTION FOR MEAN AUTOCORRELATION COEFFICIENTS*** */
int acor(int valid,float data[])
{
int j=0,n=0;
float dum1=0,dum2=0,prod=0,acorm=0;

  for(j=0;j<=(valid-1);j++)              /* finds average of */
    { for(n=1;n<=(valid-j);n++)          /* autocorrelation  */
        {prod=(data[n])*(data[n+j]);    /*   coefficients   */
            dum1 += prod;
        }
      dum2 += dum1;
    }
  acorm=dum2/valid;
  printf("\n Correlation Coefficients Mean : %f",acorm);
  autocross(valid,data);
} /* End of acor */

/* ***FUNCTION TO FIND FIRST CROSSING OF X-AXIS*** */
/*       ***IN AUTOCORRELATION FUNCTION***         */
int autocross(int valid,float data[])
 {
int j=0,n=0,cross=0;
float dum1=0,prod=0;

  for(j=0;j<=(valid-1);j++)              /* determines where the plot of */
    { for(n=1;n<=(valid-j);n++)          /*   autocorrelation function   */
            { prod=(data[n])*(data[n+j]); /*   first crosses the x-axis   */
             dum1+=prod;
            }
      if(dum1<=0.0) break;
    }
  cross=j;
  printf("\n First Non-positive Autocorrelation Coefficient : %d",cross);
  minpath(valid,data);
} /* End of autocross */

/* ***MINIMUM PATH FUNCTION*** */
int minpath(int valid,float data[])
{
int t=2;
float sum=0,dum=0,lastpt=0;

  sum=data[1];
  while(t<valid)
    { if(data[t]<=data[t+1])
            { sum += (data[t]);
          t++;
        }
      else
            { sum += (data[t+1]);
            t += 2;
        }
      }
  if(data[valid]<=data[1])
    { sum += (data[valid]);
     t=1;
    }
  else
    {sum += (data[1]);
    t=2;
    }
  while(t<(valid-1))
       { if(data[t]<=data[t+1])
          {dum=data[1];
          t++;
          }
        else
          {dum=data[t+1];
              t += 2;
          }
            sum += dum;
       }
  lastpt=dum;
  if(lastpt==data[valid-2])
    {sum += (data[valid-1]);
    }
  printf("\n Minimum Path Value : %f",sum);
  hmsqr(valid,data);
} /* End of minpath */

/* ***HALF-MEAN-SQUARE FUNCTION*** */
int hmsqr(int valid,float data[])
{
int hw=0,t=0;
float dum1=0,dum2=0,havsqr1=0,havsqr2=0;

  if((valid%2)==0) hw=valid/2;   /* finds halfway mark of signal */
  else hw=(valid-1)/2;

  for(t=1;t<=hw;t++)             /* calculates mean square energy */
    {                     /*    of first half of signal     */
     dum1 += data[t]*data[t];
    }
  havsqr1=dum1/hw;

  for(t=hw+1;t<=valid;t++)       /* calculates mean square energy */
    {                     /*   of second half of signal     */
     dum2 += data[t]*data[t];
    }
  havsqr2=dum2/hw;

  printf("\n Mean square energy of first half of signal : %f",havsqr1);
  printf("\n Mean square energy of second half of signal : %f",havsqr2);

  habs(valid,data,hw);
}/* End of hmsqr */


/* ***HALF-ABSOLUTE-VALUE FUNCTION*** */
```

```
int habs(int valid,float data[],int hw)
{
int t=0;
float dum1=0,dum2=0,hav1=0,hav2=0;

  for(t=1;t<=hw;t++)          /* calculates mean square energy */
    {                     /*    of first half of signal    */
     dum1 += fabs(data[t]);
    }
  hav1=dum1/hw;

  for(t=hw+1;t<=valid;t++)      /* calculates mean square energy */
    {                     /*   of second half of signal    */
     dum2 += fabs(data[t]);
    }
  hav2=dum2/hw;

  printf("\n Abs. value of first half of signal : %f",hav1);
  printf("\n Abs. value of second half of signal : %f",hav2);

  hprob(valid,data);
}/* End of habs */
/* ***HALF-PROBABILITY FUNCTION*** */
int hprob(int valid,float data[])
{
int t=0,hw=0;
float pdf1=0,pdf2=0,num1=0,num2=0;
float AR1=3.0,AR2=2.0;

  if((valid%2)==0) hw=valid/2;  /* finds halfway mark of signal */
  else hw=(valid-1)/2;

  for (t=1;t<=hw;t++)
     {if ((data[t] <= AR1) && (data[t] >= AR2))
     num1 += data[t];
     }            /* calculates probability that a sample will fall    */
  pdf1=num1/hw;      /* within the range AR1 to AR2 in 1st half of signal */

  for (t=hw+1;t<=valid;t++)
     {if ((data[t] <= AR1) && (data[t] >= AR2))
     num2 += data[t];
     }            /* calculates probability that a sample will fall    */
  pdf2=num2/hw;      /* within the range AR1 to AR2 in 2nd half of signal */

  printf("\n PDF In 1st Half Of Signal From First Principles For The");
  printf("\n     Range Of Amplitudes %f To %f : %f",AR1,AR2,pdf1);

  printf("\n PDF In 2nd Half Of Signal From First Principles For The");
  printf("\n     Range Of Amplitudes %f To %f : %f",AR1,AR2,pdf2);
  freqcnt(valid,data);
}/* End of hprob */


/* ***FUNCTION TO ESTABLISH FREQUENCY OF OCCURANCE OF SAMPLES*** */
int freqcnt(int valid,float data[])
{
int i=0,n=0,freq=0;
float freqav[6];
```

```
  printf("\n Log10 Geometric Mean : %f",lave);
  ngm(valid,data);
 }

 int ngm(int valid,float data[])
 {
 int i=0;
 float ldat=0,lave=0;

  for(i=0;i<=valid;i++)
    {
    ldat += exp(data[i]);   /* calculates ln{SUM(exp(data[i]))/valid} */
    }
  lave=log(ldat/valid);

  printf("\n ngm : %f",lave);
  lgm(valid,data);
 }

 int lgm(int valid,float data[])
 {
 int i=0;
 float ldat=0,lave=0;

  for(i=0;i<=valid;i++)
    {
    ldat += pow10(data[i]);   /* calculates log{SUM(10^(data[i]))/valid} */
    }
  lave=log10(ldat/valid);

  printf("\n lgm : %f",lave);

 }
/*******************************************************************************/
```

```
/************************************ SIFE.PRJ **********************************/
                /* This is the project file necessary to run all the modules together */
c:\specon\sifemain.c
c:\specon\grafunc.c
c:\specon\sigana.c
c:\specon\algrthm.c
/*******************************************************************************/
```

