ML SULTAN TECHNIKON

HARDWARE IMPLEMENTATION OF AN AUTOMATIC SPEAKER RECOGNITION SYSTEM USING ARTIFICIAL NEURAL NETWORKS

by

Viresh Moonasar

March 2002

ML SULTAN TECHNIKON

HARDWARE IMPLEMENTATION OF AN AUTOMATIC SPEAKER RECOGNITION SYSTEM USING ARTIFICIAL NEURAL NETWORKS

by

Viresh Moonasar

(BTech. Electrical Engineering)

Submitted in fulfillment of the academic requirements for the degree of Master of Technology in Electrical Engineering in the Department of Electronic Engineering, Faculty of Engineering, ML Sultan Technikon of Durban in South Africa, March 2002.

ABSTRACT

The use of speaker recognition technology in interactive voice response and electronic commerce systems has been limited. This is due to the lack of research attention and published results when compared to all the other areas of speech recognition technologies.

This thesis describes the Digital Signal Processor (DSP) hardware-based implementation of an automatic speaker recognition system using artificial neural networks. A supervised learning vector quantization neural network is used as the pattern classifier. Linear predictive coding and cepstral signal processing techniques are utilized to form hybrid feature parameter vectors as inputs to the pattern recognition system. The practical application of a committee of neural networks for pattern recognition rather than the conventional single-network decision system is also implemented.

The entire system is developed on the target system, incorporating a Texas Instrument's TMS320C32 floating point DSP processor. This enables real-time performance, which is an afterthought on host PCs working under Windows and other operating systems. The code can now be embedded onto a stand-alone target system independent of the host platform.

The results of both the MATLAB-based implementation and the DSP-based implementation for the speaker recognition system are presented in this thesis. The impact of using a committee of neural networks, for the task of pattern recognition, on the recognition accuracy of the speaker recognition system is also presented

ACKNOWLEDGEMENTS

The work presented in this thesis was carried out under the supervision of Dr G.K. Venayagamoorthy. My gratitude goes to him for his assistance, contributions, coordination, supervision, guidance and acquisition of resources for the project.

I also wish to thank:

- Professor F. Takawira from the University of Natal, Durban for his suggestions and efforts in proof-reading this thesis.
- The National Research Foundation (NRF) for their financial assistance.
- All speakers who contributed their voice samples to be added to the speech database.
- Telkom-Ericsson-THRIP Centre of Excellence (COE) for sponsoring the DSP equipment used to carry out this work.
- Mr S. Gounden for his input, and assistance with the DSP hardware.
- Finally a special note of thanks to my family for their understanding, assistance and support.

TABLE OF CONTENTS

CHAPTER 3 THEORY OF NEURAL NETWORKS

CHAPTER 4 MATLAB IMPLEMENTATION AND RESULTS OF THE SPEAKER RECOGNITION SYSTEM

CHAPTER 5 HARDWARE IMPLEMENTATION AND RESULTS OF THE SPEAKER RECOGNITION SYTEM

CHAPTER 6 CONCLUSION

APPENDIX A SOUND FILE FORMAT

APPENDIX B MATLAB IMPLEMENTATION MEMORY CONSTRAINTS

APPENDIX C CHARACTERISTIC FEATURE PARAMETER PROGRAM CODE

APPENDIX D PSD CROSS-CORRELATIONS

APPENDIX E MATLAB IMPLEMENTATION NEURAL NETWORK CODE

APPENDIX F COMMITTEE OF NEURAL NETWORK RESULTS (MATLAB IMPLEMENTATION)

APPENDIX G ADC64 SCHEMATICS

APPENDIX H DSP IMPLEMENTATION SOURCE CODE

APPENDIX I MEMORY ALLOCATION OF THE TARGET SYSTEM

REFERENCES

LIST OF FIGURES AND TABLES

FIGURES

TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS

ADC	Analogue to Digital Converter
ANN	Artificial Neural Network
ANNIE	Artificial Neural Networks In Engineering
ANSI	American National Standards Institute
ARPA	United States Advanced Research Project Agency
ASR	Automatic Speech Recognition
CCEPS	Complex Cepstrum
CD	Compact Disk
DAC	Digital to Analogue Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FIFO	First In First Out
FFT	Fast Fourier Transform
I/P	Input
IJCNN	International Joint Conference on Neural Networks
KBPS	Kilobits per second
LPC	Linear Predictive Coding
LVQ	Linear Vector Quantization
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
O/P	Output
PC	Personal Computer
PPM	Parts Per Million
PSD	Power Spectrum Density
SLP	Spoken Language Processing
SNR	Signal to Noise Ratio

LMS	Least Mean Squares
SOM	Self Organizing Map
HMM	Hidden Markov Model
GMM	Gaussian Mixture Model
DTW	Dynamic Time Warping
MLP	Multi-Layer Perceptron
GSF	Generalized Feature Set
EM	Expectation Maximization
ML	Maximum Likelihood
MEM	Maximum Entropy Method
SSE	Sum Squared Error goal
lr	Learning rate
PPM	Parts Per Million
TI	Texas Instruments
COFF	Common Object File Format

SYMBOLS

****	Backslash or left division
B	Number of Bits
F_{max}	Maximum Frequency
Fs	Sampling Frequency
NFFT	Length of FFT
P	Input Matrix
Pxx	PSD of a signal
q	Quantisation Step Size
T	Target Vector
V_{fs}	Full-Scale (Range) Voltage

CHAPTER 1

INTRODUCTION

1.1 Introduction

Technological innovations and the information technology era have created a huge need for on-line security. The reluctance of on-line shoppers using their credit cards over the Internet has been a major factor for the slow take-off of e-commerce. Speaker recognition, which is the process of automatically recognising who is speaking based on unique information inherent in speech signals, is a method that may be adopted to enhance security over the Internet and other security applications. With the development of telecommunications and acoustic recordings, the need for speaker recognition has become more important.

1.2 Objectives

This Master of Technology thesis is the result of work continued from January 1999. This thesis was preceded by a Bachelor's thesis [1] completed in November 1998, which showed that automatic speaker recognition could be achieved using Artificial Neural Networks. More specifically, the study in [1]

shows a success rate of ninety percent for a five-subject group using the Linear Vector Quantization (LVQ) neural network.

The principle objective of this work is to investigate the practical considerations when implementing an automatic speaker recognition system using single and committee LVQ neural networks on a real-time target system that is independent of any host Personal Computer (PC). The hardware chosen for this purpose was Innovative Integration's ADC64 data acquisition card. The ADC64 incorporates a Texas Instruments TMS320C32 Digital Signal Processor (DSP), which is the ultimate target system.

Another objective is to investigate increasing the group size (number of subjects in test group) while still maintaining the success rate of the system. The recognition rate of the original system deteriorated drastically as the group size was increased. The group size is now extended to twenty subjects (while still maintaining the ninety- percent success rate). This was achieved by using a hybrid technique to derive the feature parameters used to identify each speaker. Chapter Two describes this in great detail.

In addition to achieving the objective specified above the following was also accomplished:

The use of a committee of neural networks rather than a single-network pattern recognition system was realised to improve the recognition accuracy of the twenty-subject system from ninety to ninety-five percent. The committee network has also been implemented on the hardware system.

The ability of the system is also evaluated using the YOHO speaker database.

1.3 Thesis Overview

This chapter is a general introduction to the methodology and results presented in the following chapters. It has outlined the main objective of this study as well as additional achievements that were realised e.g. expanding the group size to twenty subjects. It also presents a brief conclusion of the work documented.

Chapter Two is titled "Speaker Recognition Systems". This chapter discusses details of the basic building blocks and terminology associated with any speaker recognition system. It also contrasts speech recognition and speaker recognition. These two concepts are commonly confused. The aim of speech recognition is to determine the actual words spoken irrespective of who is speaking.

Speaker identity is correlated with the physiological and behavioural characteristics of the speaker. These characteristics exist both in the spectral envelope (vocal tract characteristics) and in the supra-segmental features (voice source characteristics and dynamic features spanning several segments) [2]. Feature parameter extraction involves extraction of these unique parameters. This is a key aspect of the recognition system and determines it's accuracy. Extracted feature parameters such as Linear Predictive Coding and cepstral analysis are also discussed in Chapter Two.

Chapter Three outlines the use of neural network architectures for pattern recognition. It presents the architecture of the Learning Vector Quantization neural network as well as the concept of using a committee of neural networks rather than a single-network in decision-making systems.

Chapter Four presents the simulation techniques and the results obtained. These simulations have determined the methods and algorithms used during the eventual hardware implementation onto the target system. The MATLAB implementation is also described in Chapter Four.

The hardware implementation onto the TMS320C32 DSP target system is presented in Chapter Five. This has enabled real-time performance, which is an afterthought on host PCs working under Windows and other operating systems. The complete ANSI C source code used to generate the eventual machine code is also included in Appendix H.

Finally, the conclusions drawn from this application are presented in Chapter Six. Possible future directions of research in this field are also proposed.

1.4 New Contributions

The work in this thesis is an extension to a Bachelor of Technology thesis [1]. It makes the following new contributions:

(a) It develops a hybrid feature parameter, which are the cepstral coefficients calculated from the Linear Prediction Coefficients (LPCs). The success rate of the system is improved with the use of this hybrid feature parameter.

(b) It increases the group size of the system whilst maintaining the success rate of the system. This is enabled by the use of the hybrid feature parameter described above.

(c) It shows, via MATLAB simulations that the success rate of the system can be further improved with the use of a committee of neural networks rather than a single neural network. This is confirmed by the results achieved when the committee is implemented in hardware.

(d) It goes on further to show that the entire committee-based speaker recognition system can be implemented on a real-time hardware target system.

(e) It also proves that it is possible to implement speaker recognition using neural networks on a target system.

1.5 Research Publications

Three international [3, 4, 5] and two national [6, 7] conference papers have resulted from the work of this thesis:

In October 1999 a paper outlining the input of a combination of the Linear Predictive Coding (LPC) and Power Spectrum Density (PSD) feature parameters, rather than a single feature parameter, into the pattern recognition network was presented at the Africon '99 conference held at the Cape Technikon [3].

The hybrid feature parameter technique was evolved to include the Cepstral analysis of the LPC coding. This enabled the group size to be extended to twenty subjects and still maintain a ninety percent recognition success rate. This paper was presented at the Artificial Neural Networks In Engineering (ANNIE) conference held in St. Louis, Missouri, USA in November 2000 [4].

The success rate of ninety percent was further increased to ninety-five percent with the introduction of a five-member LVQ committee neural network. The group size of twenty was still maintained. This paper was published in the proceedings of the International Joint Conference on Neural Networks (IJCNN) held in Washington DC, USA in July 2001 [5].

1.6 Summary

The type and method of feature parameter extraction is crucial to the success of the system. The LPC-Cepstral hybrid parameter has produced optimal results. The use of the LVQ classification ANN has been successfully applied to the speaker recognition application. It was showed that the LVQ classification neural network is better suited for speaker recognition application than the popular Multi-Layer Perceptron (MLP) neural network. Finally, all the above simulation results were verified on a real-time hardware implementation on the Texas Instruments TMS320C32 DSP.

CHAPTER 2

SPEAKER RECOGNITION SYSTEMS

2.1 Introduction

Speaker recognition, which can be classified into identification and verification, is the process of automatically recognising who is speaking on the basis of individual information included in speech waves. This technique makes it possible to use the speaker's voice to verify their identity and control access to services such as voice dialling, banking by telephone, telephone shopping, database access services, information services, voice mail, security control for confidential information areas, and remote access to computers. In this way, speaker recognition technology is expected to create new services that will make our daily lives more convenient. Another important application of speaker recognition technology is its use in forensic purposes [8].

The term "automatic speaker recognition" needs to be justified since certain applications, e.g. forensic investigations, manually analyse individual segments of recorded speech to identify the speaker.

This chapter describes the application of Automatic Speaker Recognition, which is a division of Spoken Language Processing (SLP) discussed in section 2.2. The key aspect of any ASR system is the extraction of the characteristic features, or feature parameters, from the voice samples that indicate the specificity of the speaker. This is the real challenge of the recognition system. To date, no serious

scientific protocol has been able so far to evidence the existence of a fixed, robust, non-modifiable, individual voice characteristic that could be extracted from a speech signal and indicate without doubt the speaker's identity.

2.2 Spoken Language Processing

The ability to converse freely with a machine remains the ultimate challenge to our understanding of human perception processes. For example, interactive networks, using spoken language input, can provide easy access to information and services that are still limited to those people who can read and write. This section briefly describes the various natural language processing techniques. Figure 2.1 shows the spoken language interfaces (input and output).

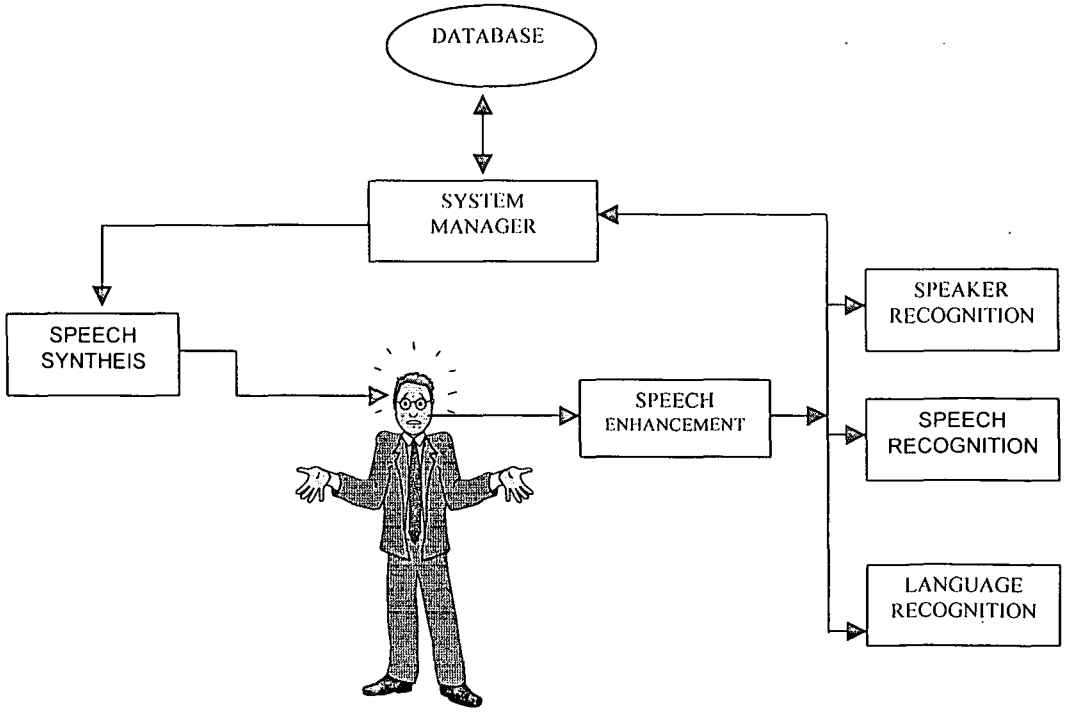


Figure 2.1- Spoken language interfaces

Spoken Language Processing (SLP) may be roughly divided into the following classes since it concerns many different technologies and applications:

- ASR - Automatic Speech Recognition
- ASG - Automatic Speech Generation
- Speech input-output (understanding)
- General Speech Processing
- Speech analysis / Paralinguistic processing

2.2.1 Automatic speech recognition

Speech recognition is commonly referred to as voice recognition. It should therefore not to be confused with speaker recognition. The aim here is to recognize what words are being spoken i.e. conversion of an acoustic signal into a stream of words. It involves the input of human speech samples into a computer, which identifies the speech. This is a very active area of research with good success rates being achieved recently. Typical applications include cellular voice dialing e.g. "Call home".

Speaker dependent and speaker independent systems are available. The system is trained to recognise speech from only one speaker in a speaker dependent system but can recognise speech from many different speakers in a speaker-independent system [9]. This is irrespective of the certain factors of variability e.g. possible foreign accent. By definition, a speaker dependent system is designed for use by only a single speaker. Conversely, a speaker independent system is designed for use by any speaker. For both multi-speaker and speaker dependent systems, a large increase in accuracy can be obtained by *adapting* the system to a specific speaker during operation. These systems are known as *speaker-adaptive* and work by tuning themselves to a specific speaker during operation.

There are many external sources that affect the performance of the speech recognition system. These include characteristics of the environmental noise and the type and placement of the microphone. This is a difficult problem due to the numerous sources of variability associated with the signal. Phonetic and acoustic variability (across speakers and within the same speaker) result from the speaker's physical and emotional state, physiological and social background as well as his speaking rate. Speech recognition networks will have to de-emphasise these speaker dependent characteristics [14]. Table 2.1 illustrates the characteristics that indicate the capabilities of a speech recognition system.

Table 2.1 - Capabilties of speech recognition systems

Characteristic	Range
SNR	10 dB < > 30 dB
Input Transducer	High quality microphone to noisy telephone
Speaker Dependence	Speaker-dependent or Speaker-independent
Vocabulary	From a couple of words to a large database (> 2000 words)
Capability	Isolated word to continuous speech

Substantial progress has been made with speech recognition systems. This has led to the lowering of barriers to speaker independence, continuous speech and large vocabularies. These steady improvements have led to the deployment of systems within many telephone and cellular networks.

2.2.2 Automatic speech synthesis / generation

Automatic Speech Generation (ASG) - words are 'spoken' by computer. This is the synthesis of speech by a machine. Speech production can be viewed as an acoustic filtering operation. The vocal and nasal tracts represent the acoustic filter [12]. Voiced speech is modeled as a periodic pulse train, filtered by the glottal pulse model (vocal tract model). Unvoiced speech is modeled by random noise filtered by the vocal tract.

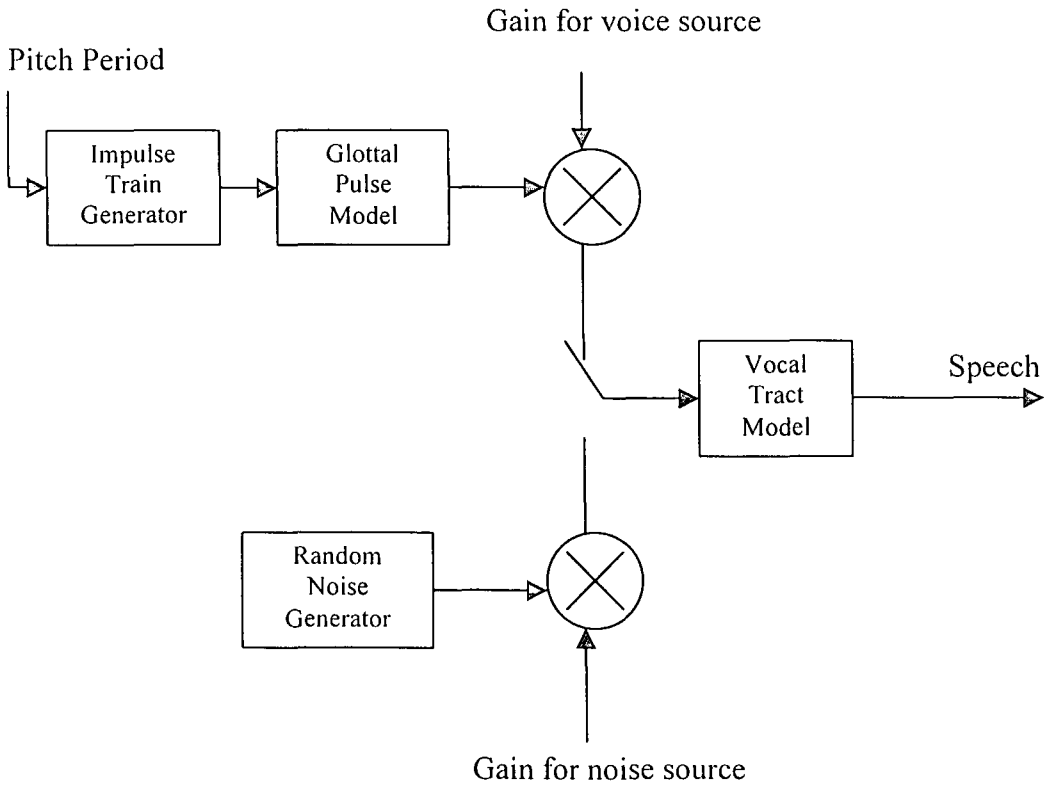


Figure 2.2 - Electric filter model for speech production [10]

2.2.3 Speech input and output

This is also termed speech understanding. The computer reacts to speech input. This type of interactive application is quite challenging. The solution involves combination of several spoken language technologies to achieve this goal i.e. speech recognition to identify what is required, possibly language understanding to interpret the command and speech synthesis to acknowledge action to be taken. Lexical knowledge, i.e. an understanding of the individual words of the language is necessary for a language processing system.