```
%******************************** MEDFILT.M ********************************
% This function removes unwanted spurious fluctuations from a SIFE signal using a median filter.
% The resulting signal is loacally smoothed, but retains the important sudden signal changes due
% to a transition to a new phoneme and does not combine error samples into the signal as in a
% linear or averaging filter.
%
% Usage: medfilt(x,n)      ,where x is the SIFE signal and n is the filter order.
% Note: n must be ODD !
%***************************************************************************
function filsig=medfilt(x,n)

k=0; flag=1; tmp=0;

sifelen=length(x);
k=sifelen/n;                % the number of frames in the signal
f=x;
frstr=1;
frend=n;
for i=1:k,
        flag=1;
        while flag==1,
                flag=0;
                for j=frstr:frend-1,
                        if f(j)>f(j+1),                 % bubblesort of data
                                tmp=f(j+1);
                                f(j+1)=f(j);
                                f(j)=tmp;
                                flag=1;
                        end;
                end;
        end;
med=(frend-((n-1)/2));                          % median value

if frend<=sifelen,
        for i=frstr:frend,
                filsig(i)=f(med);               % signal consisting of median values
        end;
frstr=frstr+n;                                  % slide window
frend=frend+n;
end;
end;
%***************************************************************************
```

Appendix F

```
%************************************ DIFFIL.M ********************************
% Operates a difference filter: (1-B)^n  on a signal x.
% ie. Y(t) = {(1-B)^n}X(t) = c(1)X(t) - c(2)BX(t) +...+ c(n+1)B^nX(t)
% Usage : diffil(x,n) ,where x is signal, n is filter order.
%*******************************************************************************
function [fd]=diffil(x,n)            % x is signal, n is order of filter.

l=length(x);
a=[]; b=[]; c=[];
q=[]; r=[]; s=zeros(1,l);

a=pascal(n+1);                       % pascal matrix of order n+1
b=fliplr(a);                         % flip matrix to remove right-left diagonal
c=diag(b);                           % returns coefficients of polynomial (1+B)^n.

for m = 1:(n+1),                     % repeats for each polynomial coefficient.
  q = roll(x,(m-1));                 % delay operator.
  r = ((-1)^(m-1))*c(m) * q;         % mults. coeff. by delayed sig. elements.
  s = s + r;                         % incs. sum by new sig. elements.
  fd = s;
end;
%fd
%*******************************************************************************
```

```
%********************************* SUMFIL.M **********************************
% Operates a summation filter: (1+B)^n  on a signal x.
% ie. Y(t) = {(1+B)^n}X(t) = c(1)X(t) + c(2)BX(t) +...+ c(n+1)B^nX(t)
% Usage : sumfil(x,n) ,where x is signal, n is filter order.
%*******************************************************************************
function [fs]=sumfil(x,n)            % x is signal, n is order of filter.

l=length(x);
a=[]; b=[]; c=[];
q=[]; r=[]; s=zeros(1,l);

a=pascal(n+1);                       % pascal matrix of order n+1
b=fliplr(a);                         % flip matrix to remove right-left diagonal
c=diag(b);                           % returns coefficients of polynomial (1+B)^n.


for m = 1:(n+1),                     % repeats for each polynomial coefficient.
   q = roll(x,(m-1));                % delay operator.
   r = c(m) * q;                     % multiplies coefficient by delayed signal elements.
   s = s + r;                        % increments sum by new signal elements.
   fs = s;
end;
fs
%*******************************************************************************
```

```
%************************************ ROLL.M ***********************************
% This algorithm rolls signal x by k samples.
% For example, if x = [1 2 3 4] and k = 2 then,
%                                       roll(x,k) = [3 4 1 2]
% Usage: roll(x,k)          ,where x is a vector and k is a scalar
%******************************************************************************
function y = roll(x,k)

l = length(x);
a = []; b = [];

if k < 0,                          % For a negative value of k
    k = l + k;
end;
if k > l,                          % For a positive value of k
    k = k-l;
end;

for i = 1:(l - k),                 % Shifts first section of samples
          a(i) = x(i);
end;
for i = (l - k + 1):l,             % Shifts final section of samples
          b(i - (l - k)) = x(i);
end;

y = [b a];                         % Places samples in matrix y
%******************************************************************************


%********************************* ZCOUNT.M ***********************************
% Counts the no. of zero-crossings in a signal.
% Usage : zcount(x) ,where x is signal.
%******************************************************************************
function f=zcount(x)

y=[]; za=[];
z=zeros(1,1);
l=length(x);
for m=1:l,
   if x(m)>=0                      % creates ICS from speech signal
          y(m)=1;
   else y(m)=-1;
   end;
end;
%subplot(211),plot(x),subplot(212),plot(y)

for m=2:l,
   if y(m)~=y(m-1);                % if successive samples in the ICS are not equal
          za(m-1)=1;               %         a zero-crossing has occurred
   else za(m-1)=0;
end;
end;
for m=1:(l-1),                     % add up the total number of zero-crossings encountered
   z=z+za(m);
   f=z;
end;
%******************************************************************************
```

```
%********************************** HOCPLOT.M ********************************
% This m-file calculates HOC sequences and displays HOC plots for (summation, difference etc.)
% filters of order n.
% Usage : hocplot(x,n) ,where x is signal, n is order.
%*****************************************************************************
function f=hocplot(x,n)

a=[]; b=[];
z1=[]; z2=[];
domfreqs=[]; domfreqd=[];

clg;
l=length(x);

plot(x), title('Input Signal'), xlabel('Sample No.'), ylabel('Amplitude'), pause

clg;
for m=0:n,
   a=sumfil(x,m);                        % applies summation filter to signal.
   Z1(m+1)=zcount(a);                    % counts no. of zero-crossings in filtered signal.
   domfreqs(m+1)=(pi*z1(m+1))/(l-1);     % estimates dominant frequency.
   B=diffil(x,m);                        % applies difference filter to signal.
   z2(m+1)=zcount(b);                    % counts no. of zero-crossings in filtered signal.
   domfreqd(m+1)=(pi*z2(m+1))/(l-1);     % estimates dominant frequency.
end;

domfreqs
domfreqd

subplot(211), plot(z1), title('HOC Plot for Summation Filters'), xlabel('Filter Order + 1  (n+1)'),
ylabel('No. of Crossings');

subplot(212), plot(z2), title('HOC Plot for Difference Filters'), xlabel('Filter Order + 1  (n+1)'),
ylabel('No. of Crossings');
%*****************************************************************************
```