2.2.4 General speech processing

This includes the enhancement, storage and distribution of speech.

Speech compression

The aim of speech compression is to minimize the number of bits used to represent the signal without a significant degradation in signal quality. Speech coding has been classified into two categories: vocoders and waveform coding. Waveform coding aims to preserve the speech waveform while vocoders preserve only the spectral properties of the signal.

Figure 2.3, taken from [8], shows an opinion of the various speech qualities obtained with the different bit rates. The quality ranges from 0 to 5. A rating of 5 reflects excellent quality. Compact Disc (CD) quality music uses a bit rate greater than 44 Kbps. Normal speech is sampled at a minimum of 8 Kbps.

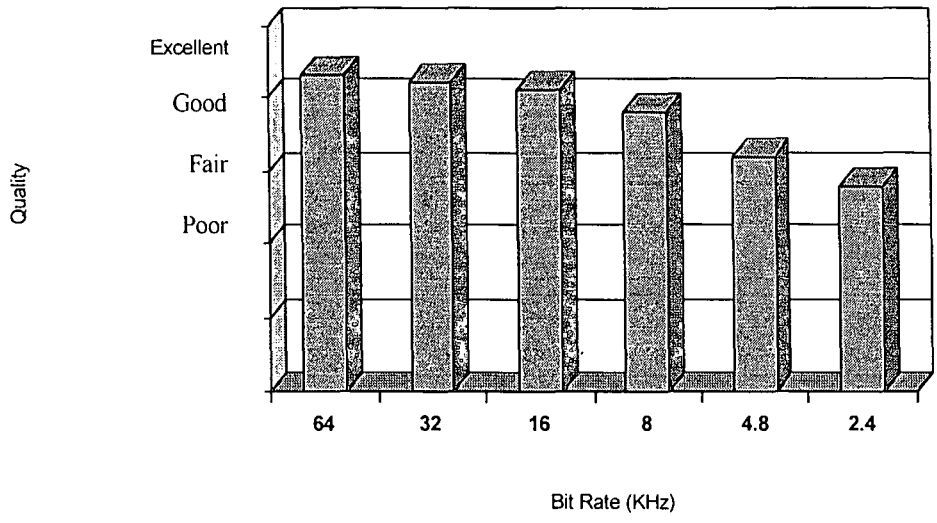


Figure 2.3 - Compressed speech quality

The storage and transmission of speech has also created the need for the compression of speech. This is particularly important in applications where there is a limited bandwidth available for transmission. Compression techniques are also advantageous when there is limited storage availability e.g. Huffman coding of prediction residuals has been used for storage of large speech databases.

Speech enhancement

Noise contaminated speech signals need to be enhanced such that intelligible speech may be extracted from the message. The suppression of background noise is a factor that affects the performance of both speech and speaker recognition systems. Therefore speech signal enhancement needs to be applied to speaker recognition application as well e.g. the use of an array of microphones rather than a single transducer.

One of the hardest problems to overcome is background noise management i.e. the art of listening in the presence of noise. Noise management algorithms require extensive amount of processing power and data memory to implement e.g. adaptive noise cancellation deals with the problem of removing correlated noise (noise that has some redundancy associated with it).

Another technique that may be used to counter noise is the use of a microphone array for acoustic beam forming. The array can focus in a specific direction and cancel noise from all other directions. Also, noise templates may be created and used to ignore the noise component of the signal.

2.2.5 Speech analysis / paralinguistic processing

Automatic speaker recognition forms part of this discipline. It is sometimes required to establish the identity of the speaker and not the underlying linguistic content. Speaker recognition will be elaborated in greater detail in the section on typical speaker recognition systems. This is our focus area of research.

2.3 The Human Vocal Tract and Speech Production

The main components of human physiology that determine the speech waveform are the lungs, trachea, larynx, pharyngeal cavity (throat), oral cavity (mouth) and the nasal cavity [8]. The cross-sectional area of the vocal tract is non-uniform and is determined by the position of the articulators, which comprise the lips, jaw, tongue and velum. The velum movements control the coupling of the nasal and vocal tracts. For non-nasal sounds, the velum is drawn up towards the back of the pharyngeal cavity, closing the entry to the nasal tract. The vocal tract may be viewed as a set of resonating cavities bounded by anatomical structures, which are either fixed or movable.

Sound is generated in several ways and at several locations in the human vocal tract. The most common sound generation sources are the quasi-periodic vibration of the vocal cords and turbulent noise generated by the passage of air through a narrow constriction, usually in the oral cavity. More rarely, sounds are generated by plosive releases of air (following the build-up of pressure behind an obstruction in the vocal tract), implosion (following the creation of a vacuum behind an obstruction in the vocal tract) and clicks created by, the action of the tongue pulling away from the roof of the mouth.

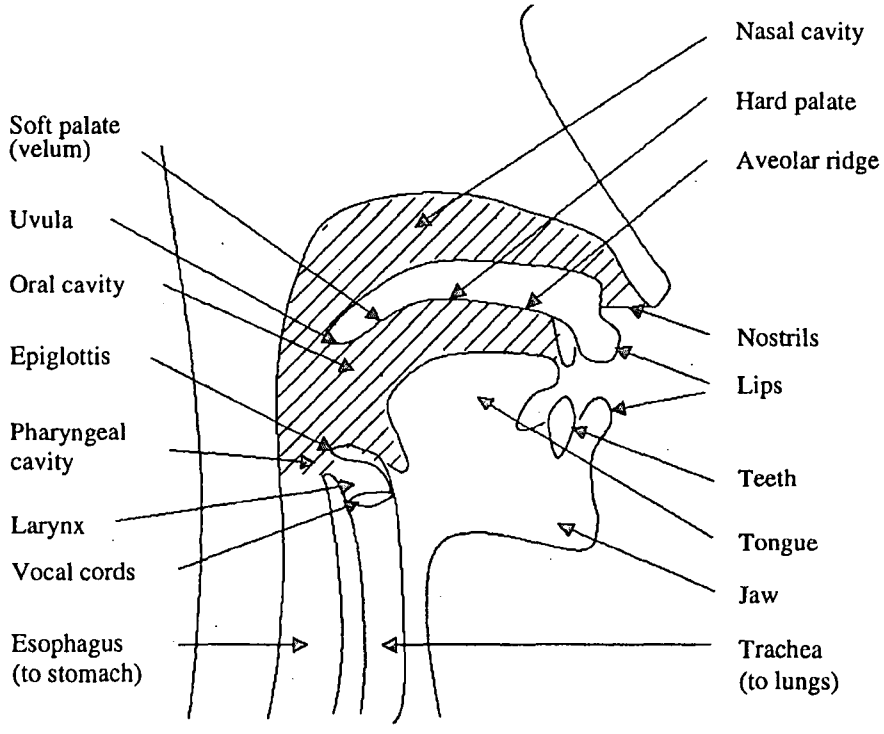


Figure 2.4- Physiology of human speech production

Wherever generated, the sounds underlying speech are relatively unstructured. For instance, the buzz-like sound of vocal cord vibration has a relatively simple spectral properties with harmonics at frequencies corresponding to integer multiples of the fundamental frequency. Similarly, friction noise generated by turbulence has a relatively broad frequency distribution with a somewhat high pass characteristic.

Despite the simple characteristics of the sound sources used in speech, the speech signal itself is complexly structured in both frequency and time. This structure derives from the response characteristics of the vocal tract with resonances (poles) and anti-resonances (zeroes) located at frequencies determined by a variety of factors, but primarily the length and cross-sectional area of the vocal tract above the location of the sound source. The signal is further structured in time by the motions of articulators, which constantly effects changes in the vocal tract response characteristics.

The phoneme is known as the basic unit of the spoken language [8]. More precisely, the phoneme is the smallest segment of sound. In the literature, one finds suggestions that some phonemes, like nasals or some vowels are the best for speaker recognition [10].

2.4 The Basis for Automatic Speaker Recognition

The basis for automatic speaker classification and recognition is that in addition to the linguistic message, the human voice conveys a lot of paralinguistic information about the speaker, i.e.. The "encoder". These factors of variability are well-known obstacles to speech recognition, as they increase the variability of the speech signal [13].

The main sources of a speaker's specificity are the physiological configuration of his speech production organs, his neuro-motor control of these organs, and his internal speech pattern prototypes. In practice, there may exist more or less systematic correlation's between these factors and some of the speaker's characteristics, such as his sex, age, health conditions, mood, regional, cultural, educational background, possible foreign accent and the language he is speaking.

2.5 Speaker Identification and Speaker Verification

Speaker recognition covers two different areas: speaker identification and speaker verification. The goal of a speaker identification task is to classify an unlabelled voice token as belonging to one of a set of *n* reference speakers. This is applied mostly in forensic investigations. The speaker verification task is to decide whether or not the unlabelled voice belongs to a specific reference speaker through comparison of some samples of his speech. Is the speaker who he claims to be? We want to verify the claimed identity of the speaker. The only information available is a set of known utterances from each subject of the group. Verification is concerned with validating the claimed identity of a person from an open set. Verification systems also reject subjects known as impostors. Applications in which a voice is used as the key to confirm the identity of a speaker are classified as speaker verification.

Since most systems can easily be deceived by the playback of a recording, most systems prompt the user for a password (key words that are randomly used every time the system is used) which is used to determine whether the speaker is an impostor. The speaker verification task can now be performed after it has been established that the subject is a member of the group. Proposals for the combination of the above tasks into a single solution where the impostor can be determined without passwords have been included in this thesis.

2.6 The Basic Speaker or Speech Identification System

The system is trained to recognize a person's voice by each person speaking out a specific utterance (text-dependent) into the transducer (microphone). The speech signal is digitized and some signal processing creates a feature parameter vector template for the voice pattern that is then stored in memory. The system recognizes a speaker by comparing a test utterance with the

respective templates stored in the memory. When a match occurs the speaker is identified.

The two important operations in an identifier are parameter extraction, where distinct patterns are obtained from the utterances of each person for stored templates, and pattern matching, where the test utterance is compared with the templates stored in memory. Correlation techniques are usually employed for pattern matching. Neural networks have been used for this purpose in these studies. Neural networks are discussed in Chapter Three.

Figure 2.5 illustrates the components of a speaker verification system. The ADC digitizes the analog input utterance. Characteristic feature parameters are extracted and either stored as templates or compared with the templates already in the memory area of the system. The system then outputs the closest match of the input with the templates in memory (test mode only).

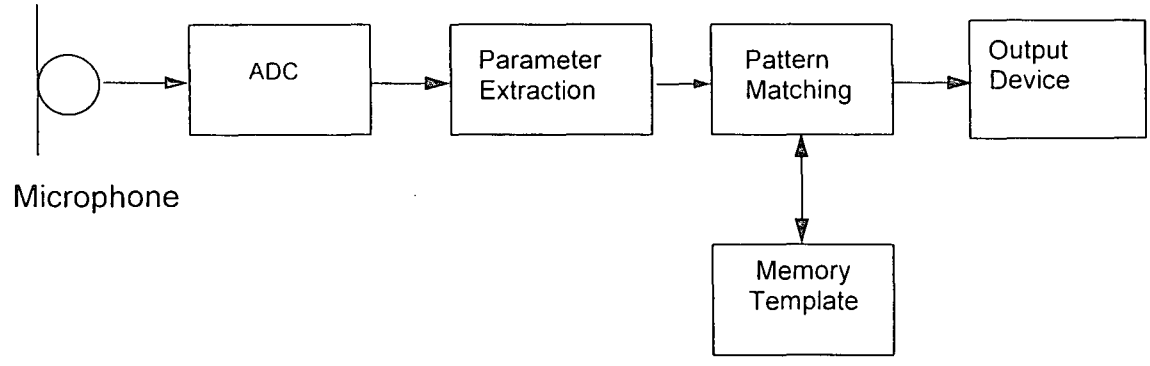


Figure 2.5 - Basic components of a speaker verification system

Parameter extraction is discussed in section 2.9 and later in Chapter Four. A neural network accomplishes the pattern matching responsibilities of the system. Neural networks are the topic of discussion of Chapter Three. A complete practical system is simulated and implemented in Chapter Four and Chapter Five respectively.

2.7 Modes of Operation of a Speaker Recognition System

Any speaker classification or recognition system functions in three modes:

i) Training Mode
ii) Test Mode
iii) Un-training Mode

2.7.1 Training mode

During which speaker or speaker class models are built, estimated and stored in the system as reference patterns. Alternative terms: learning mode, registration, enrolment, subscription, etc.

2.7.2 Test mode

During which the system performs the recognition of an utterance to be identified (or verified). The input signal is processed and the extracted feature parameter characteristics are correlated with the templates stored in the system memory. Alternative terms: recognition mode, trial mode, operating mode, etc.

2.7.3 Un-training mode

During which a speaker or speaker class model is removed from the list of reference patterns.

2.8 Factors Affecting the Performance of the Speaker Recognition System

- Channel difference between training and test set
- Background and channel noise
- Speaker variability (volume, emotional state, speaking rate)
- Amount of training data
- Time interval between training and testing
- Type and placement of the microphone
- Group Size

The number of subjects in the group is naturally an important factor when assessing the success of a speaker recognition system. Thus, it is easier to obtain a high accuracy for a small group (less than five speakers). As the size of the group is increased, it becomes increasingly difficult to maintain this high accuracy.

2.9 Signal Representation and Characteristic Feature Extraction

Speaker identity is correlated with characteristics that exist both in the spectral envelope (vocal tract characteristics) and in the speech segmental features (voice source features spanning several segments) [18].

Speaker recognition can be achieved by comparison of the raw speech waveforms. The problem with this is the huge amount of computational resources that will be required by the recognition system. If training samples are restricted to three second recording intervals and the number of training utterances per speaker are limited to ten then the system storage resources required will be approximately 192000000 bits (8 bit resolution and minimum sampling rate of

8000 Hz). There will also be a huge strain placed on the pattern-matching component of the system (neural network).

Feature extraction is therefore needed to extract parameters that will represent the spectral and supra-segmental characteristics of the original speech waveform adequately. This is a real problem area. Speaker identification can be attempted in a variety of ways using widely different sets of features. Some of the methods that have been considered for feature extraction include the following:

- Data Analysis:
  - Pitch Extraction
  - Formant Extraction
  - Intensity
  - Speech Rate
- Phoneme Extraction and manipulation
- Fast Fourier Transforms (FFT)
- Power Spectral Density (PSD)
- Linear Predictive Coefficients (LPC)
- Cepstral Analysis
- Spectrogram

Representations of the signal for speaker/speech recognition are mostly derived from the power spectrum. The phase structure of the signal has been ignored. Our ears are insensitive to phase effects. The recognizers have concentrated on properties of the signal that are attributable to the shape of the vocal tract rather than the excitation.

2.9.1 Pitch and formant

The pitch of a signal is also known as the fundamental frequency. An important parameter for feature extraction is the frequency of glottal excitation during voiced sounds [11]. This is 'pitch-synchronous processing'.

A formant is a resonant peak in the spectrum of voiced speech.

2.9.2 Speech rate and phoneme extraction

The rate at which people speak can also be used to distinguish them from each other. Individual phoneme characteristics of recorded speech may also be compared to find differences between speakers.

2.9.3 Fast fourier transform

The discrete Fourier transform, DFT, is the primary tool of digital signal processing. The Fast Fourier Transform (FFT), a method for computing the DFT in minimal execution time based on the radix-2 method was employed. Functions to calculate the FFT and it's inverse IFFT are given as equations 2.1 and 2.2 respectively [15].

$$X(k+1) = \sum_{n=0}^{N-1} x(n+1) W_N^{kn} \qquad ,k=0,1,\ldots,N \tag{2.1}$$

$$x(n+1) = \frac{1}{N} \sum_{k=0}^{N-1} X(k+1) W_N^{-kn} \qquad , n=0,1,\ldots,N \tag{2.2}$$

$$W_N = e^{-j(2\pi/N)} \tag{2.3}$$

The Fourier transforms of two training samples are shown in figure 2.6. Fourier transforms also have complex components, which have not been plotted.

In digital filter design and power spectrum estimation, the choice of window function can play an important role in determining the quality of the overall results. The main role of a window is to damp out the effects of the Gibbs phenomenon resulting from the truncation of an infinite series [32].

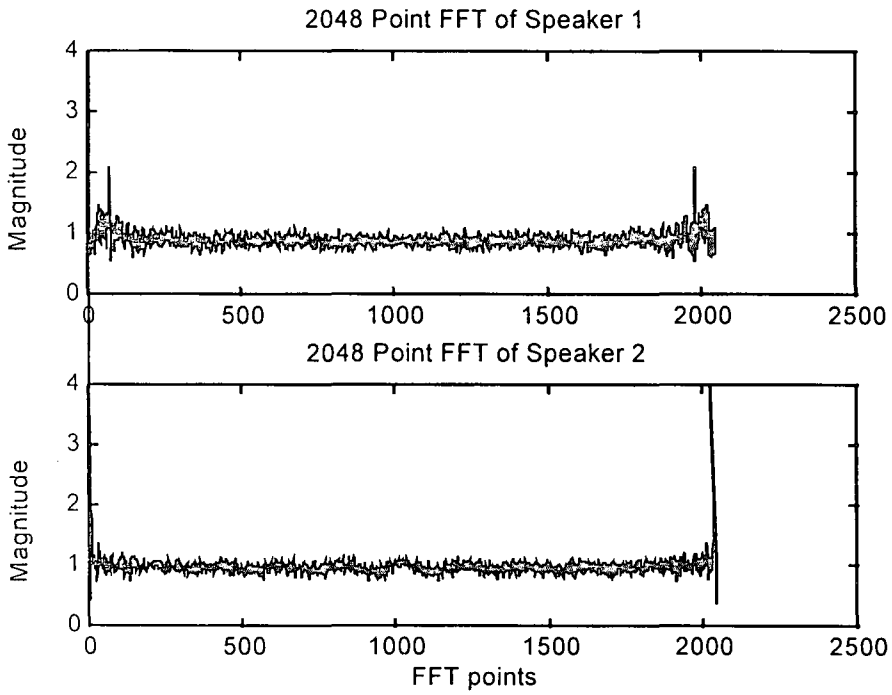


Figure 2.6 - Fourier transforms

2.9.4 Power spectral density (PSD)

Spectral Analysis seeks to describe the frequency content of a signal based on a finite set of data. The Power Spectral Density (PSD) is computed by taking the magnitude-squared result of the Fast Fourier Transform (FFT) of the speech signal [15]. These periodograms are then averaged and scaled. The PSD of a voice sample contains unique features attributed to an individual and these are

used in our studies. These PSD values are presented in a vector form to the pattern-matching network.

The power spectral density (PSD) is related mathematically to the correlation sequence by the discrete time Fourier transforms. The resulting unit of PSD is volts^2 / Hz.

$$P_{xx}(\omega) = \sum_{m=-\infty}^{\infty} R_{xx}(m) e^{-j\omega m} \tag{2.4}$$

The power spectrum is represented on a log scale. The shape of the log of the power spectrum remains unchanged when the gain applied to the speech signal is changed. Also, the log of zero is infinite resulting in problems representing very low components of the energy spectrum. The log function prevents excessive sensitivity to low-energy, noise-dominated parts of the spectrum.

2.9.5 Linear predictive coding

Linear prediction is a feature that is most commonly used in speech and speaker recognition. It is accomplished using past data. If future samples can be predicted from present and past samples then only the unpredicted component needs to be transmitted, and the redundancy will be reduced.

In 1949, Kolmogorov and Wiener devised a general polynomial that contained 94 terms. Only the linear terms were are adequate for most practical applications. Modern speech devices only use linear terms, hence the expression 'linear prediction', a special case of Kolmogorov's polynomial:

$$s(n) = \sum_{i=1}^{M} a_i s(n - i) \tag{2.5}$$

where *s(n)* is the predicted value of the nth sample based on the previous M samples. The a_i is the constant coefficients to be determined.

In linear prediction representation, the speech signal is modeled as the output of an all pole filter *H(z)* that is excited by a sequence of pulses separated by the pitch period for voiced sounds, or pseudo random noise for unvoiced sounds [10]. a_i are the coefficients of the filter. The filter models the combined modulation properties of the glottal source and the vocal tract. This technique is also called the maximum entropy method (MEM) of spectral analysis. The process generates a stable filter, but it might not model the process exactly. This is because the data is windowed, that is, it assumes signal samples beyond the length of x are zero.

Linear predictive coding is also known as *autoregressive modeling*. Although the parameters of an all-pole filter are fitted to the speech spectrum, the spectrum itself need not be computed explicitly. This is the most common short-term spectral measurement.

Distance measure is a method of normalization used with LPC: Assume we have a segment of speech that can be properly represented by the LPC coefficients a_k. Now let us distort this same segment of speech and determine the LPC coefficients b_k. The difference between the two sets of coefficients is used to identify speech or speaker [10]. This method is used to combat the effect of speaker variations from trial to trial.

PCs have been used in the system simulation and results obtained using this extracted feature parameter are presented in Chapter Four.

2.9.6 Cepstral analysis

Cepstrum analysis is a non-linear signal processing technique with a variety of applications such as speech and image processing. It is also widely used for echo detection and cancellation [15].

Real and complex cepstrums may be calculated. The complex cepstrum of a function is calculated by finding the natural logarithm of the Fourier transform of x, then the inverse Fourier transform of the resulting sequence.

The real cepstrum of x is calculated by finding the natural logarithm of the magnitude of the Fourier transform of x, then the inverse Fourier transform of the resulting sequence.

A spectral envelope reconstructed from Cepstral coefficients is much smoother than one reconstructed from LPC coefficients.

2.9.7 Spectrograms

This is the time-dependent Fourier transform of voice signals [17]. Instead of just the frequency or spectral representation of the signal we also have the time domain representation as well. Feature extraction plays a very important role in speaker identification. Human speech can be sensibly interpreted using frequency-time interpretations such as spectrograms.

2.9.8 Hybrid feature parameters

The use of hybrid feature parameters, instead of single individual parameters, has been proposed in an effort to optimize the performance of the speaker

recognition system since there is no individual voice characteristic that can be extracted from a speech signal and indicate without doubt the speaker's identity.

This work has attempted various combinations of hybrid feature vectors and Chapter Four shows that the hybrid vector comprising of the complex cepstrum derived from the LPC coefficients are the most successful for speaker recognition.

PSD, LPC, Cepstral analysis and hybrid feature vectors have been used in our simulations and the eventual implementation on the target system. These are discussed in detail in Chapter Four.

2.10 Dynamic Time Warping

Prior to the Hidden Markov Model word recognizer, speech recognition was based almost entirely on Dynamic Time Warping (DTW) [21]. Dynamic time warping systems are limited in that they are speaker dependent and can only work on discrete words or phrases (pseudoconnected word recognition). This speaker dependence implies that they may also be utilized for speaker recognition.

In DTW, a set of speech templates is maintained in memory for each word or phrase in the vocabulary of the system. The templates are based on LPC models. As new words or phonemes are acquired and placed in the speech queue, its features or characteristics are compared with the memory-resident templates one word/frame at a time. As the speech is compared, it is stretched or compressed in time to optimize the correlation with the templates in memory (hence dynamic time warping). The utterances are thus aligned to match the templates. The template with the highest correlation is the recognized word.

DTW is a good method for pattern matching but it has limitations. Word templates cannot model acoustic variability between speakers very well. Also, only templates for entire words may be used. It is impossible to utter parts of a word e.g. phonemes in isolation.

2.11 Hidden Markov Models

A Russian organic chemist, Vladimir Vasilyevich Markovnikov developed the Markov model (or Markovnikov rule), in 1870. Hidden Markov Models (HMMs) when applied to speech recognition bring about high performance systems. Among speech and research scientists it is widely believed that HMMs are one of the best and most successful modeling approaches for acoustic events in speech processing. There are many HMM-based successful speech recognition systems available.

The HMM can be applied to speaker recognition as an alternate to the template-based neural networks. The hidden Markov model approach to speaker verification is a statistical method of characterising the spectral properties of the frames of a speech pattern. It removes the need for speech templates by using a probabilistic acoustic model. The advantage of this is that data distributions may be learned automatically from a set of training utterances.

The five elements that define an HMM are:

i. **N**, the number of states in the model. We denote the individual states as {1, 2, ..., N} and the state at time t as q_t.
ii. **M**, the number of distinct observation symbols per state.
iii. **A** = $\{a_{ij}\}$, the state transition probability density function where $a_{ij} = P\,[q_{t+1} = j \mid q_t = i]$ (ie. given that the current state is i, what is the probability that the next state is j).
iv. **B** = $\{b_j(k)\}$, the observation symbol probability distribution. This defines the symbol distribution in state j.

v. $\pi = \{\pi_i\}$, the initial state distribution.

In compact notation, we can use $\lambda = \{A, B, \pi\}$ to indicate the complete parameter set of the model. This parameter set defines a probability measure for **O** (an observation sequence), i.e. $P[O|\lambda]$.

The first task in speech recognition is to build individual word models. This is done by adjusting the model parameters $\lambda = \{A, B, \pi\}$ to maximize $P[O|\lambda]$. Here we attempt to optimize the model parameters to best describe how a given observation sequence comes about. The observation sequence used to adjust the model parameters is called a *training sequence* because it is used to 'train' the HMM. For this solution, we choose $\lambda = \{A, B, \pi\}$ such that its likelihood, $P[O|\lambda]$, is locally maximized using an iterative procedure such as the Baum-Welch method.

Next we segment each of the word-training sequences into states, and then study the properties of the spectral vectors that lead to the observations occurring in each state. The observation sequence, $\mathbf{O} = (o_1, o_2, \ldots, o_T)$ and the model λ, is used to choose a corresponding state sequence $q = (q_1, q_2, \ldots, q_T)$ that is optimal. In practical situations, we use an optimality criterion to solve the problem as best as possible. Hence the goal here is to make refinements to the model (e.g. include more states or use of a different codebook size) to improve its capability of modeling the spoken word sequences. The solution to this problem involves using a Viterbi algorithm to find the single best sequence. The Viterbi algorithm is based on dynamic programming methods.

Finally, once the set of HMMs has been designed and optimized, recognition of an unknown word or phrase is performed. This is done by using the solution to the 'evaluation' problem to score each word model based upon the given test observation sequence. Given the observation sequence, $\mathbf{O} = (o_1, o_2, \ldots, o_T)$, and a model, $\lambda = \{A, B, \pi\}$, we efficiently compute $P[O|\lambda]$ (i.e. the probability of the

observation sequence, given the model). This problem is viewed as one of scoring how well a given model matches a given observation sequence. For example, if we are trying to determine from which speaker a phrase was uttered, the solution to this problem allows us to choose the model that best matches that speaker.

Although HMMs provide a powerful mechanism for simultaneously modeling temporal and spectral variations of an acoustic signal, there are some limitations [22]. These limitations arise from the assumptions that are made when applying HMMs to speech signals. The most dubious one is the independence assumption. This assumes that successive feature vectors matched to a model state are independently and identically distributed. This implies that the state output probability calculation makes no allowance for the influence of preceding vectors. The segmental and inter-frame dependence of speech needs to be overcome.

HMMs are available as application-specific DSP chips e.g. the TMS320C53 HMM speech recognition system from Texas Instruments. The system consists of 3 processes running together: a feature extractor, a sentence hypothesizer and a word hypothesizer. The feature extractor reduces the continuous speech to a series of states whose features are reduced to a finite feature set called the Generalized Feature Set (GSF). The HMM processes then compute sentence and word recognition on a state-by-state basis. The HMM processes can be described in terms of mathematical probabilities of the likelihood that one state following another.

HMMs do posses certain limitations, which are discussed in greater detail in section 3.14.

Gaussian Mixture Models (GMMs) is another current popular and successful statistical modeling technique applied to the application of speaker recognition.

DTW, GMMs and HMMs have not been included in the hardware implementation of this speaker recognition system.

2.12 Visual Speaker Recognition

An alternate method for person identification has been proposed in [9]. This is based on the visual spatio-temporal analysis of the talking face. A lip model is used to locate and track the lips of the talking person and to extract speaker dependent information from the motion sequence. This method is related to the common acoustic recognition system. The difference is that the features are extracted from the visual rather than the acoustic signal.

The benefit of using visual inputs is that they are often complementary to the acoustic signal. Phonemes that are difficult to distinguish acoustically are easier to distinguish visually by comparing facial movements.

This alternative technique is attempting to maximize the success of the speaker recognition system using the acoustic and visual techniques together. This is further proof of the lack of a single characteristic that can be extracted and used to indicate the speaker's identity.

2.13 Summary

The need for machines to interact with humans has increased dramatically with the rapid development of multi-media based systems. All the facets of spoken language processing can be applied to a fully interactive system that attempts human perception capability. Specifically, our application of speaker recognition is particularly useful for remote authentication. The development and

advancement of telecommunications and acoustic recording technology has enabled many applications to be "run" remotely e.g. electronic banking.

Speaker recognition has been attempted using various speech processing techniques. Of these, linear prediction, cepstral analysis, spectral and spectrogram analysis are the most popular and efficient with relative good recognition success rates.

The most significant factor affecting automatic speaker recognition performance is variation in the signal characteristics from moment to moment (inter-session variability and variability over time). Speakers cannot repeat an utterance precisely the same way at different moments in time. It is well known that samples of the same utterance recorded in one short period are much more highly correlated than samples recorded in different periods of time. Duration of a period can vary from a day to weeks. A long-term period (months to years) can cause major changes in voices. For example, especially with males, in the process of growing up, their voices' change indicating adulthood. Normalization techniques are used to counter this.

Although speaker recognition has made great strides towards the implementation of practical systems, computational advanced machines cannot compete with the perception capability of humans as yet.

CHAPTER 3

THEORY OF NEURAL NETWORKS

3.1 Introduction

This chapter describes typical artificial neural networks. Further, Learning Vector Quantization (LVQ) pattern recognition/classification networks and committee neural networks, which were applied to this speaker verification system, are described in this thesis.

There are numerous definitions of artificial neural networks:

> "... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes" [27].

> "A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired by the network through a learning process.

2. Interneuron connection strengths known as synaptic weights are used to store the knowledge" [28].

"A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock" [29].

"Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge"[30]

A pattern recogniser generally consists of a feature representation of each pattern class and a measure of membership to each class. Obviously both modules should be jointly designed with the single objective of minimising recognition errors for the optimality of the entire recognition process [22].

Traditional computer logic based systems require comprehensive programming in order to perform given tasks, Artificial Neural Networks (ANN) observe data and infers what needs to be done.

This is particularly useful where the comprehensive models that are required for conventional computing methods are either too large or complex to accurately represent systems/signals. The speaker recognition application is a good example of this. The responsibility of identifying the differences between the signal representations of different speakers is now that of the neural network.

Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. In the figure below, the network is adjusted,

based on comparison of the target and output, until the network output matches the target. Typically, many such input/target pairs are used. This called a *supervised* network [25].

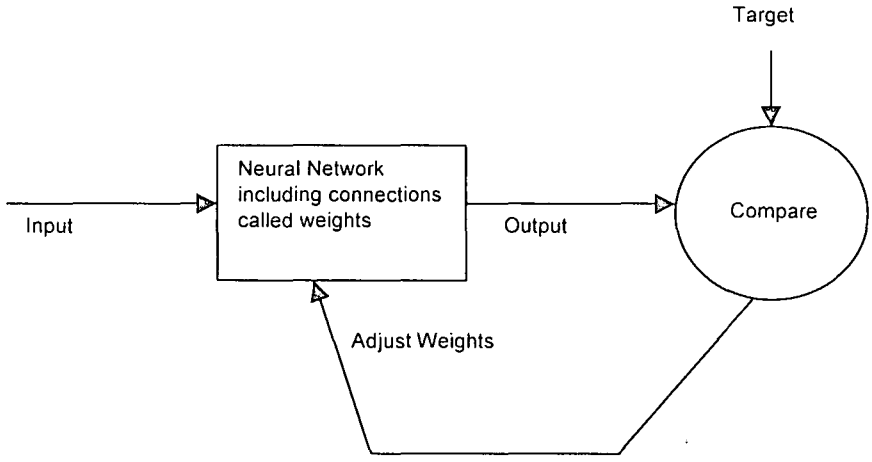


Figure 3.1 - Neural network adjustment

The functions of pattern matching and the memory template of the speaker recognition system, as shown in figure 3.2 below, will be accomplished using neural networks. A single neural network represents the pattern classification and memory template blocks.

Neural networks are composed of many simple elements operating in parallel. These elements are inspired by biological nervous systems. Artificial neural networks are intelligent systems based on the biological structure of the human brain.

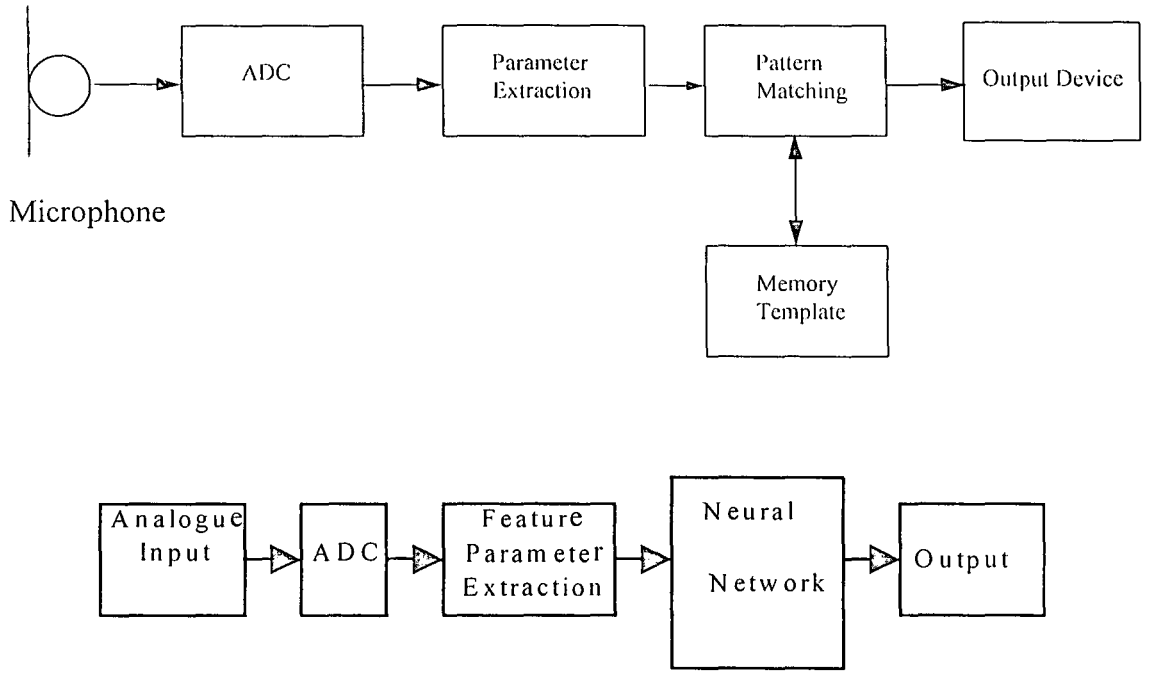


Figure 3.2 - Speaker Recognition System using a Neural Network

Kohonen's work on pattern classfication with the Self Organising Map (SOM) and Learning Vector Quantization (LVQ) neural networks have been applied to the pattern recognition component of our system [24].

Biological neural networks are much more complicated than the mathematical models we use for ANNs. Practical ANNs are extremely small in comparison with the human brain but their basic structure remains highly connected and distributed in nature which affords them a high degree of noise immunity and fault tolerance. ANN learning algorithms are also relatively insensitive to noise since they are aimed at identifying general trends and relationships in data and noisy training data actually helps to accomplish this in some cases.

3.2 Neural Network Applications

Artificial neural networks have evolved from their biologically inspired roots to a well-established means to solve a broad spectrum of problems. ANNs have found widespread acceptance for being robust systems for information processing in noisy environments.

Neural networks have been applied diversely, from stock market prediction to dynamic systems such as the induction machine. In about 1984 ANNs were implemented in the adaptive channel equaliser. This single neuron network is an outstanding commercial success. ANNs have since been applied to numerous other fields [23]:

- Aerospace

  High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors

- Automotive

  Automobile automatic guidance systems, warranty activity analyzers

- Banking

  Check and other document readers, credit application evaluators

- Defense

  Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification

- Electronics

  Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling

- Financial

  Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction

- Manufacturing

  Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process systems

- Medical

  Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement

- Robotics

  Trajectory control, forklift robot, manipulator controllers, vision systems

- Speech

  Speech recognition, **speaker recognition**, speech compression, vowel classification, text to speech synthesis

- Securities

  Market analysis, automatic bond rating, stock trading advisory systems

- Telecommunications

  Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems

- Transportation

  Truck brake diagnosis systems, vehicle scheduling, routing systems

The large number of neural network applications must not obscure the fact that there are some major unsolved problems concerning neural networks. There are still no satisfactorily constructive ways to determine the optimal structure (elements as well as organization) or the learning and evaluation dynamics .

3.3 The Neuron Model

3.3.1 Single Input Neuron

The inputs to a neuron, shown in figure 3.3, include its bias (b) and the sum of it's weighted inputs (p) using inner product. Biasing is optional and not present in all neurons. The input to the neuron is then augmented by the transfer function F which produces a scalar output a. The bias is much like a weight except that it has a constrained input 1. w and b are adjustable scalar parameters.