```
%******************************** HOCDIF.M *********************************
% Modified version of diffil.m for speed and efficiency, using a dif. fil.
% Usage : hocdif(x,n)   ,where x=speech signal, n=order of HOC seq.
%***************************************************************************
function f=hocdif(x,n)

hocdat=[]; y=[];
yy=[]; zc=0; l=0;

l=length(x);
y=x;

for s=0:n,                            % for each filter order 0-n
        zc=0;

        for m=1:l,                    % create ICS signal
                if y(m) >= 0
                        yy(m)=1;
                else    yy(m)=-1;
                end;
        end;

        for m=2:l,                    % if ICS sample(m) does not equal
                if yy(m) ~= yy(m-1)   % sample(m-1) sign change took place
                        zc=zc+1;      % ie. zero-crossing
                end;
        end;

        hocdat(s+1)=zc;               % matrix of hoc data
        z=(roll(y,1))';               % } [Y(z+1)-Y(z)]/2
        y=(y-z)/2;                    % } divide by 2 to ensure stability

%save new y;
end;

f=hocdat;
%***************************************************************************
```

```
%********************************* HOCSUM.M **********************************
% Modified version of hocroc.m for speed and efficiency, using a summation filter.
% Usage : hocsum(x,n)   ,where x=speech signal, n=order of HOC sequence.
%*****************************************************************************
function f=hocsum(x,n)

hocsum=[]; y=[];
yy=[]; zc=0; l=0;

l=length(x);
y=x;

for s=0:n,                                    % for each filter order 0-n
         zc=0;

         for m=1:l,                           % create ICS signal
                  if y(m) >= 0
                           yy(m)=1;
                  else   yy(m)=-1;
                  end;
         end;

         for m=2:l,                           % if ICS sample(m) does not equal
                  if yy(m) ~= yy(m-1)         % sample(m-1) sign change took place
                           zc=zc+1;           % ie. zero-crossing
                  end;
         end;

         hocsum(s+1)=zc;                      % matrix of hoc data
         z=(roll(y,1))';                      % } [Y(z+1)+Y(z)]
         y=y+z;                               % }
end;

hocsum
%*****************************************************************************
```

```
%********************************* HOCSSKY.M *********************************
% Modified version of hocroc.m for speed and efficiency, using a Slutsky filter
% Usage : hocssky(x,n)   ,where x=speech signal, n=order of HOC seq.
% Slutsky filter : L[m,n] = [(1-B)^(m-1)]*[(1+B)^(n-1)]
%*****************************************************************************
function f=hocssky(x,n)

hocssky=[]; y=[];
yy=[]; zc=0; l=0;

l=length(x);
y=x;
for s=0:n,
        zc=0;
        for m=1:l,
                if y(m) >= 0
                        yy(m)=1;
                else    yy(m)=-1;
                end;
        end;
        for m=2:l,
                if yy(m) ~= yy(m-1)
                        zc=zc+1;
                end;
        end;
        hocssky(s+1)=zc;
        z=(roll(y,2))';                 % (1-B)*(1+B)=(1-B^2)
        y=y-z;
end;
hocssky
%*****************************************************************************
```

```
%********************************** HOCALG.M **********************************
% This algorithm is used to:
%         (i)Pad the signal to allow an integral number of window sections;
%         (ii)Window the signal, (using a 30 msec hanning window,1/2 overlap);
%         (iii)Carry out HOC analysis on each windowed section using a differencing filter;
%         (iv)Place HOC data into an n x m matrix called H, where n is the number of window
%         frames in signal and m is the filter order.

% Usage : hocalg(x,n)   ,where x=speech signal, n=order of HOC seq.
%*****************************************************************************
function f=hocalg(x,n)

hocalg=[]; y=[]; winsig=[];          % initialisation of matrices & variables
H=[]; Ht=[]; zc=0; l=0; numsmp=0;

winlen=330;                          % length of window
n1=1;                                % starting point of window
n2=winlen;                           % ending point of window

ovrlap=winlen/2;                     % length of overlap
l=length(x);                         % length of speech signal
y=x;

a=rem(l,165);                        % }calculation of no. of mirrored samples to be
numsmp=165-a;                        % }   appended to the signal to ensure
                                     % }       divisibility by overlap
padl=l+numsmp;                       % length of padded signal = original signal length
                                     %         plus number of samples to be added

ctr=0;
for i=(l+1):padl,                    % appending zeros to matrix
        y(i)=y(l-ctr);
        ctr=ctr+1;
end;

r=1;
while(n2<padl)                                  % }apply HOC analysis until
        winsig=y(n1:n2).*hanning(330);          % }complete signal has been
        H(r,:)=hocdif(winsig,n);                % }considered and place HOC
        r=r+1;                                  % }data into matrix H
        n1=n1+ovrlap;                           % sliding window
        n2=n2+ovrlap;
end;
f=H;
%*****************************************************************************
```

Appendix G

```
%****************************** RLKTRMS.M *********************************
% This function removes like-terms (ie. samples of the same amplitude) from a SIFE signal
% resulting in a sequence of data representing the spacing between each ZC present in the original
% speech signal.
% Usage: rlktrms(x) ,where x is the SIFE signal.
%***************************************************************************

function sifevals=rlktrms(x)

sifelen=length(x);
k=0;

for i=1:(sifelen-1),                  % if successive samples are equal only include once
        if x(i)~=x(i+1),
                k=k+1;
                sifevals(k)=x(i);
        end;
end;
if x(sifelen-1)~=x(sifelen),          % deals with second last and last samples
        sifevals(k+1)=x(sifelen);
end;
%***************************************************************************
```

```
%******************************** HOCALG5.M ********************************
% This algorithm is used to:
%         (i)Pad the signal to allow an integral number of window sections;
%         (ii)Window the signal,(using a 30 msec hanning window,1/2 overlap);
%         (iii)Carry out HOC analysis on each windowed section using a differencing filter;
%         (iv)Place HOC data into an n x m matrix called H, where n is the number of window
%             frames in signal and m is the filter order.
%         (v)Tag the D0,D1,D2 data strings end-to-end for compatiblity with DTW algorithm;
%
% Usage : hocalg5(x,n,'dstfname')
%                                , where x=speech signal,
%                                        n=order of HOC sequence,
%                                        dstfname=name of file to contain HOC data.
%***************************************************************************

function f=hocalg5(x,n,dstfname)

hocalg=[]; y=[]; winsig=[];             % initialisation of matrices & variables
H=[]; htrans=[]; htag=[];
zc=0; l=0; numsmp=0;

winlen=330;                             % length of window
n1=1;                                   % starting point of window
n2=winlen;                              % ending point of window

ovrlap=winlen/2;                        % length of overlap
l=length(x);                            % length of speech signal
y=x;

a=rem(l,165);                           % }calculation of no. of mirrored samples to be
numsmp=165-a;                           % }   appended to the signal to ensure
                                        % }       divisibility by overlap
padl=l+numsmp;                          % length of padded signal = original signal length
                                        %         plus number of samples to be added

ctr=0;
for i=(l+1):padl,                       % appending samples to matrix
        y(i)=y(l-ctr);
        ctr=ctr+1;
end;

r=1;
while(n2<padl)                                 % }apply HOC analysis until
        winsig=y(n1:n2).*hanning(330);         % }complete signal has been
        H(r,:)=hocdif(winsig,n);               % }considered and place HOC
        r=r+1;                                 % }data into matrix H
        n1=n1+ovrlap;                          % sliding window
        n2=n2+ovrlap;
end;
htrans=H';                                     % transpose the H matrix
htag=[htrans(1,:),htrans(2,:),htrans(3,:)]     % tags rows of data

hdr=length(htag);                              % file header is length of string htag
htag_hdr=[hdr,htag];                           % append header to string

datoint(htag_hdr,dstfname);                    % converts data to integer format and stores
                                               % data in file specified by dstfname
%***************************************************************************
```