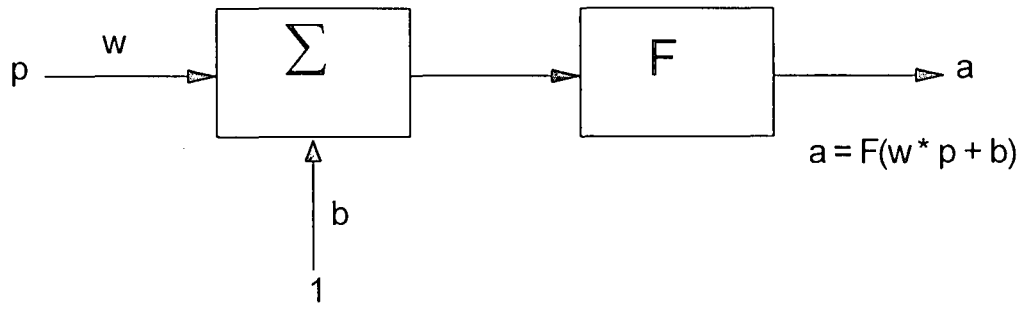


Figure 3.3 - Single input neuron

3.3.2 Transfer functions

Hard limit transfer function

The output of the neuron is limited to either 0 or 1 depending on whether or not input, p, exceeds -b/w as illustrated in figure 3.4.

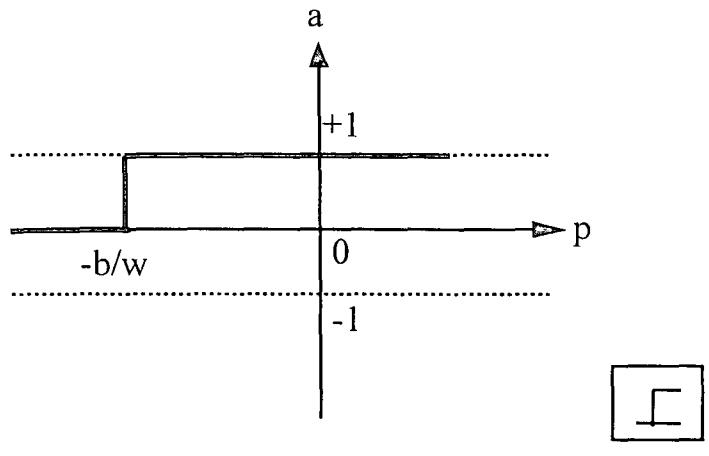


Figure 3.4 - Hard limit transfer function with bias

The Linear Transfer Function

The input output relationship for a single input neuron with a linear transfer function is shown in figure 3.5:

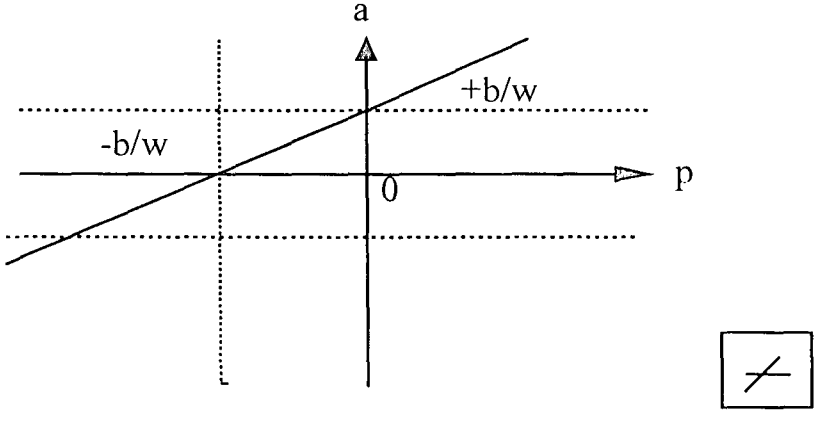


Figure 3.5 - Linear transfer function

The sigmoid transfer function

The sigmoid transfer function, shown in figure 3.6, takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 and 1. This transfer function is commonly used in backpropagation networks and is differentiable.

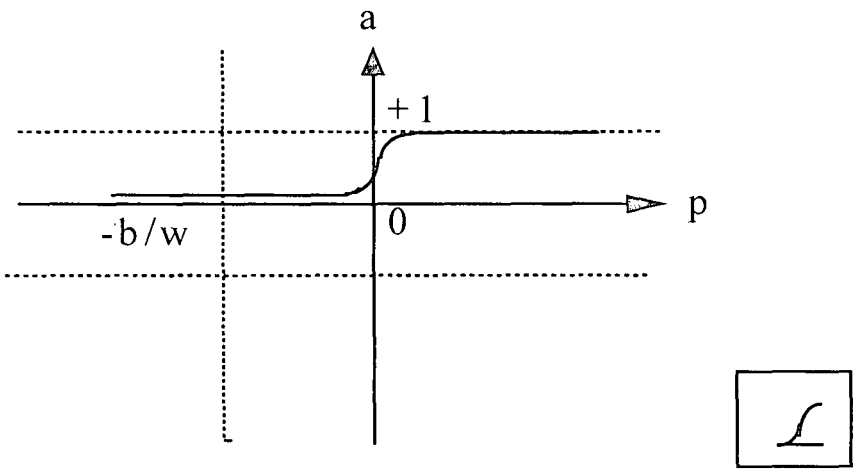


Figure 3.6 - Sigmoidal transfer function

3.3.3 Multiple input neurons

A single neuron with R inputs is shown in figure 3.7 below. Individual inputs p(1), p(2),...p(R) are weighted by elements w(1,1), w(1,2),.....w(1,R), and the weighted values are inputs to the summing junction. Their sum is w*p, the dot product of the row vector w and the column vector p. The neuron has a bias value b, which is summed with the weighted inputs to form the net input n. This sum n is the argument of the transfer function F.

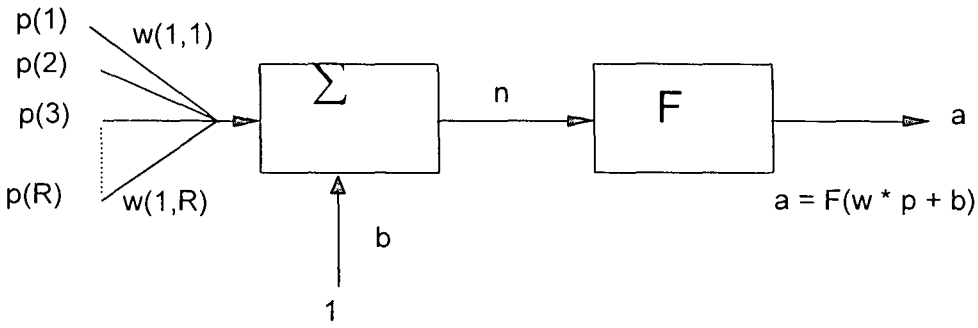


Figure 3.7 - Multiple input neuron

The application of speaker recognition requires an input layer capable of multiple input per neuron. Chapter Four illustrates simulations using input feature parameter vectors comprising of up to 100 elements per input vector. The abbreviated notation of the multiple input neuron is shown in figure 3.8 below.

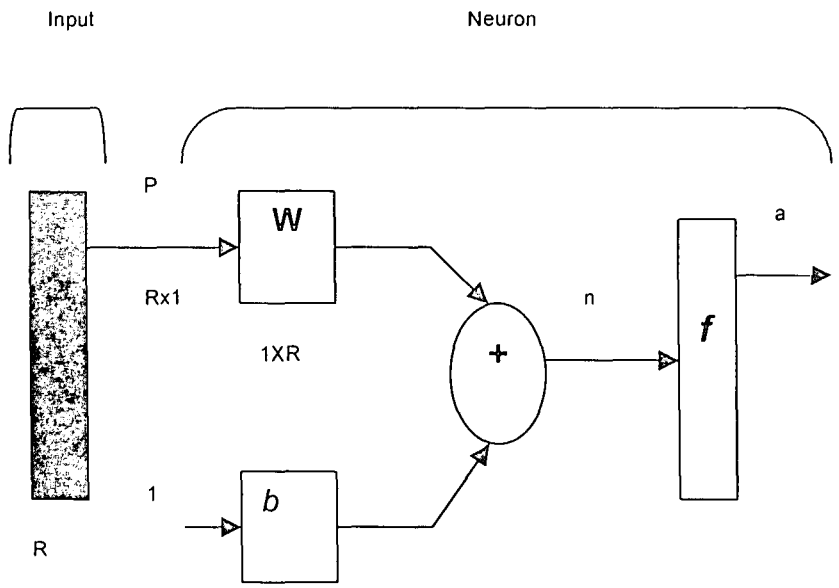


Figure 3.8 - Multiple input neuron (abbreviated notation)

3.4 Network Architectures

A layer of a network is defined as follows: A layer includes the combination of the weights, the multiplication and summing operation, the bias b, and the transfer function F. The array of inputs, p will not be considered a layer. Two or more neurons may be combined in a layer, and a particular network might contain one or more such layers.

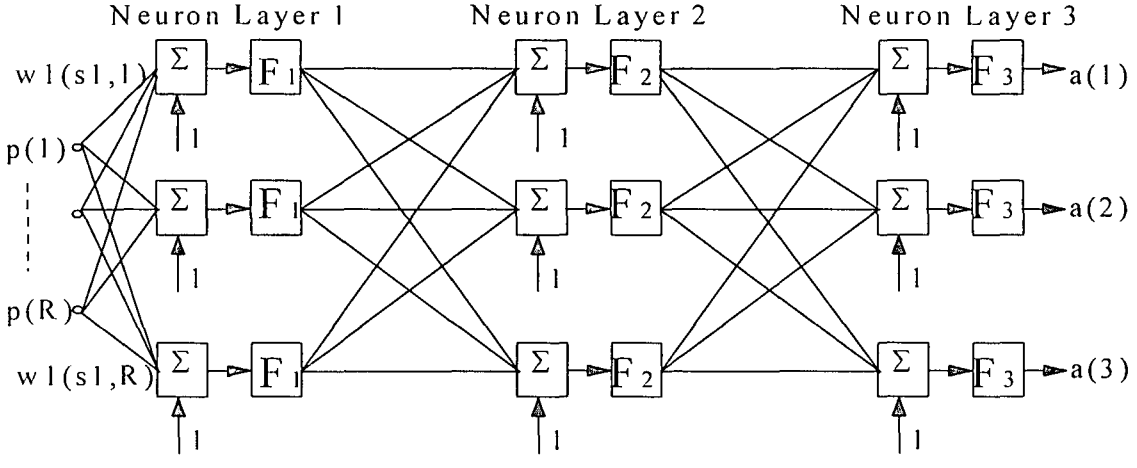


Figure 3.9 - Multiple layers of neurons

The network shown above has R inputs, S1 neurons in the first layer. It is common for different layers to have different numbers of neurons. The outputs of each intermediate layer are the inputs to the next layer. A layer to produce the output is called the output layer. The other layers are called hidden layers. Multiple layer networks are quite powerful.

The architecture of a network consists of a description of how many layers a network has, the number of neurons in each layer, each layer's transfer function and how the layers are connected together and to the inputs. The best type of architecture to use depends on the type of problem.

Aside from the number of neurons in the output layer, the number of neurons in each layer is up to the designer. Linear networks are the exception where the more neurons in the hidden layer the more powerful the network.

Networks with biases can represent relationships between inputs and outputs more easily than those without biases.

3.5 Perceptrons

A perceptron neuron uses the hardlimit transfer function. The output of a perceptron neuron is limited to either 0 or 1 due to the hard limit transfer function. Perceptrons can only classify linearly separable sets of input vectors. Data is seldomly linearly separable in real world problems. Another problem is that an outlier, an input vector much larger or smaller than the rest of the input vectors slows the convergence process.

3.6 Linear Networks

Linear networks differ from perceptrons in that their neurons have a linear transfer function. This allows outputs to take on any value as opposed to just a 0 or a 1. It also enables the Least Mean Square (LMS) learning rule, to adjust weights and biases according to the magnitude of errors and not just their presence.

Linear networks when presented with a set of given input vectors must output their corresponding target vectors. For each input vector we calculate the output vector. The difference between the output vectors and the target vectors are the

errors. We would like to find values for the biases and weights such that the sum of the errors squared is minimized or below a specified value. Linear systems have a single error minimum.

Once a network is trained successfully and an input vector not in the training set is presented to the network, it will tend to produce an output vector similar to output vectors associated with similar input vectors. This behaviour is called *generalization* and is what makes neural networks so powerful.

3.6.1 Learning rate

A high learning rate leads to unstable learning which means that the network may have trouble in converging to the error minimum. If the learning rate is too small then the network will take very long to converge to the error minimum.

Multiple layers in a linear network do not result in more powerful networks, however linear networks can solve only linear problems. Unless relationship between input and targets are linear backpropagation may be a good alternative [25].

3.7 Supervised and Unsupervised Networks

In supervised learning, the correct results (target values, desired outputs) are known and are given to the ANN during training so that the NN can adjust its weights to try match its outputs to the target values. The data presented to the network in this stage is called the training data. After training, the ANN is tested by giving it only input values, not target values, and seeing how close it comes to

outputting the correct target values. Supervised neural networks are required to learn an input-output mapping from existing data.

In unsupervised learning, the ANN is not provided with the correct results during training. Unsupervised ANNs usually perform some kind of data compression, such as dimensionality reduction or clustering. Te distinction between supervised and unsupervised methods is not always clear-cut. An unsupervised method can learn a summary of a probability distribution, then that summarized distribution can be used to make predictions. Furthermore, supervised methods come in two subvarieties: auto-associative and hetero-associative. In auto-associative learning, the target values are the same as the inputs, whereas in hetero-associative learning, the targets are generally different from the inputs. Many unsupervised methods are equivalent to auto-associative supervised

3.8 Backpropagation

Backpropagation was created by generalizing the LMS rule multiple layer networks and non-linear differentiable transfer functions. Input vectors and their corresponding output vectors are used to train a network until it can approximate a function, associate input vectors with a specific output vector, or classify input vectors in a way defined by the user. Networks with biases, a sigmoid layer and a linear output layer are capable of approximating any function with a finite number of discontinuities [25]. The generalisation property of backpropagation networks makes it possible to train a network on a representative set of input/target pairs and get good results without training the network on all possible input/output pairs.

3.8.1 Backpropagation transfer functions

Backpropagation networks often use the log-sigmoid transfer function. The function generates outputs between 0 and 1 as the neurons net input goes from negative to positive infinity. Tan-sigmoid and occasionally linear transfer functions may also be used. If the last layer of a backpropagation network has sigmoid neurons then the outputs of the network are limited to a small range. If linear output neurons are used then the outputs can take on any value. Therefore backpropagation networks often have one or more layers of sigmoid neurons followed by a layer of linear neurons. The non-linear transfer function allows the network to learn both linear and non-linear relationships between input and output vectors. The linear output layer lets the network produce values outside the range -1 to +1.

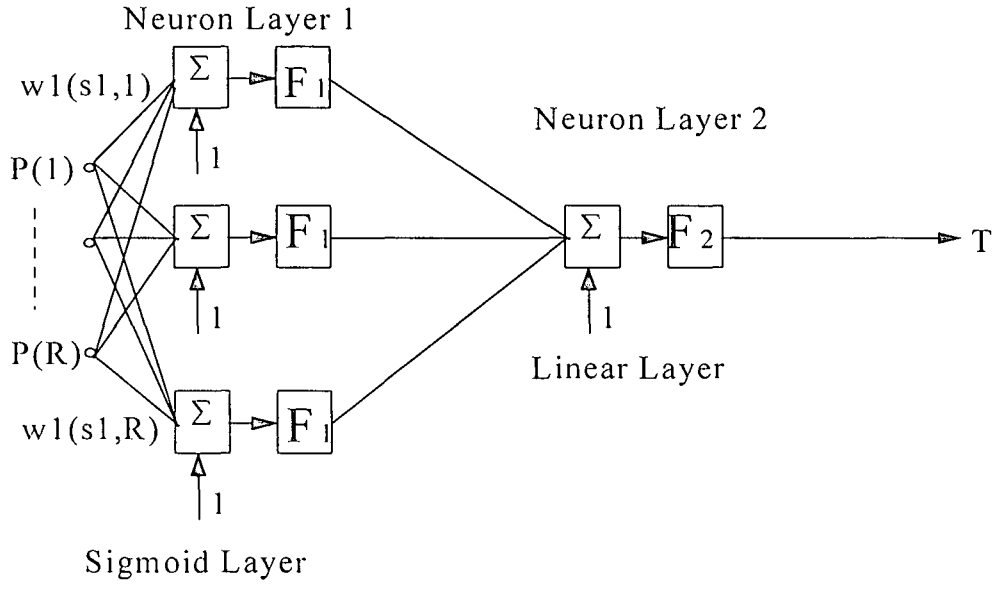


Figure 3.10 - Backpropagation neural network

3.8.2 Momentum

Backpropagation training may lead to local rather than global error minimum. Non-linear transfer functions in multi-layered networks introduce many local minima in the error surface. Therefore a technique called momentum is used. Momentum decreases sensitivity to small details in the error surface. This prevents the network from getting stuck in shallow minima, which would prevent the network from finding a lower error solution.

3.8.3 Gradient descent

The transfer functions mentioned above are differentiable and are also monotonically increasing functions - the output of each function increases as the input increases.

Back propagation is based on gradient descent where parameters such as weights and biases are moved in the opposite direction to the error gradient (Dem94). The basic procedure of gradient descent is simple. Starting from an arbitrary chosen weight w, the gradient $\nabla \mathbf{E}(\mathbf{w})$ of the current error function is computed. The next value of w is obtained by moving in the direction of the negative gradient along the multidimensional error surface (Zur92). The direction of negative gradient is the one of steepest descent. Algorithm summarized as follows:

$$\mathbf{w}^{k+1} = \mathbf{w}^{k} - l_r \nabla \mathbf{E}(\mathbf{w}^{k}) \qquad (3.1)$$

where l_r is the learning rate.

3.8.4 Underfitting

Training is sensitive to the number of neurons in the hidden layers. If too few neurons are present in this layer then the network will not be able to solve the problem. Thus if training a network for a long time still results in large errors, the problem is most likely a lack of hidden layer neurons.

3.8.5 Training

Figure 3.11 shows how the network performance improves with training until the performance of the network matches the desired output. Function approximation has been achieved.

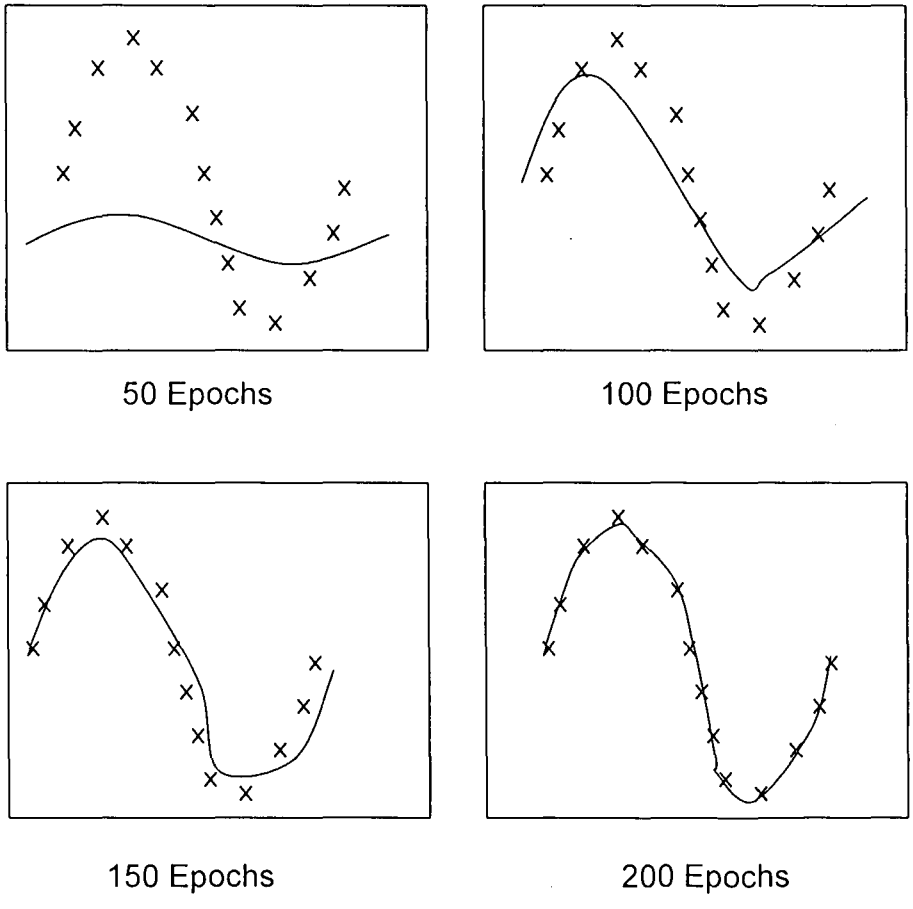


Figure 3.11 - Training of a backpropagation network

3.8.6 Over-fitting

The result of too many neurons is that the network appears to be trained quite easily but test vectors (not included in training set) do not generate reasonable results.
Backpropagation is used in perhaps 80 to 90% of practical applications [25].

3.9 Associative Learning Rules

The Hebb rule is used to create associations between individual elements in a neural network. Hebb rule with decay allows associations to be learned while keeping the weights from getting unreasonably large.

The instar rule creates associations between a layer's input vectors and the layer's individual neurons. Each neuron in the layer learns to respond to a different input vector.

The outstar rule creates associations between a layer's output vectors and a linear layer's input elements.

These three rules are local learning rules and may be used to train layers embedded in a network. Weight changes do not depend on knowledge of the rest of the network architecture.

Associative learning rules are used as building blocks of a network.

3.10 Self-Organizing Networks

Self-Organising networks can learn to detect regularities and correlations in their input and adapt their future responses to that input accordingly.

3.10.1 Competitive learning

The neurons of competitive networks learn to recognize groups of similar input vectors. Competitive networks may be trained with the instar learning rule. Each neuron competes to respond to an input vector p. The neuron whose weight vector is closest to p wins and outputs a 1. All other neurons output a 0. Only the neuron that wins has its weight updated as result it is most likely to win the next time.

One limitation is that some neurons may never be allocated to an input. Therefore dead neurons. Therefore biases are used to give neurons that win rarely an advantage over those that win regularly. Forces each neuron to classify roughly the same percentage of input vectors.

3.10.2 Self organizing maps (SOMs)

They are competitive networks with a variation. SOMs learn to recognize input vectors in such a way that neurons physically close together in a neuron layer learn to respond to similar input vectors. This results in the clustering of neurons representing a similar type of input. Each cluster of neurons represents a specific input class. This is useful for input classification. This is an unsupervised network

developed by Teuvo Kohonen who is one of the most famous and prolific researchers in neurocomputing.
The SOM differs from competitive learning neurons in which neurons get their weights updated. Instead of updating only the winner, feature maps update the weights of the winner as well as its neighbours.

Self-organising networks are good for categorisation of input vectors.

3.10.3 Learning vector quantisation (LVQ)

LVQ is a method for training competitive networks in a supervised manner. A competitive layer automatically classifies input vectors. Classes that the competitive layer finds depends only on the distance between the input vectors. If two input vectors are similar then they put into the same class. LVQ networks learn to classify vectors into target classes specified by the user.

LVQ neural networks have been used to perform the pattern recognition task of our speaker recognition system. LVQs are closely related to Self-Organizing Maps (SOM). It is an algorithm that effectively maps similar patterns (pattern vectors close to each other in the input signal space) onto locations in the output space [24]. Learning Vector Quantization is a supervised version of SOM particularly suitable for statistical pattern recognition.

LVQ training requires a training set of examples of the proper network behaviour. If the input pattern is classified correctly, then move the winning weight toward the input vector according to the **Kohonen** rule.

$$ {}_{i*}W^1(q) = {}_{i*}W^1(q-1) + \alpha(p(q) - {}_{i*}W^1(q-1)) \qquad (3.2) $$

If the input pattern is classified incorrectly, then move the winning weight away from the input vector according to the rule:

$$_i.W^1(q) = {}_i.W^1(q-1) - \alpha(p(q) - {}_i.W^1(q-1)) \quad (3.3)$$

This is known as LVQ1 training. Other variations exist where the neighbours of the winning weight are adjusted as well as the winning weight (LVQ2 and LVQ3). The weight vectors of the competitive layer are calculated using the midpoint theorem. The training input vectors must contain expected minimum and maximum values in their range. The net input to the competitive layer is the negative distance between the prototype vectors, W^1 and the input:

$$n_i^1 = -\| {}_iW^1 - p \| \quad (3.4)$$

Figure 3.12 shows an LVQ network with a competitive layer being followed by a linear layer. The classes learned by the competitive layer are termed subclasses and the classes of the linear layer are termed target classes.

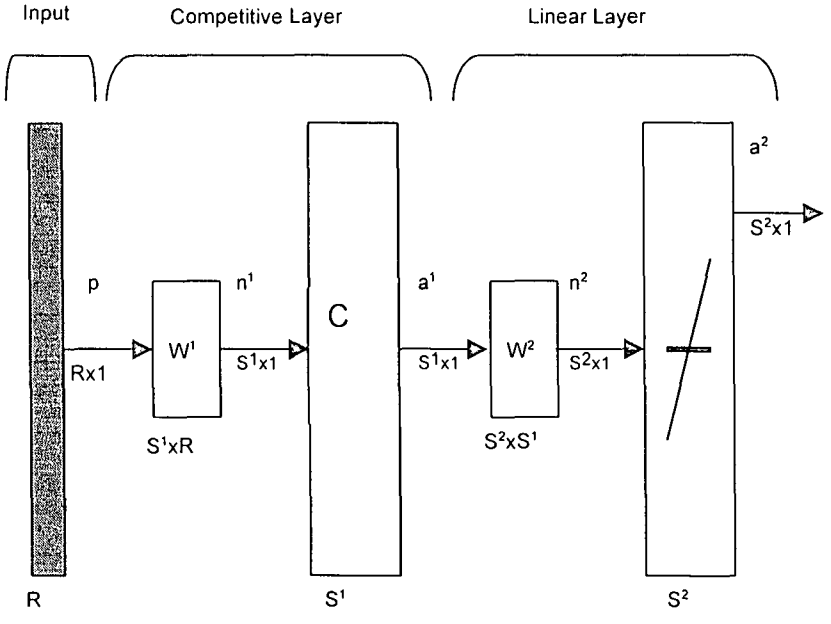


Figure 3.12 - Architecture of an LVQ network

The output of the linear layer is given by the relationship:

$$a^2 = W^2 a^1 \tag{3.5}$$

S^1 is the number of neurons in the competitive layer. This parameter is user defined. The magnitude of W^1 is dependent on S^1 according to the relationship:

$$W^1 = S^1 \times R \tag{3.6}$$

For the LVQ network, the winning neuron in the first layer indicates the **subclass** which the input vector belongs to. There may be several different neurons (subclasses) which make up each class.

The second layer of the LVQ network combines subclasses into a single class. The columns of $\mathbf{W}^2$ represent subclasses, and the rows represent classes. $\mathbf{W}^2$ has a single 1 in each column, with the other elements set to zero. The row in which the 1 occurs indicates which class the appropriate subclass belongs to.

E.g.

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.7}$$

Subclasses 1, 3 and 4 belong to class 1.
Subclass 2 belongs to class 2.
Subclasses 5 and 6 belong to class 3.

A single-layer competitive network can create convex classification regions. The second layer of the LVQ network can combine the convex regions to create more complex categories [23].

LVQ2

If the winning neuron in the hidden layer incorrectly classifies the current input, we move its weight vector away from the input vector, as before. However, we also adjust the weights of the closest neuron to the input vector that does classify it properly. The weights for this second neuron should be moved toward the input vector.

When the network correctly classifies an input vector, the weights of only one neuron are moved toward the input vector. However, if the input vector is incorrectly classified, the weights of two neurons are updated, one weight vector is moved away from the input vector, and the other one is moved toward the input vector. The resulting algorithm is called **LVQ2** [24].

3.11 Stability of Self-Organizing Networks

If the input vectors don't fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.

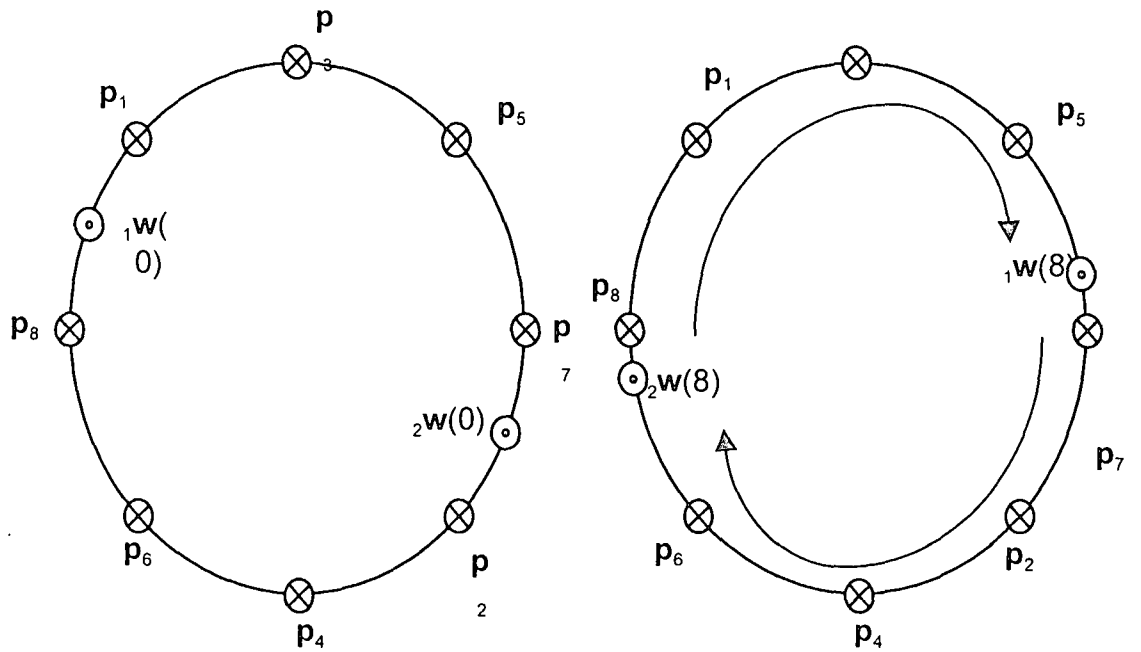


Figure 3.13 - Stability of self-organizing networks

If the input vectors don't fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.

3.12 Committee of Neural Networks

To improve neural network decision making, we have proposed the fusing of multiple networks, called the committee network approach. This thesis illustrates

improved predictions by using committee networks trained as a product of cross-validation. Speaker verification is used as the test application.

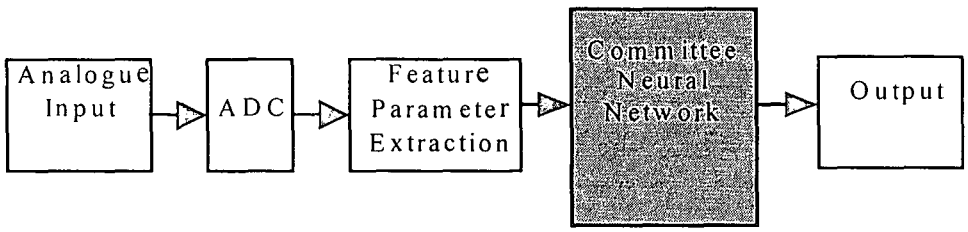


Figure 3.14 - Committee of neural networks

This thesis illustrates the introduction of a Committee of Neural Networks instead of a single recognition network. The final committee decision will be based on majority voting of the member networks. Using several individual networks rather than a single neural network optimizes the output of the committee network. Each member of this network can be a complete individual neural network. The LVQ network was the neural network architecture of choice since it produced optimum recognition results when used alone. Refer to Chapter Four.

3.13 Recurrent Networks

Elman and *Hopfield* networks are two examples of recurrent networks:

Elman networks are two layer backpropagation networks with the addition of a feedback connection from the output of the hidden layer to it's input. This feedback path allows the network to recognize and generate temporal and spatial patterns. Useful in signal processing and prediction where time plays a dominant role.

Hopfield networks are used as error correction or vector categorization networks. Seldom used in practice since they may have spurious stable points that lead to incorrect answers. These types of neural networks have therefore not been considered in this study.

3.14 Hybrid HMM-NNET Systems

Although the HMM is very good at modeling the temporal nature of processes such as speech, it has a limited capacity in recognizing complex structures involving more than the first order dependencies of the observed data sequences [22]. This is due to the following HMM limitations:

First order Markov assumption: This implies that the time dependencies between successive observations are first order linear correlations. This is not true for speech signals since coarticulation can range to a few frames.

State conditional observation independence: This implies that there are no correlations between adjacent observations other than first order dependencies. The human vocal system cannot change instantly and thus is limited by the HMM model.

Match Density Models: The discrete probability match suffers from quantisation errors while the continuous Gaussian mixture ssuffers from mismatch since only small mixtures can be estimated reliably.

Discrimination: The Maximum Likelihood (ML) algorithm used is an unsupervised algorithm that only uses the data belonging to a certain class to train the model representing that class.

Artificial neural networks are the opposite, they cannot model dynamic phenomena very well but they are good at static classification and regression tasks. Combining the advantages of HMMs and ANNs can lead to a more powerful model with more classification abilities. A hybrid method called the Hidden Neural Network (HNN) is proposed in [22].

3.15 Summary

Most problems can be solved using a variety of methods, although there is always one optimal method. Conventional programs lend themselves to problems that are clearly understood and can be described completely. It is important to understand that there are no methods for training ANNs that can magically create information that is not contained in the training data. Therefore, the more training data, the better the performance of the ANN.

Traditional Computer logic based systems require comprehensive programming in order to perform given tasks, ANNS observe data and infers what needs to be done.

LVQ networks in particular are suited to our application of speaker recognition since we may specify the desired target class of each input to the system.

Committee neural networks may be used to increase the performance accuracy of the system by taking the majority vote of a multi-member network rather than a single individual network.

ANNs can be applied to areas where the information is incomplete and not fully understood. They are particularly useful where comprehensive models required for conventional computing methods are either too large or complex to accurately represent systems or signals.Another advantage of ANNs is their high degree of noise immunity and fault tolerance.

The advantages of neural networks stated above makes it an ideal solution for speaker recognition. The extracted featured parameters aren't complete (representations of the original signal) and the low SNR is often a factor that reduces the reliability of the system.

CHAPTER 4

MATLAB IMPLEMENTATION OF THE SPEAKER RECOGNITION SYSTEM

4.1 Introduction

Prior to implementation on the DSP target platform, the speaker recognition system was implemented on a personal computer (PC) using a collection of functions built in the MATLAB based computing environment. This chapter focuses on the methodology and the results obtained with the MATLAB implementation of the speaker recognition system. The results obtained from this MATLAB implementation justified the methods employed in the hardware implementation of the speaker recognition system.

Speech samples were recorded using a standard low quality (to simulate noisy conditions) microphone during the MATLAB implementation of the speaker recognition system. The analog signal was then digitized and interfaced to the PC using a standard 16-bit sound card. Sound recording software was used to record and save the samples in the required 'au' file format. Signal processing was then carried out on the signal using MATLAB signal processing software to extract characteristic features of each speaker. Feature vectors, comprising of single and hybrid characteristics, were then presented to a neural network, assembled in MATLAB, for system training and testing. A committee of neural networks was also used to generate improved results for the speaker recognition system.

4.2 Description of the Used Speaker Databases

4.2.1 Quick database

Software and hardware implementation results were obtained using a newly constructed speaker database named Quick (because of the utterance used by the speakers). The database used consisted of a total of 20 speakers. Table 4.1 indicates that the database consists of 6 female and 14 male subjects.

Table 4.1 - Speaker Key

Speaker	Name	Sex	Age	Abbreviation/Key
1	Viresh	M	29	vir
2	Suzan	F	26	suz
3	Satish	M	21	sat
4	Kumar	M	29	kum
5	Kasturi	F	31	kas
6	Tammy	F	24	tam
7	Kuben	M	25	kub
8	Dolly	F	49	mum
9	Harry	M	53	dad
10	Maggie	F	21	mag
11	Aubrey	M	20	aub
12	Lushern	M	21	lus
13	Valerie	F	40	val
14	Bevlin	M	26	bev
15	Seelan	M	26	see
16	Shane	M	38	sha
17	Sagren	M	24	sag
18	Anban	M	24	anb
19	Vinesh	M	27	vin
20	Q Lee	M	23	qui

Each speaker produced 10 samples that were used either as training or test utterances during the MATLAB implementation of the recognition system.

There are a number of speech and speaker research databases already in existence but it would not have been possible to test the hardware system implementation using samples from these since it is required to train and test the system with real-time (not recorded) inputs.

4.2.2 YOHO speaker database

The large corpora on speech and speaker recognition has led to many research projects e.g. National Institute of Standards and Technology (NIST) project and the CAVE project (CAVE is an acronym for Caller Verification).

These corpora and other research projects have led to many speech databases, some of which are available to the public:

a) GANDALF
b) SESP
c) POLYCOST
d) TIMIT
e) YOHO
f) ARHUMADA
g) SPIDRE

GANDALF, SESP and POLYCOST are three large telephone quality corpora. The speaker-oriented telephone-line POLYCOST database is a multi-lingual database with non-native English and mother-tongue speech by subjects from 14 countries. TIMIT is a benchmark database of 630 speakers and ARHUMADA is a large speech database in Spanish.

YOHO is a standard database for testing voice verification systems and is available from the Linguistic Data Consortium (LDC). It consists of office

environment speech and has been used to test the performance of the software implementation on an expanded dataset. There are 138 speakers (106 males and 32 females); for each speaker, there are 4 enrollment sessions of 24 utterances each and 10 verification sessions of 4 utterances each.

The syntax used in the YOHO database is "combination lock" phrases. For example," twenty-six, eighty-one, fifty-seven." All speaker utterances have been sampled at 8 kHz. The data has been collected over a three-month period. This database has only been used to test the software implementation of an expanded database of forty-nine speakers. All other software and hardware implementations and results were obtained using the Quick database.

4.3 Verification Utterance

The Quick database subjects were all required to use the following verification utterance:

"The quick brown fox jumps over the lazy dog"

This phrase, which contains all the letters of the alphabet, was chosen in order to achieve maximum variation between speakers. Another utterance: " Alas men men alas alas" was also tried in order to test theory of different phonemes being more appropriate than others for speaker identification as suggested in section 2.3. No noticeable improvement in the performance of the system was achieved with the new utterance when used on a five-speaker backpropagation system [1].

4.4 Recognition System Specification

A block diagram of the MATLAB based system is shown in figure 4.1. This is actually an implementation of a PC-based speaker recognition system. This system was implemented on a PC with the following specifications: Pentium processor (233MHz), 16-bit sound card and 40 Mbytes of RAM memory.