```
%********************************** DATOINT.M ********************************
% This function converts an ascii file of 8 digits per data value saved in
%                          matlab to a file of integers.
% Usage: datoint(source_varname,destination_fname)
%*****************************************************************************

function f=datoint(var,fname)

l=length(var);

fid = fopen(fname,'w');

for n=1:l,
        y = var(n);
        fprintf(fid,'%4d',y);
end;

fclose(fid);
%*****************************************************************************
```

```
%********************************** TRAINALL.M ********************************
% This m-file trains the system with the full vocabulary
% Usage: trainall
%*****************************************************************************
trainer('tgo');
trainer('tstop');
trainer('trev');
trainer('tleft');
trainer('tright');
%*****************************************************************************
```

```
%******************************** TRAINER.M *********************************
% This m-file is used to train the speech recognition system for a certain vocabulary.
% Usage : trainer(word)
%****************************************************************************
function trainer(word)

pathout='f:\dtw\'; % sets output path

FIDout=fopen([pathout 'train.dat'],'wb');            % opens file to store training data
fprintf(FIDout,'%-8.8s',word);                       % places name of word as header to file

for k=0:9,

  % read data files to be trained

    pathin='f:\dtw\hdr\';                            % }defines input path
    extin='.dat';                                    % }and file extension

    datak = [word int2str(k)];                       % }finds file with filename
    filename = [pathin word int2str(k) extin];       % }of form "name_number.dat"
    if ~exist(filename), break, end                  % }  (eg. stop4.dat) and
    eval(['load ' filename ' -ascii']);              % }extracts data from it for
    data = eval(datak);                              % }       processing


% create training files for TRAIN.EXE

    len=length(data);                                % }length of data string is already
    fwrite(FIDout,(len-1),'uchar');                  % }present in data file
    fwrite(FIDout,data(2:len),'uchar');              % }=>len-1 = length of data so write
                                                     % }length and data to training file
end
fclose(FIDout);


!f:\dtw\train.exe                                    % runs train.exe
%****************************************************************************
```

```
/********************************** TRAIN.C **********************************/
#include <math.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <conio.h>
#include <values.h>
#include <string.h>

#define TEST_MAX 150        /* max size of test pattern */
#define WORDS_MAX 15         /* max no. of classes that can can be recognised */

void disp_cmatrix(unsigned char **, char ,char);
void disp_imatrix(unsigned int **, char ,char);




void main (void)
{
/* Declare and initialise variables and data structures */
unsigned char m, n;                          /* row and column index */
unsigned char i, j, c, k;                    /* for loop counters */
unsigned char N, M;                          /* lengths of current test and ref patterns */
unsigned char match[5], index;
unsigned int minAD=0;
unsigned int total, minPD;
unsigned avgPD[10];
FILE  *ftrain, *fref;                        /* file pointers for data files*/

struct training
{
 unsigned char name[8];
 unsigned char length[10];
 unsigned char data[10][TEST_MAX];
} *train;

/* Allocate memory required for data structures */
unsigned char(*LDM)[150]=calloc(150*150,1);
unsigned int(*ADM)[150]=calloc(150*150,2);
unsigned int(*PDM)[10]=calloc(10*10,2);

/* Check for existance of data files */
if((ftrain=fopen("f:\\dtw\\train.dat","rb"))==0)
          {
           printf("Error opening test.dat\n");
           exit(1);
          }

/* Get training patterns */
{
train=(struct training *)malloc(sizeof(struct training));
fread(&(train->name),sizeof(unsigned char),8,ftrain);
for(j=0; j<10; j++)
          {
           fread(&(train->length[j]),sizeof(unsigned char),1,ftrain);
           fread(&(train->data[j][0]),sizeof(unsigned char),train->length[j],ftrain);
          }
}
fclose(ftrain);

clrscr();
gotoxy(1,1);
printf("Training the word \'%.8s\'\n",train->name);
for(i=0; i<10; i++)
        {
         N=train->length[i];
         for(j=0; j<10; j++)
                {
                 gotoxy(1,2);
                 printf("%u -> %u",i,j);
                 M=train->length[j];




                 /* Calculate local distance matrix (LDM) */
                 for (n=0; n<N; n++)
                        for (m=0; m<M; m++)
                                LDM[m][n]=abs(train->data[i][n]-train->data[j][m]);

                /* Calculate accumulated distance matrix (ADM) */
                for (n=0; n<N; n++)
                        {
                         for (m=0; m<M; m++)
                                {
                                 if ((m-1)<0 && (n-1)<0)
                                        minAD=0;
                                 else if ((n-1)<0 )
                                        minAD=ADM[m-1][n];
                                 else if ((m-1)<0)
                                        minAD=ADM[m][n-1];
                                 else
                        minAD=min(ADM[m][n-1],min(ADM[m-1][n],ADM[m-1][n-1]));

                                 ADM[m][n]=(int)(LDM[m][n])+minAD;
                                }
                        }
                PDM[i][j]=ADM[M-1][N-1];
                }
        }

/* Display results */
clrscr();
printf("%.8s\n\n\t\t\tTESTED AGAINST\n\t",train->name);
for(i=0; i<10; i++)
        printf("U%u    ",i);

/* Display PDM matrix */
for(i=0; i<10; i++)
        {
         printf("\nU%u  ",i);
         for(j=0; j<10; j++)
                printf("%6u",PDM[i][j]);
        }
```

```
/* Find utterances to store as templates */
for (i=0; i<10; i++)
        {
        total=0;
        for(k=0; k<5; k++)
                {
                minPD=MAXINT;
                for (j=0; j<10; j++)
                        if (min(PDM[i][j],minPD)==PDM[i][j])
                                {
                                minPD=PDM[i][j];
                                index=j;
                                }
                total+=minPD;
                PDM[i][index]=MAXINT;
                }
        avgPD[i]=total;
        }

/* Find Minimum Totals */
printf("\n\nTOTALS\n\t");
for(i=0; i<10; i++)
        printf("%6u",avgPD[i]);

printf("\n\nSTORED :\t");
for(c=0; c<5; c++)
        {
        minPD=MAXINT;
        for(i=0; i<10; i++)
                if (min(avgPD[i],minPD)==avgPD[i])
                        {
                        match[c]=i;
                        minPD=avgPD[i];
                        }
        avgPD[match[c]]=MAXINT;
        printf("U%u     ",match[c]);
        }

/* Store patterns as templates */
fref=fopen("f:\\dtw\\ref.dat","ab");
fprintf(fref,"%-.8s",train->name);
for(j=0; j<5; j++)
        {
        fwrite(&(train->length[match[j]]),sizeof(unsigned char),1,fref);
        fwrite(&(train->data[match[j]][0]),sizeof(unsigned char),train->length[match[j]],fref);
        }
fclose(fref);

/* Exit routine */
free(train);
free(LDM);
free(ADM);
free(PDM);
}
/*********************************************************************************/
```

```
%******************************** MAKEINI.M ********************************
% This informs the system of the vocabulary size
%***************************************************************************
function makeini(x);
fid=fopen('f:\dtw\ini.dat','w');
fwrite(fid,x,'uchar');
fclose(fid);
%***************************************************************************



%******************************** WORDREC.M ********************************
% This m-file is used to execute cluster analysis and recognise an unknown test word.
% Usage : wordrec('word')
%***************************************************************************
function wordrec(word)

pathout='f:\dtw\'; % sets output path

FIDout=fopen([pathout 'test.dat'],'wb');      % opens file to store test data

  pathin='f:\dtw\hdr\';                       % }defines input path
  extin='.dat';                               % }and file extension

  datak = [word];                             % }finds file with filename
  filename = [pathin word extin];             % }  of form "name.dat"
  if ~exist(filename), break, end             % } (eg. stop.dat) and
  eval(['load ' filename ' -ascii']);         % }extracts data from it for
  data = eval(datak);                         % }      processing



  % create test file for DTWCLUS5.EXE

  len=length(data);                           % }length of data string is already
  fwrite(FIDout,(len-1),'uchar');             % }present in data file
  fwrite(FIDout,data(2:len),'uchar');         % }=>len-1 = length of data so write
                                              % }length and data to training file
end
fclose(FIDout);


%!f:\dtw\dtwclus6.exe                         % runs dtwclus6.exe
%***************************************************************************
```

```
/*********************************** DTWCLUS6.C ****************************/
/*        DYNAMIC TIME WARPING PATTERN RECOGNITION PROGRAM Ver 6        */
/**************************************************************************/

#include <math.h>
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <conio.h>
#include <values.h>

#define CLUSTER_SIZE 5          /* number of trained reference templates per word */
#define REF_MAX 180             /* max size of reference pattern */
#define TEST_MAX 180            /* max size of test pattern */
#define WORDS_MAX 15            /* max size of vocabulary */
#define KNN 3                   /* K nearest neighbour */

void main (int argc, char *argv[])
{
 /* Declare and initialise variables and data structures */
 unsigned char m, n;            /* row and column index */
 unsigned char i, j, k;         /* for loop counters */
 unsigned char N, M;            /* lengths of current test and ref patterns */
 unsigned char nwords, match;
 unsigned int minAD=0;
 unsigned long tmpmin, tmptot;
 float tmpatot,curatot;
 unsigned char index;

 FILE *fref, *ftest, *fini;     /* file pointers for data files*/
 char *testfile;

 struct reference
 {
  unsigned char name[8];
  unsigned char length[CLUSTER_SIZE];
  unsigned char data[CLUSTER_SIZE][REF_MAX];
 } *R[WORDS_MAX];

 struct test
 {
  unsigned char length;
  unsigned char data[TEST_MAX];
 } *T;

 /* Allocate memory required for data structures */
 unsigned char(*LDM)[180]=calloc(180*180,1);
 unsigned int(*ADM)[180]=calloc(180*180,2);
 unsigned int(*PDM)[CLUSTER_SIZE]=calloc(WORDS_MAX*5,2);

 /* Check for command line argument */
 if (argc>=1)
        testfile=argv[1];
 else
        testfile="f:\\dtw\\test.dat";

 /* Check for existance of data files */
 if((ftest=fopen(testfile,"rb"))==0)
```

```
		{
		printf("Error opening test.dat\n");
		exit(1);
		}

	if((fref=fopen("ref.dat","rb"))==0)
		{
		printf("Error opening ref.dat");
		exit(1);
		}

	if((fini=fopen("ini.dat","rb"))==0)
		{
		printf("Error opening ini.dat");
		exit(1);
		}

	/* Get number of words trained to recognise */
	fread(&nwords,sizeof(unsigned char),1,fini);
	fclose(fini);

	/* Get test pattern */
	T=(struct test *)malloc(sizeof(struct test));
	fread(&(T->length),sizeof(unsigned char),1,ftest);
	fread(&(T->data[0]),sizeof(unsigned char),T->length,ftest);
	fclose(ftest);

	/* Get reference patterns */
	for(i=0; i<nwords; i++)
		{
		R[i]=(struct reference *)malloc(sizeof(struct reference));
		fread(&(R[i]->name),sizeof(unsigned char),8,fref);
		for(j=0; j<CLUSTER_SIZE; j++)
			{
			fread(&(R[i]->length[j]),sizeof(unsigned char),1,fref);
			fread(&(R[i]->data[j][0]),sizeof(unsigned char),R[i]->length[j],fref);
			}
		}
	fclose(fref);

	N=T->length;

	clrscr();
	for(i=0; i<nwords; i++)
		{
		for(j=0; j<CLUSTER_SIZE; j++)
			{
			gotoxy(1,1);
			printf("%d -> %d\n",i,j);
			M=R[i]->length[j];

			/* Calculate local distance matrix (LDM) */
			for (n=0; n<N; n++)
				for (m=0; m<M; m++)
					LDM[m][n]=abs((T->data[n])-(R[i]->data[j][m]));

			/* Calculate accumulated distance matrix (ADM) */
				for (n=0; n<N; n++)
```

```
                                        {
                                        for (m=0; m<M; m++)
                                                {
                                                if ((m-1)<0 && (n-1)<0)
                                                        minAD=0;
                                                else if ((n-1)<0 )
                                                        minAD=ADM[m-1][n];
                                                else if ((m-1)<0)
                                                        minAD=ADM[m][n-1];
                                                else
                                                        minAD=min(ADM[m][n-1],min(ADM[m-
1][n],ADM[m-1][n-1]));