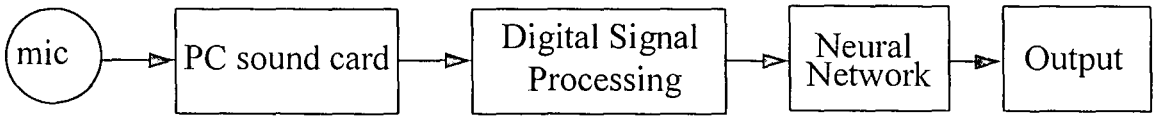


Figure 4.1 – MATLAB implementation of the speaker recognition system

Each block of figure 4.1, shown above, performs the following functions:

mic – Conversion of sound waves to analog electrical signals
PC sound card – ADC conversion of the electrical signal to digital form
MATLAB's Digital Signal Processing Toolbox – Feature extraction
MATLAB's Neural Network Toolbox – Pattern templates and pattern matching

4.5 Sample Acquisition (ADC)

Sound, like most signals in nature, are analog in form. This means that they vary continuously with time. It is impossible, for a computer, to process an infinite amount of data, which would be required if continuous signals were to be represented in the machine. Therefore samples of data at various instances in time are necessary. A microphone was used as a transducer, converting the sound waves produced by the speaker into an electrical analog signal. The analog signal is then digitized using standard 16-bit PC sound card.

The advantages of using digital signals are numerous:

- Guaranteed Accuracy
- Perfect Reproducibility
- Digital Components may be used - eliminates temperature/age effects.
- Makes use of advantages of semi-conductor technology.
- Greater flexibility and performance

Figure 4.2 shows how the PC sound card was used to perform the analog to digital conversion.

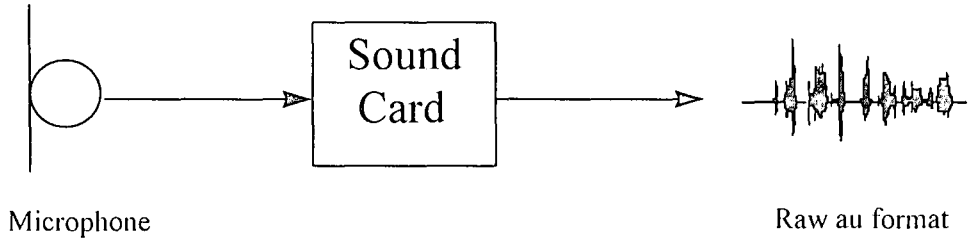


Figure 4.2 - Acquisition of speech samples

4.5.1 Sampling

Sampling consists of breaking up the waveform at constant time intervals. The sample rate then becomes the number of divisions taken per one second of audio. Sampling is the acquisition of a continuous signal at discrete time intervals.

Sampling theorem

If the highest frequency component in a signal is f_{max}, then the signal should be sampled at least $2f_{max}$ for the samples to describe the signal completely. This is known as the Nyquist Sampling Criteria:

$$F_s \geq 2 \times f_{max} \quad (4.1)$$

Although human speech ranges from 0 Hz to 20 kHz, the majority of the useful signal content lies between 300 Hz and 4000 Hz. We thus take $f_{max} = 4$ kHz. Therefore the minimum sampling rate must be :

$$F_s \geq 2 \times f_{max}$$

$$F_s \geq 2 \times 4 \text{ kHz}$$

$$F_s \geq 8 \text{ kHz}$$

Therefore, a minimum sampling rate of 8 kHz is required to represent the analog signal with discrete digital samples (Nyquist sampling criterion). A sampling rate of 16 kHz was used during the MATLAB implementation of the recognition system. A sound card with a resolution of 16-bits was chosen to minimize the quantization noise. These samples are then stored in a particular sound file format on the PC hard drive using "Cooledit" sound recording software [31]. "Cooledit" is shareware software, which is available on the Internet.

Quantisation and resolution

Before conversion to digital form, the analog sample is assigned one of 2^B values [13]. This process termed quantisation introduces an error, which cannot be removed. The level of the error is a function of the number of bits of the ADC. The quantisation step size, q is given by:

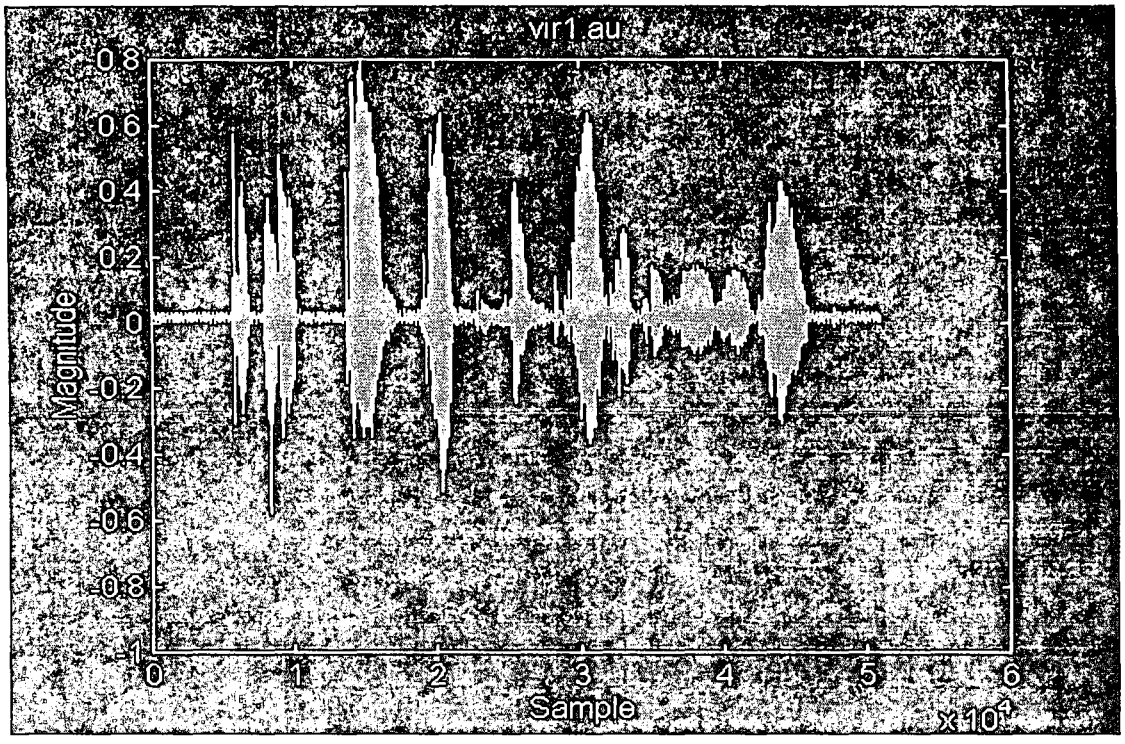
$$q = V_{fs} / 2^B \quad (4.2)$$

where V_{fs} is the full-scale range of the ADC. The maximum quantisation error, for the case where values are rounded up or down, is $\pm$ q /2. Thus quantisation noise is dependent on the bit resolution of the ADC.

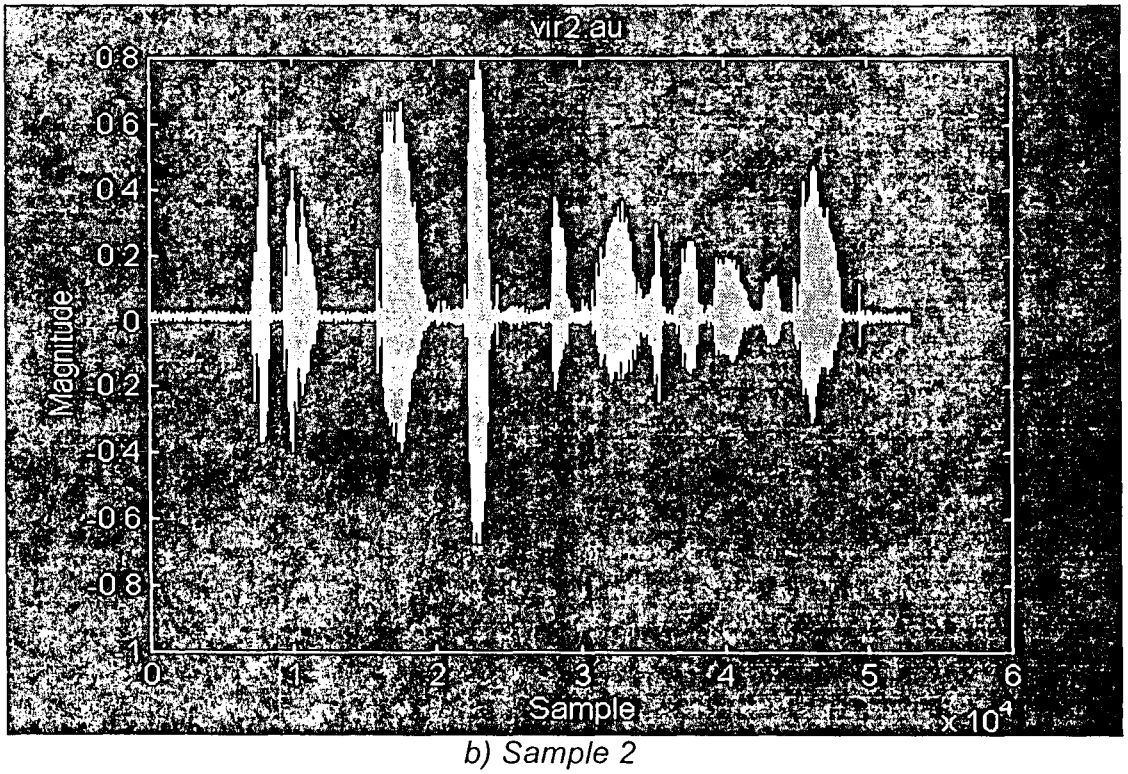
Normally 8, 12 and 16 bit resolutions are common for PC sound cards. 16-bit resolution was used to minimize distortion and noise during analog to digital conversion.

Figure 4.3. shows the similarities of 2 different discrete voice samples from the same speaker (vir). The difference in the discrete voice samples of two different speakers is shown in figure 4.4 (speakers vir and kas). The data plotted can be used as a means to distinguish between speakers since all feature vectors are derived from this data.

However, it is needed for the speaker recognition process to be completely automated and performed by a machine. The use of this large amount of input data to a pattern recognition network makes the demand on the machines computational and storage resources too expensive. Hence, the need for preprocessing prior to the pattern recognition process.

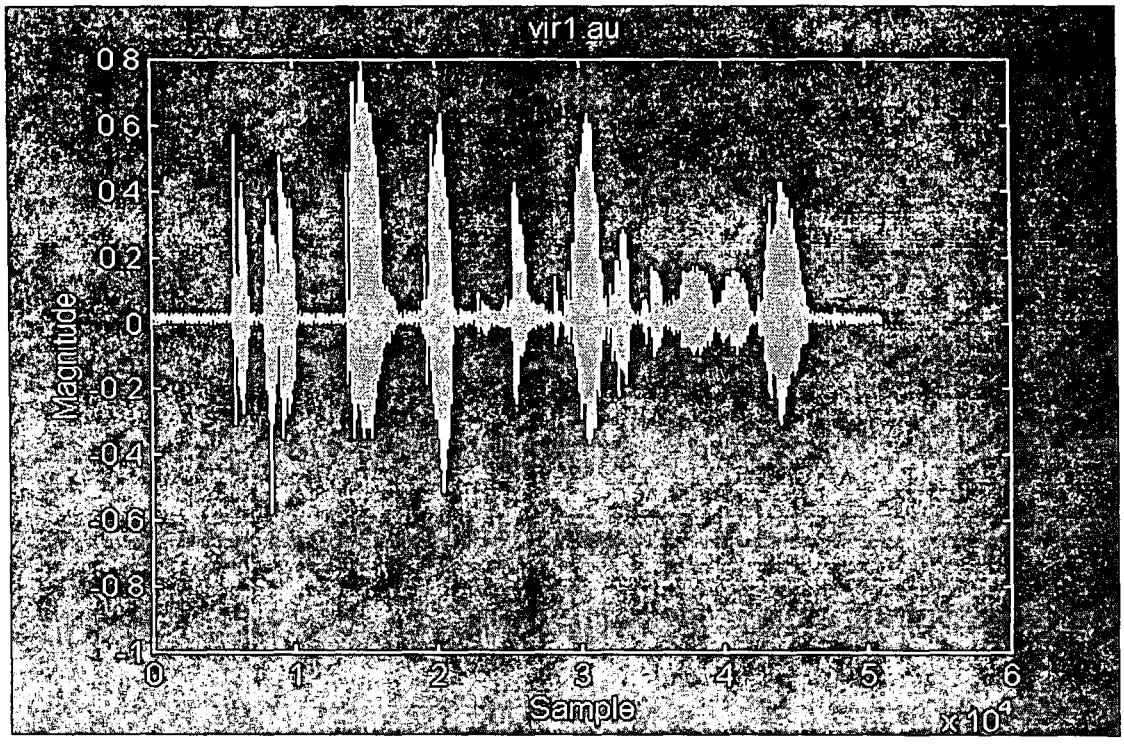


a) Sample 1

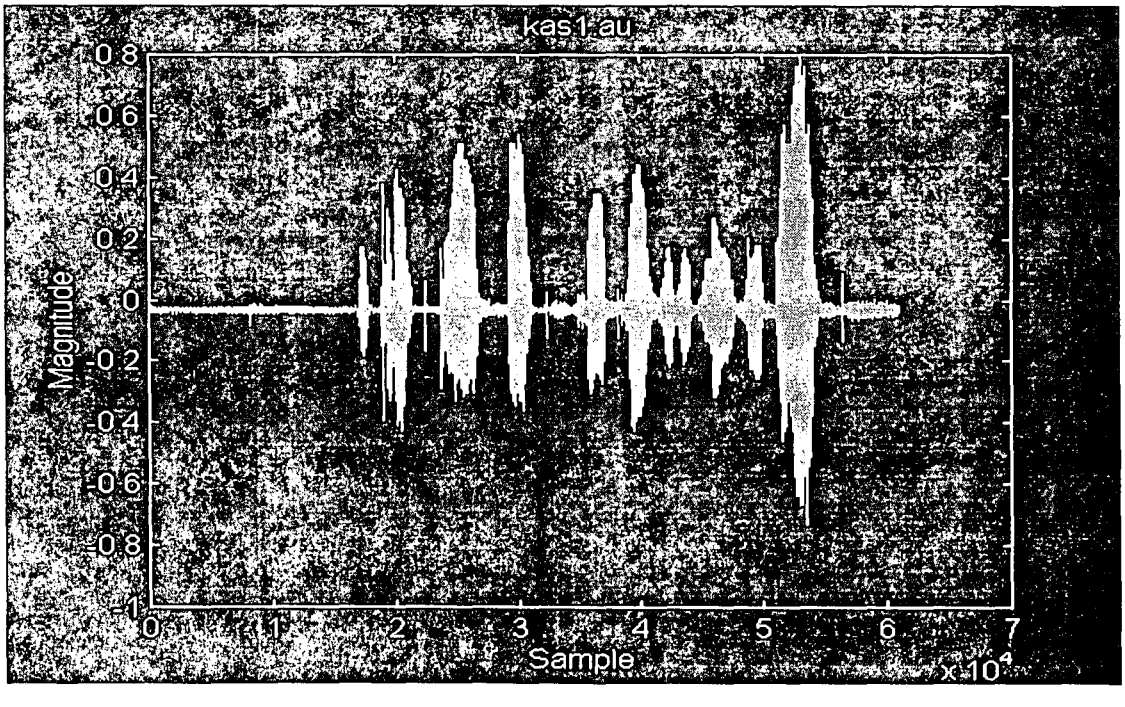


b) Sample 2

Figure 4.3 - 'Au' format plot of samples from speaker 'vir'



a) Sample 1



b) Sample 2

Figure 4.4 -'Au' format plot of samples from speakers 'vir' and 'kas'

4.5.2 Speech file formats and storage

Once the speech samples have been acquired they were stored on the PC hard drive. Digital speech samples may be stored as one of several formats. The extension of the stored filename is an indication of the type of speech format. A description of various sound speech formats may found in Appendix A.

Since the speech files were interfaced to the MATLAB environment it is necessary to choose a file format that is supported by MATLAB. MATLAB supports two types of file reading formats, WAV and AU. Speaker samples were stored as .AU in the Quick database while .WAV is used by YOHO. This way of recording the speech samples and saving in AU format allows convenient interface into MATLAB software environment. Figure 4.4 shows a plot of an AU format file.

4.6 Feature Parameters

The characteristic feature parameters that describe the unique properties of each speaker need to be extracted from the stored speech samples in section 4.5.2. This is accomplished with MATLAB's signal processing toolbox [32]. Few traditional 'neural network' algorithms have been meant to directly operate on raw data, such as pixels of an image, or samples of speech waveforms picked up from the time domain. Most pattern recognition tasks are preceded by a preprocessing transformation that extracts invariant features from the raw data, such as spectral components of acoustical signals, or elements of co-occurrence matrices of pixels. Selection of a proper preprocessing transformation for a particular task usually requires careful consideration.

4.6.1 Power spectral density (PSD)

As described in section 2.9.4, the PSD representation of a signal seeks to describe the spectral properties of a finite set of data and is computed by taking the magnitude-squared result of the Fast Fourier Transform (FFT) of the speech signal (section 2.9.3). The MATLAB code used to compute FFT and PSD is given in Appendix C.3.

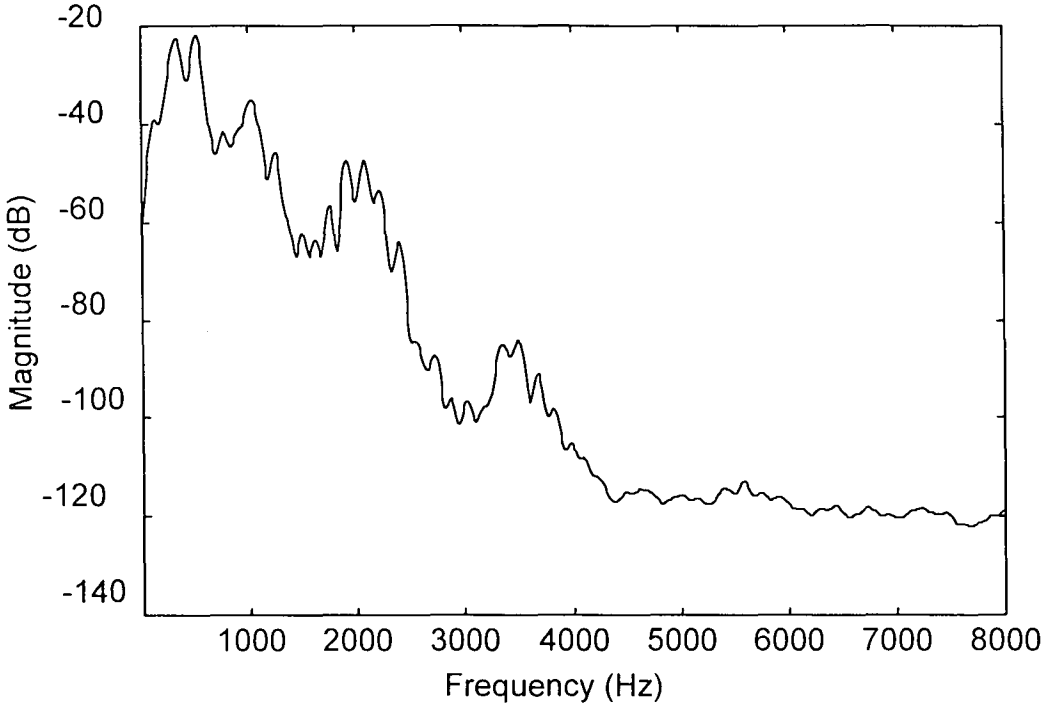
Figure 4.5 shows the PSD plots of two different voice samples of the speaker 'sat'. Similarities in the power distribution across the spectrum between the two plots of Figure 4.5 can be noticed.