                                                ADM[m][n]=LDM[m][n]+minAD;
                                                }
                                        }
                                PDM[i][j]=ADM[M-1][N-1];
                        }
        }


/* Display results */
clrscr();
printf("%s\n",testfile);
printf(" WORD \t 0  \t 1  \t 2  \t 3  \t 4  \t\n");
printf("--------\t------\t------\t------\t------\t------\t\n");
for(i=0; i<nwords; i++)
        {
        printf("%.8s",R[i]->name);
        for(j=0; j<CLUSTER_SIZE; j++)
                printf("\t%6u",PDM[i][j]);
        printf("\n");
        }

/* Decision algorithm */
curatot=MAXLONG;
for(i=0; i<nwords; i++)
        {
        tmptot=0L;
        for (k=0; k<KNN; k++)
                {
                tmpmin=MAXLONG;
                for (j=0; j<CLUSTER_SIZE; j++)
                        if (min(PDM[i][j],tmpmin)==PDM[i][j])
                                {
                                tmpmin=PDM[i][j];
                                index=j;
                                }
                tmptot+=tmpmin;
                PDM[i][index]=MAXINT;
                }
        tmpatot=tmptot/KNN;

        if (min(curatot,tmpatot)==tmpatot)
                {
                curatot=tmpatot;
                match=i;
                }
```

```
          }
 if(curatot>4000)
  {
   printf("\nWORD NOT RECOGNISED!");
  }
 else
  {
   printf("\nPATTERN MATCH IS THE WORD %.8s\n",R[match]->name);
  }

 /* Exit routine */
 for(i=0; i<nwords; i++)
          {
           free(R[i]);
          }
 free(T);
 free(LDM);
 free(ADM);
 free(PDM);
}
/****************************************************************************/
```

```
%********************************** SNDSYNT.M ******************************
% Function to synthesise any speech sound using only information from the first three formants.
%
%Usage : sndsynt(f1,a1,f2,a2,f3,a3,fs)      ,where f(n) and a(n) are the frequency and amplitude
%                                            (in dBs) of the nth formant,
%                                            fs is the sampling frequency.
%***************************************************************************
function f=sndsynt(f1,a1,f2,a2,f3,a3,fs)

t=0:fs;
A1=1;
A2=10^((a2-a1)/20);             % conversion from dBs to decimal
A3=10^((a3-a1)/20);
x=A1*sin(2*pi*f1*t/fs)+A2*sin(2*pi*f2*t/fs)+A3*sin(2*pi*f3*t/fs);       % the synthesised signal
y=25000*x(1:990);          % amplification in order to hear
plot(y);
%pause;
%sound(x,fs);
%pause;
sound(y,fs);
f=y';
%***************************************************************************
```

```
%********************************* ADDNOISE.M ******************************
% This function allows the user to add normalised white noise to a signal.
% Usage : addnoise(sig,vol) ,       where sig is the speech signal
%                                   and vol is the scaling factor of white noise.
%***************************************************************************
function nsig=addnoise(sig,vol)

noise=vol*randn(size(sig));
nsig=sig+noise;

subplot(311),plot(sig),subplot(312),plot(noise),subplot(313),plot(nsig);
sound(sig,11025),sound(nsig,11025),sound(noise,11025);
%***************************************************************************
```

```
%********************************* ADPNOISE.M ******************************
% This function allows the user to add normalised pink (bandlimited white) noise to a signal.
% Usage : addnoise(sig,vol) ,       where sig is the speech signal
%                                   and vol is the scaling factor of pink noise.
%***************************************************************************
function pnsig=adpnoise(sig,vol)

noise=vol*randn(size(sig));
a=[1];
b=[1 4 6 4 1];
pnoise=.125*filter(b,a,noise);      %white noise LPF'ed using a 5th order summation filter
pnsig=sig+pnoise;

subplot(311),plot(sig),subplot(312),plot(pnoise),subplot(313),plot(pnsig);
sound(sig,11025),sound(pnsig,11025),sound(pnoise,11025);
%***************************************************************************
```

```
%********************************** SNDSYNPS.M ******************************
% Function to synthesise any speech sound using only information from the first three formants
% and cotaining non-linear phsae shifts
% Usage : sndsynps(f1,a1,f2,a2,f3,a3,fs)     ,where f(n) and a(n) are the frequency and amplitude
%                                           (in dBs) of the nth formant,
%                                           fs is the sampling frequency.
% The phase shifts are generated by substituting a Sin with a Cos
%*****************************************************************************
function f=sndsynt(f1,a1,f2,a2,f3,a3,fs)


t=0:fs;
A1=1;
A2=10^((a2-a1)/20);           % conversion from dBs to decimal
A3=10^((a3-a1)/20);
x=A1*sin(2*pi*f1*t/fs)+A2*sin(2*pi*f2*t/fs)+A3*sin(2*pi*f3*t/fs);          % synthesised sound x
xp=A1*cos(2*pi*f1*t/fs)+A2*sin(2*pi*f2*(t)/fs)+A3*sin(2*pi*f3*(t)/fs);   % x with a non-
                                                                          %linear phase change
y=x(1:990);
yp=xp(1:990);
subplot(211),plot(y(1:300)),subplot(212),plot(yp(1:300));
%pause;
%sound(x,fs);
%pause;
sound(y,fs);
sound(yp,fs);
f=yp';
%*****************************************************************************


%********************************** CHGPHS.M *********************************
% Function used to insert a non-linear phase delay into a speech signal.
% Usage : f=chgphs(signal)
%*****************************************************************************
function f=chgphs(sig)

l=length(sig);
w=1:l;
e=exp(1);
phs=(cos(10*w/l)+sin(2*w/l));              % severe non-linear function
h=fft(sig);                                 % change to frequency domain
pc=e.^(j*phs);                              % non-linear phase change
hnew=h.*pc';                                % apply to signal
plot(angle(pc),'r'),hold on,plot(abs(pc),'b'),pause,hold off;
plot(angle(h)),pause
plot(angle(hnew))
hrec=ifft(hnew);                            % back to time domain
f=hrec;
sound(sig,18000);
sound(hrec,18000);
%*****************************************************************************
```

BIBLIOGRAPHY.

1. **Abdulla, W.H., Abdul-Karim, A.H.,** '*Pattern Recognition Model to Isolated Word Recognition*', AMSE Review (France), Vol. 14, Iss. 3, pp. 37-52, 1990.

2. **Ainsworth, W.A.,** *Mechanisms of Speech Recognition*, Pergamon Press 1976.

3. **Arai, T., Yoshida, Y.,** '*A Method of Speech Analysis by Zero-Crossings*', ISSPA 92 Third International Symposium on Signal Processing and its Applications Proceedings, Vol. 1, pp. 283-286, (Australia) 1992.

4. **Arai, T., Yoshida, Y.,** '*A Restoration of Speech Signals from Zero-Crossings - A Study of Speech Signal Processing on Zero-Crossings*', Journal of the Acoustical Society of Japan, Vol. 48, Iss. 7, pp. 474-482, July 1992.

5. **Arai, T., Yoshida, Y.,** '*Study on Zero-Crossings of Speech Signals by means of Analytic Signal*', Journal of the Acoustical Society of Japan, Vol. 46, Iss. 3, pp. 242-244, March 1990.

6. **Bangham, J.A.,** '*Median and Morphological Scale Space Filtering and Zero-Crossings*', SPIE - The International Society for Optical Engineering, Vol. 2180, pp. 90-98, 1994.

7. **Basztura, C., Majewski, W., Barycki, W.,** '*Evaluation of the Effectiveness of Parameters of the Global Description*', Archives of Acoustics, Vol. 16, Iss. 3-4, pp. 413-432, 1991.

8. **Basztura, C., Sawczyn, T.,** '*Automatic Speech Signal Segmentation with Chosen Parameterization Method*', Archives of Acoustics, Vol. 18, Iss. 1, pp. 3-15, 1993.

9. **Bleakley, C.J.,** *Notomat Software*, DCU (Dublin).

10. **Brehm, H.,** '*Level-Crossings of Generalised Gaussian Random Processes*', Archiv fur Elektronik und Übertragungstechnik, Vol. 43, Iss. 5, pp. 271-277, Sept.-Oct. 1989.

11. **Burrus, McClellan, Oppenheim, Parks, Schafer, Schuessler,** *Computer-Based Exercises for Signal Processing Using Matlab*, Prentice Hall International Editions 1994.

12. **Coyle, E.D.,** *'Control, Energy Efficiency and Mechanical Redesign of Electric Wheelchairs'*,

13. **Craig, C.S.,** *Goldwave Version 2.10*, ©1993-1994.

14. **de Paor, A.M.,** *'The Sinusoidal Instantaneous Frequency Extractor: a new instrument for use in speech therapy'*, Innovation and Technology in Biology and Medicine, Vol. 13, No. 6, 1992.

15. **Eggebrecht, L.C.,** *Interfacing to the IBM Personal Computer*, Howard W. Sams & Co., 1987.

16. **Erdol, N., Castelluccia, C., Zilouchian, A.,** *'Recovery of Missing Speech Packets using the Short-Time Energy and Zero-Crossing Measurements'*, IEEE Transactions on Speech and Audio Processing, Vol. 1, Iss. 3, pp. 295-303, July 1993.

17. **ESCA,** Tutorial and Workshop on *'Comparing Speech Signal Representation'*, Sheffield University, April 1992.

18. **Friedman, V.,** *'A Zero-Crossing Algorithm for the Estimation of the Frequency of a Single Sinusoid in White Noise'*, IEEE Transactions on Signal Processing, June 1994, Vol.42, Iss.6, pp.1565-1569.

19. **Gabor, D.,** *'Theory of Communications'*, J. Inst. Elec. Eng., (London), Vol. 93, pp. 429-457, 1946.

20. **Gluskin, E.,** *'The Zero-Crossings for System Theory'*, Archiv für Elektronik und Übertragungstechnik, Vol. 45, Iss. 3, pp. 133-141, May 1991.

21. **Hayes, R.,** *'Circuit Theory Notes for Electrical Engineers'*, DIT Kevin Street, Dublin 8, January 1993.

22. **He, S., Kedem, B.,** '*Higher Order Crossings Spectral Analysis of an Almost Periodic Random Sequence in Noise*', IEEE Transactions on Information Theory, Vol. 35, Iss. 2, pp. 360-370, March 1989.

23. **Jian-Xun Xu,** '*A Speech Recognition Model with Time Retrenching and Dual Recognition*', Procs. of IEEE International Conference on Systems, Man, Cybernetics, Vol. 1, pp. 143-146, 1988.

24. **Joseph, E., Pavlidis, T.,** '*Bar Code Recognition using Peak Locations*', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 16, Iss. 6, pp. 630-640, June 1994.

25. **Kedem, B.,** *Time Series Analysis By Higher Order Crossings*, IEEE Press 1994.

26. **King, A.J.,** '*A Map of the Auditory Space in the Mammalian Brain: Neural Computation and Development*', Exp. Physiol. (England), Sept. 1993.

27. **Koch, R.C.N., Prasad, R., Bons, J.H.,** '*An Asymmetric Speech Coding Algorithm using Vector Quantization and Silence Detection*', European Transactions on Telecommunications and Related Technologies, Vol. 4, Iss. 5, pp. 539-546, Sept.-Oct. 1993.

28. **Kronland-Martinet, R., Morlet, J., Grossmann, A.,** '*Analysis of Sound Patterns Through Wavelet Transforms*', International Journal of Pattern Recognition and Artificial Intelligence, Vol.1 No.2, 1987.

29. **Kun-Shan Lin,** *Digital Signal Processing Applications with the TMS 320 Family Vol. 1*, Prentice Hall and Digital Signal Processing Series Texas Instruments 1987.

30. **Lawlor, R., Grimson, W., Hayes, R., Rochford, R.,** '*Time-Frequency Signal Analysis*', Irish DSP and Control Colloquium (Dublin City University), 1994.

31. **Lipovac, V.,** '*Efficient Zero-Crossing-Based Autoregressive Feature Extraction for Speech Recognition*', Alto Frequenza, Vol. 57, Iss. 8, pp. 501-504, Oct.1988.

32. **Lipovac, V.,** '*Zero-Crossing-Based Linear Prediction for Speech Recognition*', Electronics Letters, Vol. 25, Iss. 2, pp. 90-92, Jan. 1989.

33. **Lovell, B.C.,** '*Techniques for Non-Stationary Spectral Analysis*', Dept. of Elect. Eng., University of Queensland, Australia.

34. **Lovell, B.C., Williamson, R.C.,** '*The Statistical Performance of Some Instantaneous Frequency Estimators*', IEEE Transactions on Signal Processing, Vol. 40 No. 7, July 1992.