Figure 4.6 contrasts the PSD plots of the speakers 'sat' and 'kum'. Differences in the spectral distribution are evident. PSDs have therefore been considered as a characteristic feature parameter that may be used to differentiate between speakers. These have been used in this study with limited success. The results of using PSDs as inputs to the pattern classification networks (backpropagation and LVQ neural networks) are presented later in this chapter.

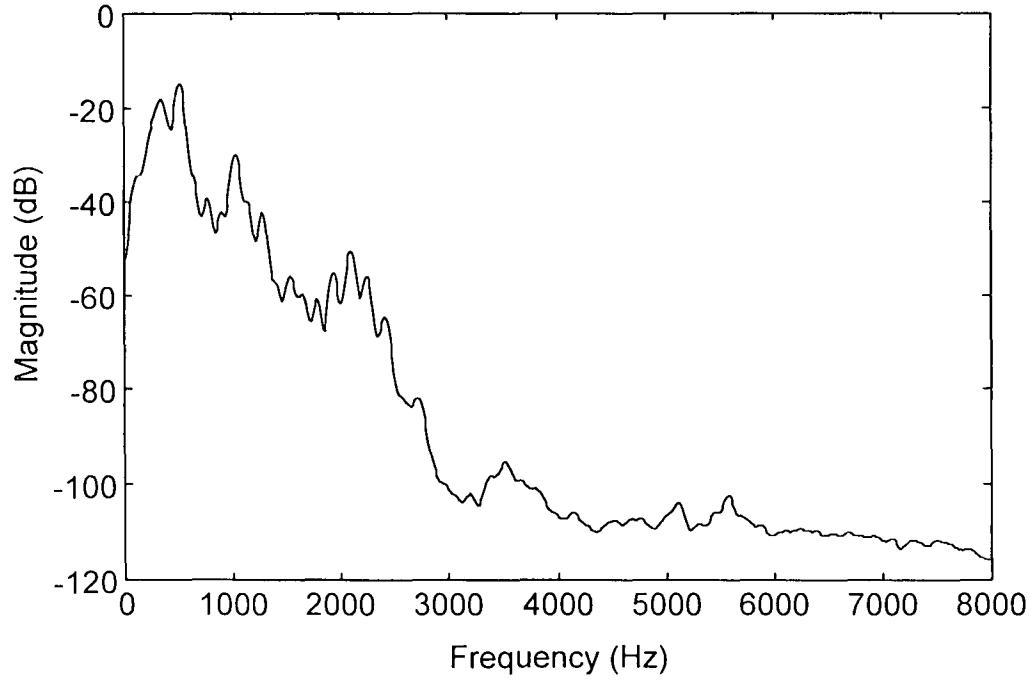
Table 4.2 shows the parameters used to calculate the PSD coefficients of figures 4.5 and 4.6.

Table 4.2 - PSD parameters

Sampling frequency (Fs)	16000 Hz
FFT length (NFFT)	1024
Window	Hanning
Window Size	256
Overlap	128

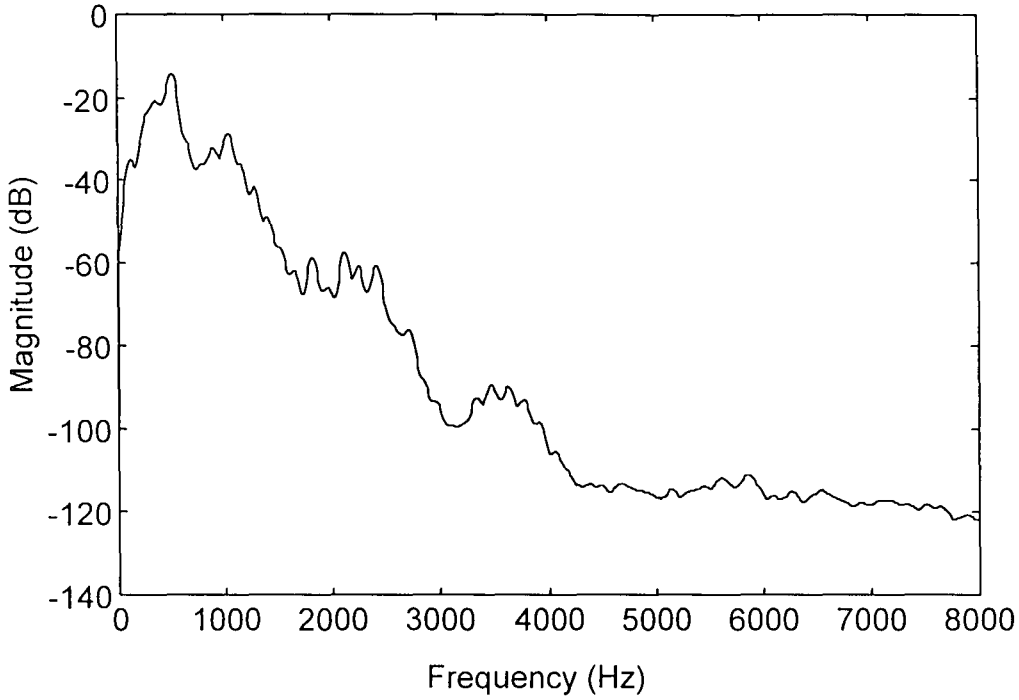


(a) Sample 'sat1'

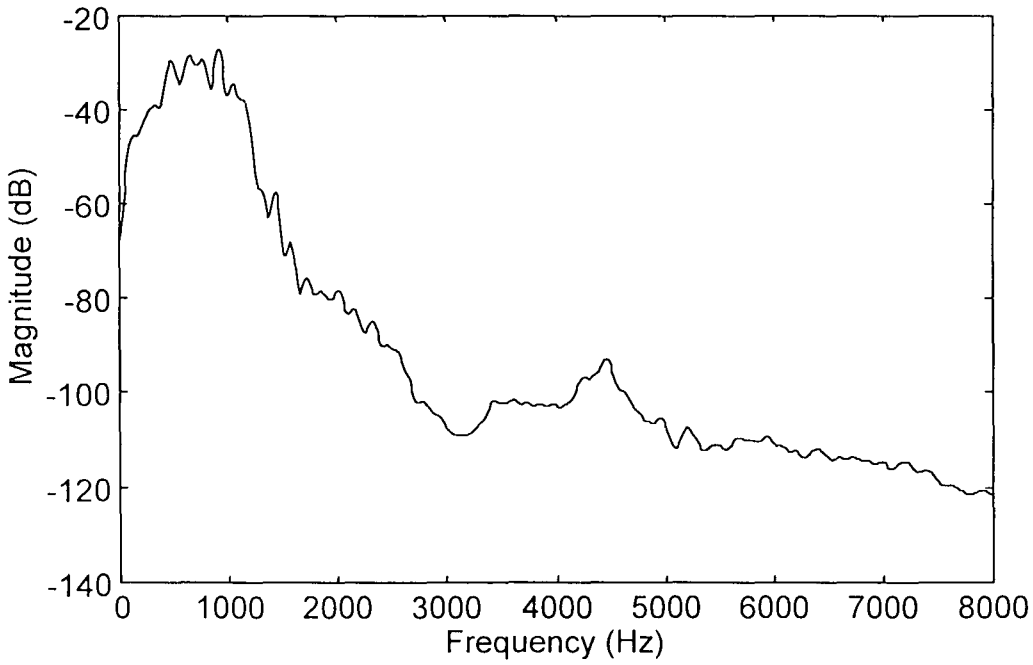


(b) Sample 'sat2'

Figure 4.5 - PSD Plots of the same speaker



(a) Sample 'sat3'



(b) Sample 'kum1'

Figure 4.6 - PSD plots of different speakers

Tables D.1 and D.2, in Appendix D.1, shows the correlation of different PSD vectors. It can be seen, from these results, that an intelligent classifier is required to distinguish between PSD samples of different speakers as the simple cross-correlation method sometimes produces higher correlation between the samples of different speakers than the samples of the same speaker.

4.6.2 Linear predictive coding

The LPC features were very popular in the early speech-recognition and speaker-verification systems [4]. The MATLAB signal processing toolbox contains a function that computes the coefficients of a *Nth* order auto-regressive process that models the time series X as follows:

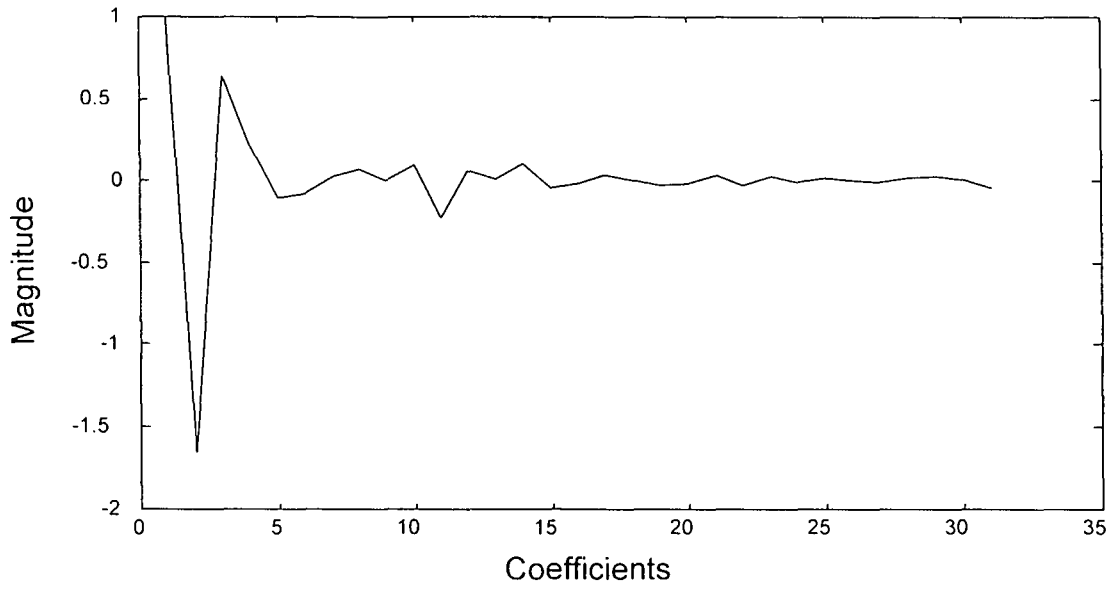
$$X(n) = -A(2)*X(n-1) - A(3)*X(n-2) - \ldots - A(N+1)*X(n-N-1) \qquad (4.3)$$

Here, X is the real input time series (a vector), and N is the order of denominator polynomial A(z), i.e. A = [1 A(2) ... A(N+1)].

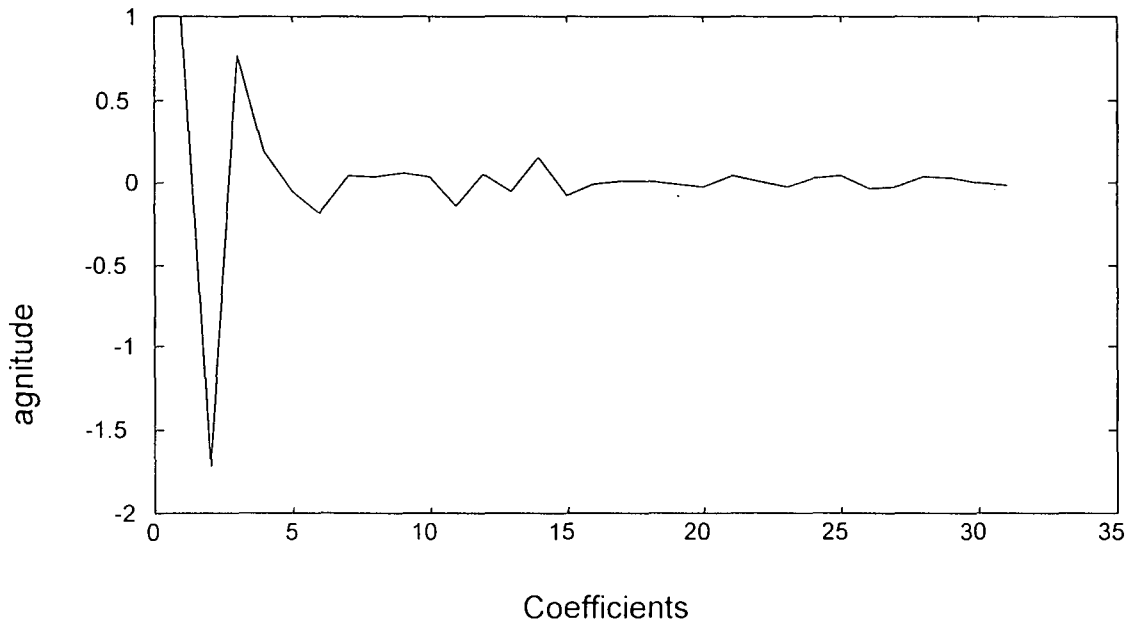
The MATLAB code for the LPC function as well as the code written to produce the LPC plots in figures 4.7 and 4.8 below are listed in Appendix C.4.

Figure 4.7 shows the similarities in LPCs between different speech samples from the same speaker. Figure 4.8 shows contrasts in LPCs of samples from different speakers.

Figure 4.9 demonstrates that the majority of useful information contained in LPCs lies in the first 20 coefficients. Thereafter the plot "smoothes" out. It is not therefore too beneficial to calculate more than 30 LPCs.

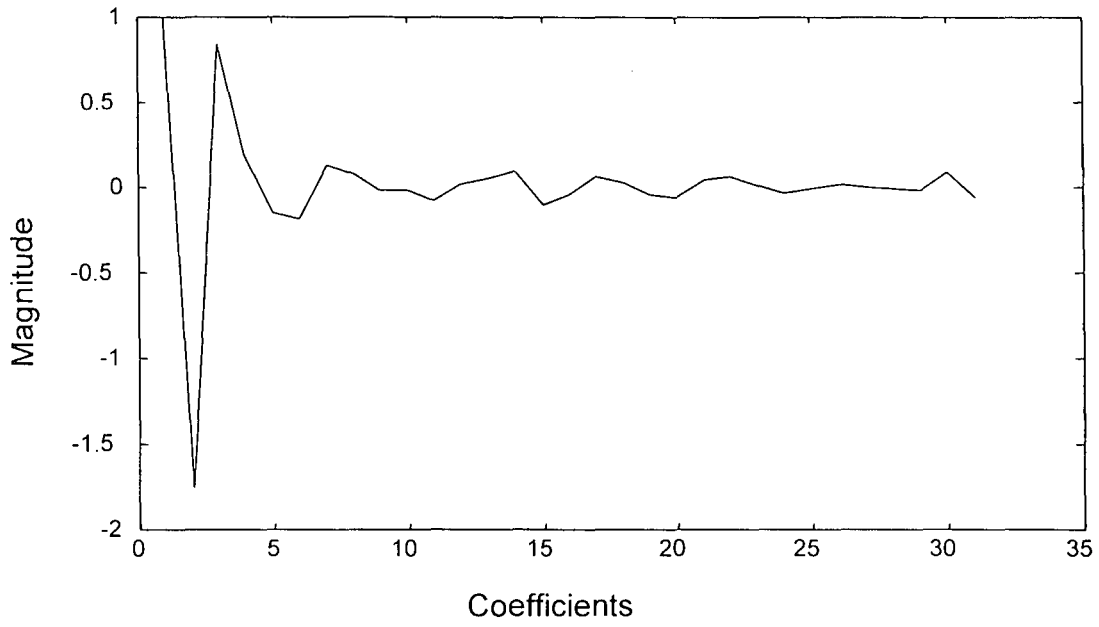


(a) Sample 'suz1'

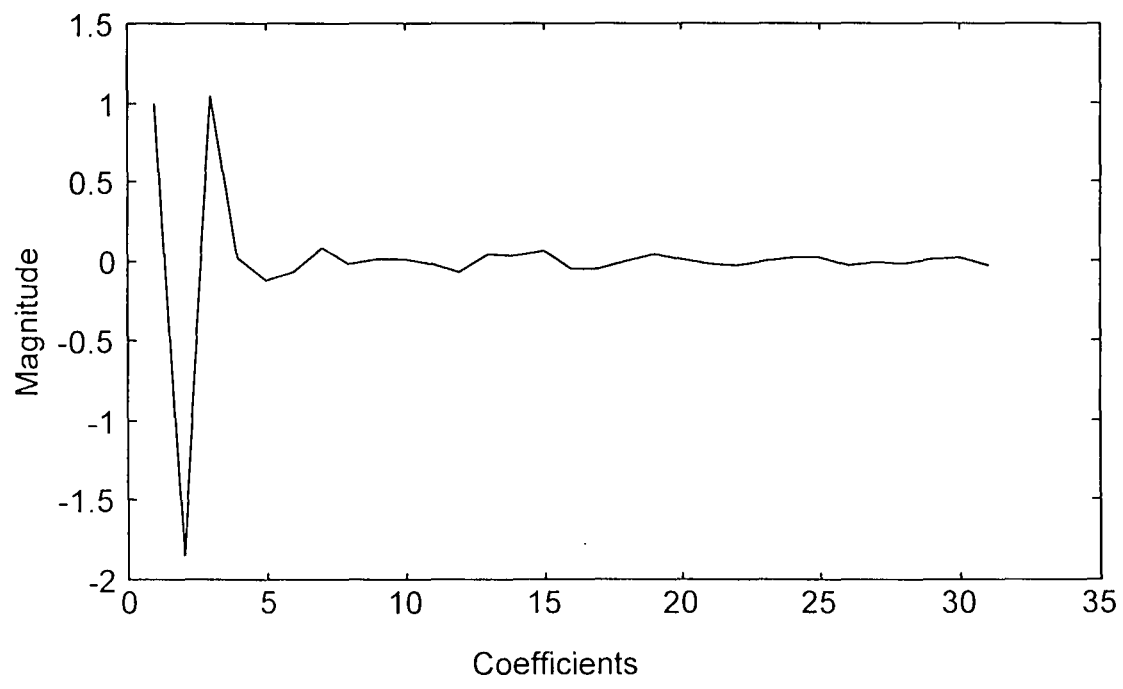


(b) Sample 'suz2'

Figure 4.7 - 30-point LPC plot of the speaker 'suz'