35. **Majewski, W.,** '*Selected Statistics of Polish Speech Signals*', Przeglad Telekomunikacynjy, Vol. 63, Iss. 1-2, pp. 22-27, (Poland), 1990.

36. **Malifax Computers (S) Pte Ltd.,** *Sound Vision 16 Gold.*

37. **Markel, J.D., Gray, A.H.,** *Linear Prediction of Speech*, Springer-Verlag 1982.

38. **Marks, J.A.,** '*Real Time Speech Classification and Pitch Detection*', COMSIG '88 [Proceeding on South African Conference on Communications and Signal Processing], 1988.

39. **Maruno, S., Schoenberg, A.,** '*Voice Command Recognition System for Handicapped Persons using Simple Multi-layered Networks*', Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Vol. 4, pp. 1590-1591, 1988.

40. **Math Works**, The, Inc. (© 1992).

41. **Microsoft Corporation,** (1985-1993).

42. **Microsoft Corporation,** (1992-1994), *Encarta '95.*

43. **Network 7,** *Beyond 2000,* (Australian TV).

44. **Nuzum, S., Davitt, A.,** '*Feature Extraction of Speech using Formant Tracking in a View to Obtaining a Recognisable Pattern*', DIT Kevin Street, 1995.

45. **O'Shaughnessy, D.,** *Speech Communication - Human and Machine*, Addison-Wesley 1990.

46. **Owens, F.J.,** *Signal Processing of Speech*, Macmillan 1993.

47. **Petsche, H., Etlinger, S.C., Filz, O.,** '*Brain Electrical Mechanisms of Bilingual Speech Management: An Initial Investigation*', Electroencephalogr. Clin. Neurophysiol. (Ireland), June 1993.

48. **Pickles, J.O.,** *An Introduction to the Physiology of Hearing*, Academic Press 1982.

49. **Poon, P.W., Brugge J.F.,** '*Virtual-Space Receptive Fields of Single Auditory Nerve Fibres*', J Neurophysiol (US), Aug. 1993.

50. **Price, C., Wise, R., Ramsey, S., Friston, K., Howard, D., Patterson, K., Frackowiak, R.,** '*Regional Response to Differences within the Human Auditory Cortex when Listening to Words*', Neurosci Lett (Netherlands), Nov. 1992.

51. **Quinn, A.,** '*Development of the Sinusoidal Instantaneous Frequency Extractor (SIFE) Based on a New Piecewise Sinusoidal Model (PSM) of Speech*', UCD (Dublin), 1988.

52. **Reddy, D.R.,** *Computer Recognition of Connected Speech*, J. Acoust. Soc. Am. 42, 329, (1967).

53. **Rosner, B.,** *Fundamentals of Biostatistics*, PWS-Kent 1990.

54. **Sakoe, H., Chiba, S.,** '*Dynamic Programming Algorithm Optimisation for Spoken Word Recognition*', IEEE Trans. A.S.S.P., Vol. 26, pp. 43-49, February 1978.

55. **Santhoshkumar, U., Anto, B.P., Sridhar, C.S.,** '*Digit Recognition using Energy Envelope and Zero-Crossing Rate*', TENCON '89 - Information Technologies for the 90's, IEEE 1989.

56. **Savoji, M.H.,** '*A Robust Algorithm for Accurate Endpointing of Speech Signals*', Speech Communication, Vol. 8, Iss. 1, pp. 45-60, March 1989.

57. **Schreiner, C.E., Mendelson, J.R., Sutter, M.L.,** '*Functional Topography of Cat Primary Auditory Cortex: Representation of Tone Intensity*', Exp. Brain Res. (Germany),1992.

58. **Shaw, M.J., Honary, B., Darnell, M.,** '*Higher Order Crossing Analysis Applied to Signal Detection and Evaluation of Radio Channels*', ICAP '89, Vol.2, No.301, pp. 68-74, IEE 1989.

59. **Soon Hyob Kim, Kyu Tae Park, Bovik, A.C.,** '*Recognition of Korean Isolated Digits using a Pole-Zero Model*', Journal of the Korean Institute of Telematics and Electronics, Vol. 25, Iss. 4, pp. 356-365, April 1988.

60. **Stearns, S.D., Hush, D.R.,** *Digital Signal Analysis*, Prentice-Hall International 1990.

61. **Takamatsu, R., Kimura, K., Sato, M., Kawarada, H.,** '*Pitch Determination with Scale Space Filtering*', Bulletin of Precision and Intelligence Laboratory , Iss. 66, pp. 1-7, Sept. 1991.

62. **Taniguchi, I., Nasu, M.,** '*Spatio-Temporal Representation of Sound Intensity in the Guinea Pig Auditory Cortex observed by Optical Recording*', Neurosci Lett (Netherlands),March 1993.

63. **Texas Instruments,** *Digital Signal Processing Applications with the TMS 320 Family*, TI Inc. 1986.

64. **Texas Instruments,** *Linear Circuits Data Book Vol. 2*, TI Inc. 1992.

65. **Texas Instruments,** *TMS320C2X User's Guide*, TI Inc. 1993.

66. **Tong, Y.C., Chang, J.S., Harrison, J.M., Hugien, J., Clark, G.M.,** '*Two Speech Processing Schemes for the University of Melbourne Multi-Channel Cochlear Implant Prosthesis*', IEEE International Symposium on Circuits and Systems, 1989.

67. **Tortora, G.J.,** '*Introduction to the Human Body*', Harper-Collins, 1991.

68. **Venkatesh, Y.V.,** '*Hermite Polynomials for Signal Reconstruction from Zero-Crossings*', IEE Proceedings I [Communications, Speech and Vision], Vol. 139, Iss. 6, pp. 587-596, Dec. 1992.

69. **Weiss, V.**, '*The Relationship Between Short-Term Memory Capacity and EEG Power Spectral Density*', Biological Cybernetics, Vol. 68, Iss. 2, pp. 165-172, 1992.

70. **Weng, J.**, '*Windowed Fourier Phase: Completeness and Signal Reconstruction*', IEEE Transactions on Signal Processing, Vol. 41, Iss. 2, pp. 657-666, Feb. 1993.

71. **Wever, E.G.**, *Theory of Hearing*, Dover Publications 1970.

72. **Woldorff, M.G., Gallen, C.C., Hampson, S.A., Hillyard, S.A., Pantev, C., Sobel, D., Bloom, F.E.**, '*Modulation of Early Sensory Processing in Human Auditory Cortex during Auditory Selective Attention*', Proc. Natl. Acad. Sci. USA, Sept. 1993.

73. **Wong, C.W., Mosttafavi, M.T.**, '*The Recognition of Isolated Words on a Speaker Dependent System*', SOUTHEASTCON '89 Proceedings - Energy and Information Technologies in the Southeast (USA), Vol. 2, pp. 755-758, April 1989.

74. **Woolfson, M.S.**, '*Study of Cardiac Arrhythmia using Zero-Crossing Analysis*', Journal of Biomedical Engineering, Vol. 11, Iss. 4, pp. 303-310, July 1989.

75. **Yanick, P., Freifeld, S.**, *The Application of Signal Processing Concepts to Hearing Aids*, Grune & Stratton 1977.

76. **Young, K., Sackin, S., Howell, P.**, '*The Effects of Noise on Connected Speech: A Consideration for Automatic Speech Processing*', Visual Representations of Speech Signals - Cooke et al., Wiley & Sons Ltd., 1993.