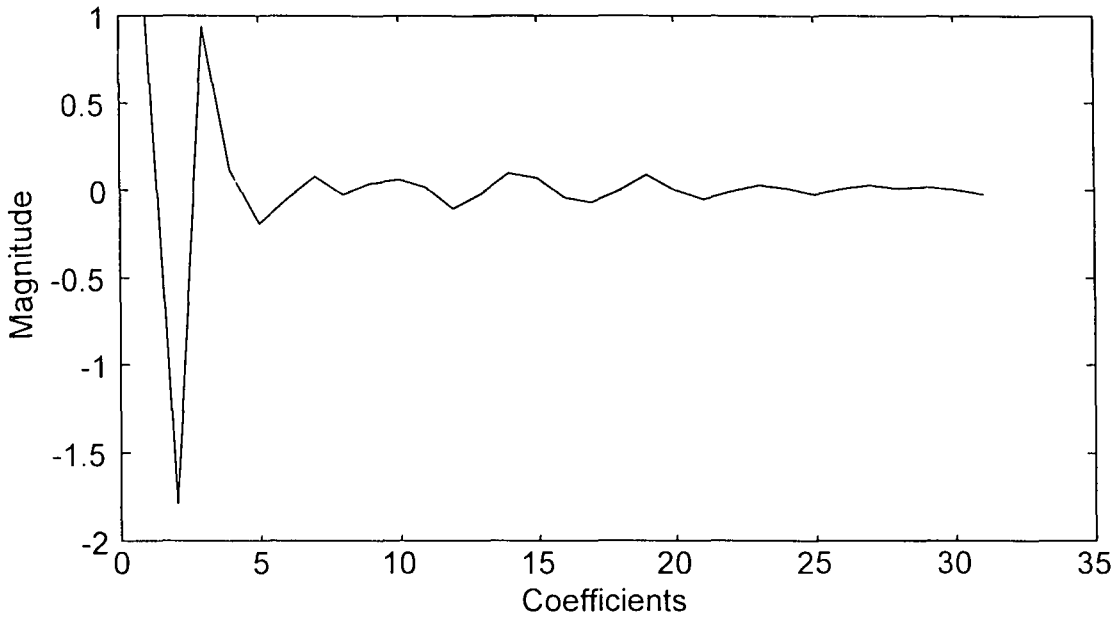


(a) Sample 'suz3'

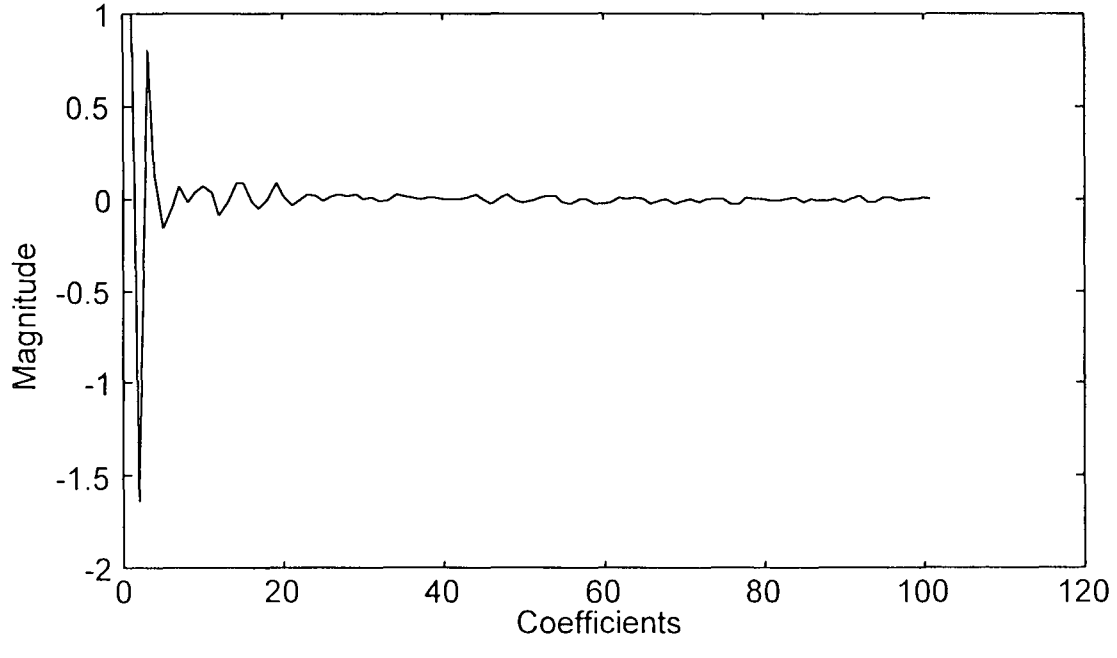


(b) Sample 'kas1'

Figure 4.8 - 30 point LPC plot of different speakers



(a) 30 point



(b) 100 point

Figure 4.9 - 30 and 100 point LPC plot of kas2

Figure 4.10 shows a flowchart of the method used to calculate the LPC coefficients.

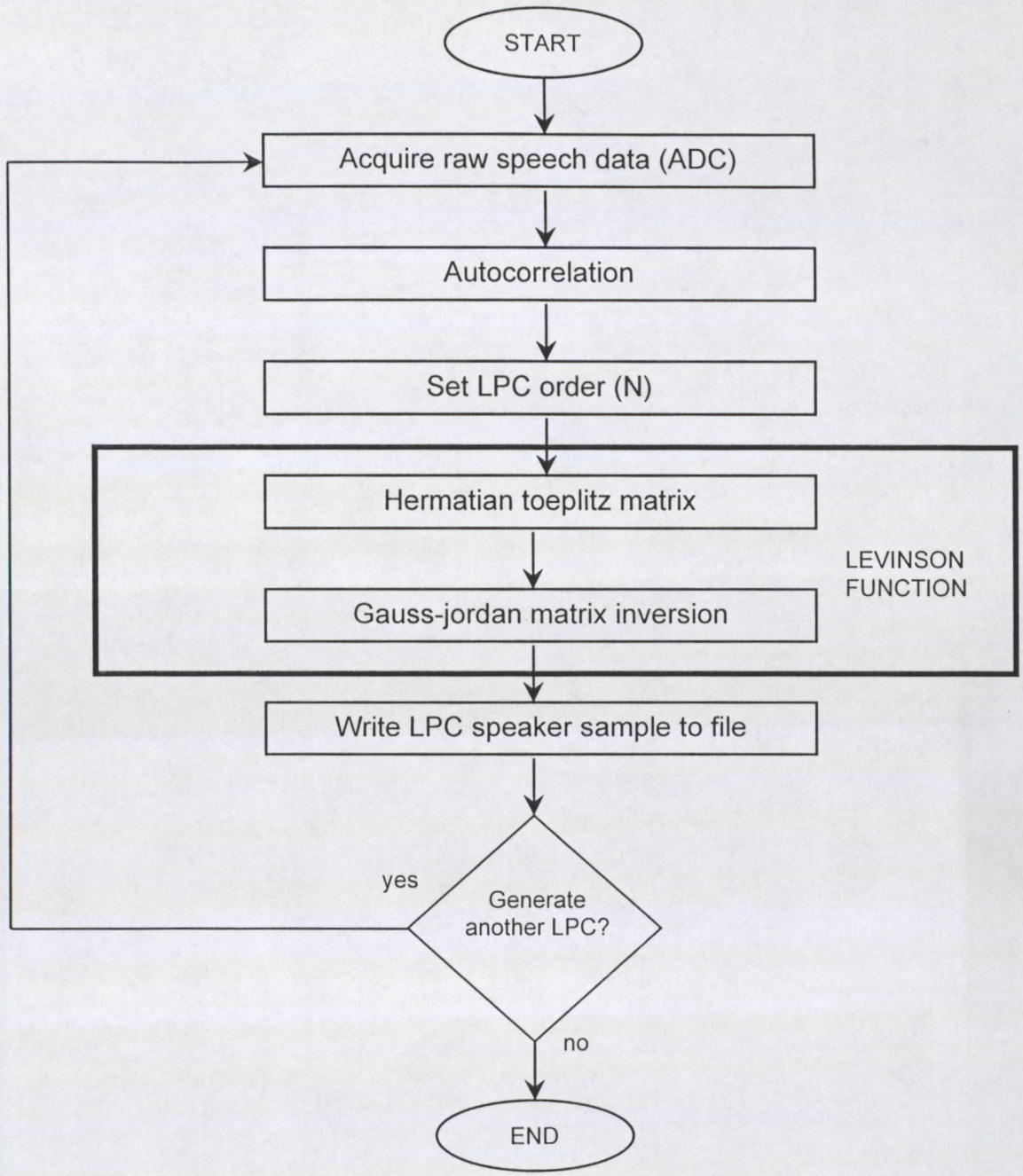


Figure 4.10 - Software flowchart for LPC coefficient derivation

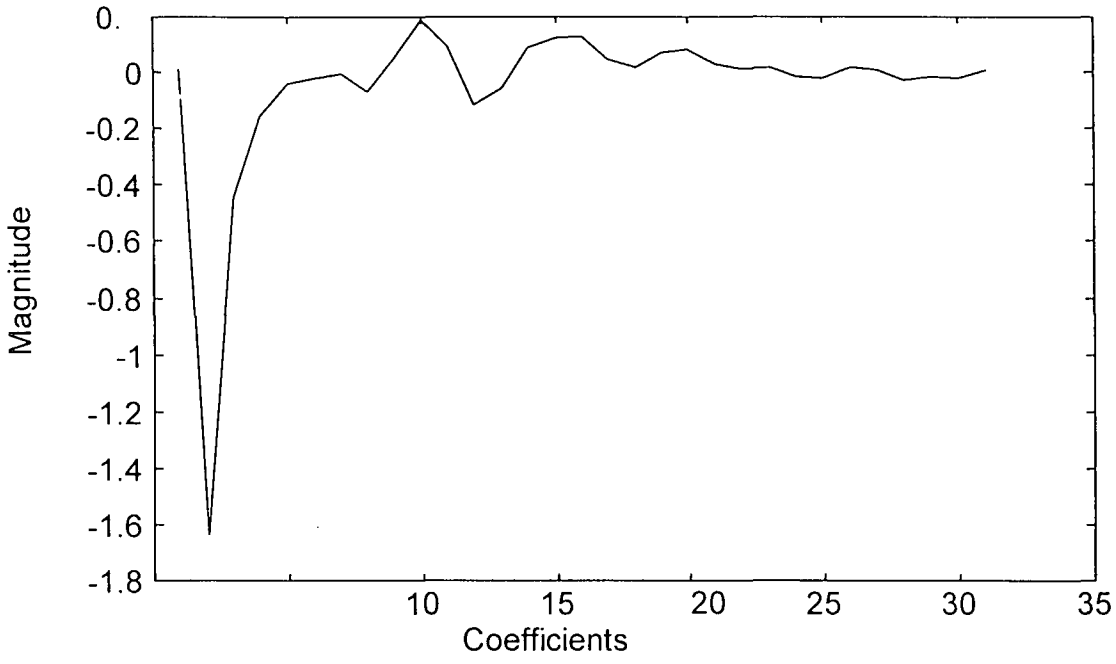
4.6.3 Cepstral analysis

The comparison of two LPC feature vectors requires the use of computationally expensive similarity measures such as the Itakura-Saito distance and hence LPC features are unsuitable for use in real-time systems. Furui suggested the use of the Cepstrum, defined as the inverse Fourier transform of the natural logarithm of the magnitude spectrum, in speech-recognition applications [32]. The use of the Cepstrum allows for the similarity between two Cepstral feature vectors to be computed as a simple Euclidean distance.

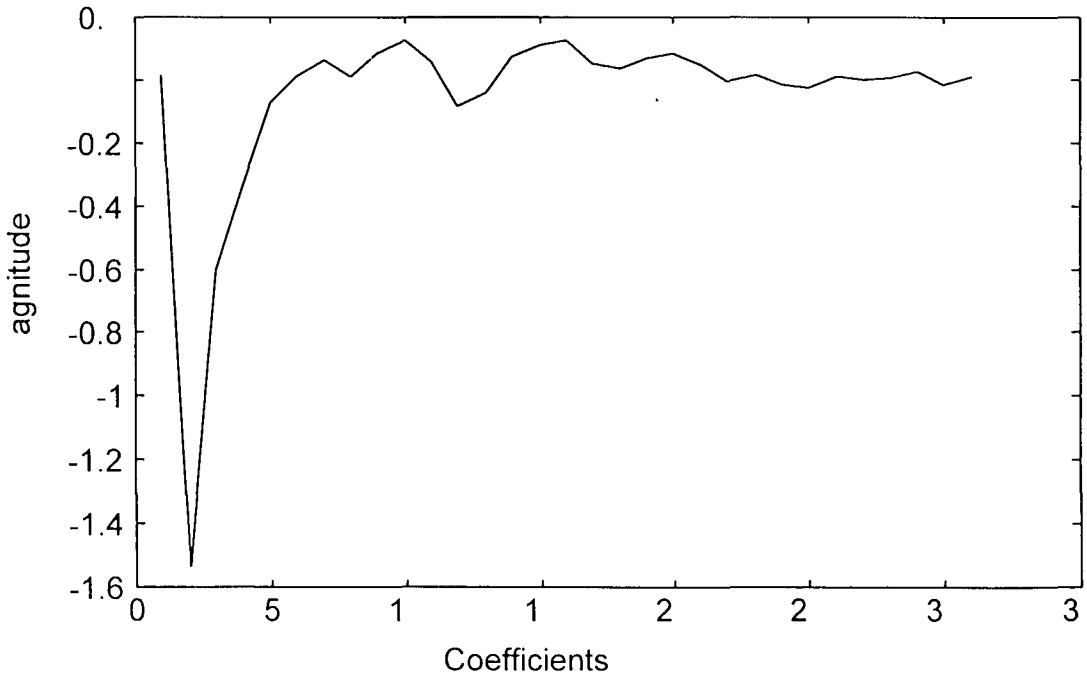
However, calculating the complex Cepstrum directly from the raw 'au' signal is computationally expensive (16000 samples per second x 3 seconds speech recording = 48000 samples per speech recording). The complex Cepstral features shown below were derived from the 100 point LPC coefficients of the speech samples and not the raw recorded speech sample. Thus it is a hybrid of LPC and Cepstral parameters, hereafter referred to as hybrid complex Cepstrum. LPC coefficients are very good representations of the original waveform since the original waveform can be reconstructed from these coefficients. One of the most common short term spectral measurement currently used is the Linear Predictive Coding (LPC) derived Cepstral coefficients and their regression coefficients. A spectral envelope reconstructed from a truncated set of Cepstral coefficients is much smoother than one reconstructed from LPC coefficients. Therefore it provides a more stable representation from one repetition to another of a particular speaker's utterances [43]. Consequently, it was decided to use the LPC derived Cepstrum for this speaker verification system.

This hybrid feature parameter can now be used to train and then test the recognition system. Appendix C.5 contains the MATLAB code for calculation of the complex Cepstrum (CCEPs). Figures 4.11 and 4.12 show the CCEPs (LPC-derived) of different speakers.

This study has concentrated on LPCs and the hybrid CCEPs as feature vector inputs to the neural network in figure 4.1.

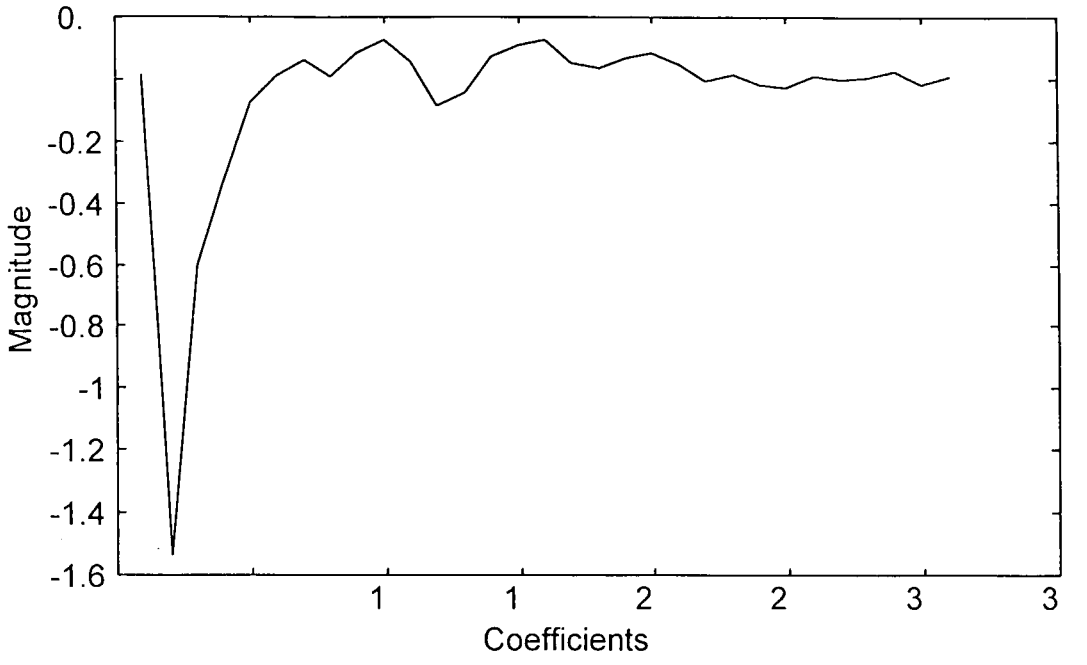


(a) Sample 'tam1'

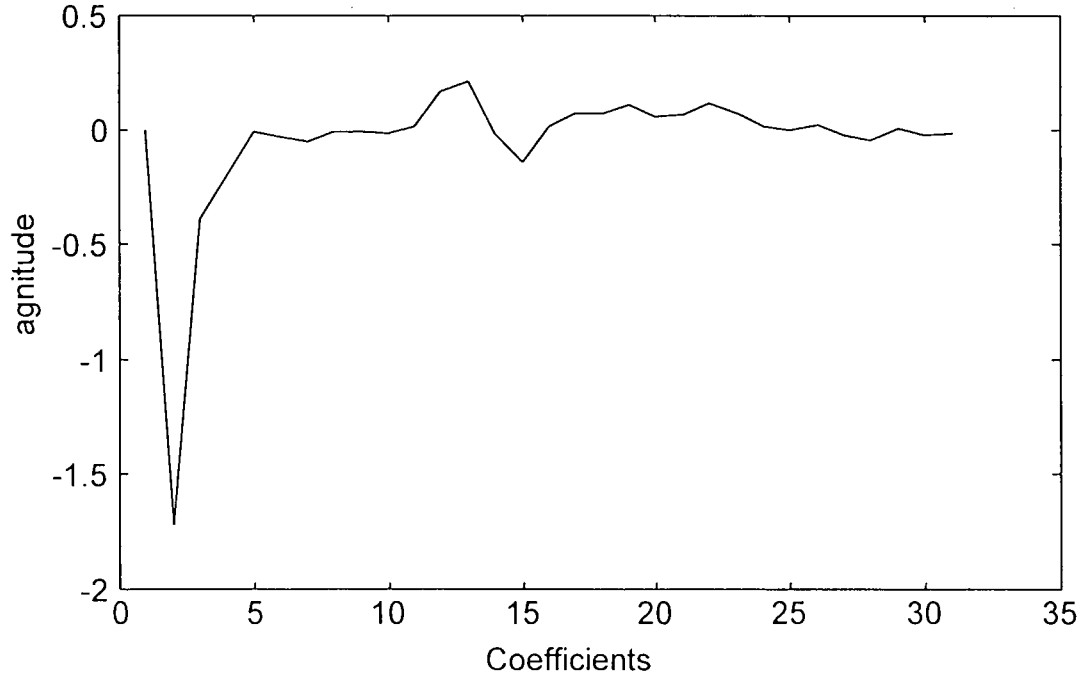


(b) Sample 'tam2'

Figure 4.11 – 30-point CCEPs plot of speaker 'tam'



(a) Sample 'tam3'



(b) Sample 'kub1'

Figure 4.12 - 30-point CCEPs plot of speakers 'tam' and 'kub'

Figure 4.13 shows a flowchart of the method used to calculate the LPC derived Cepstral coefficients.

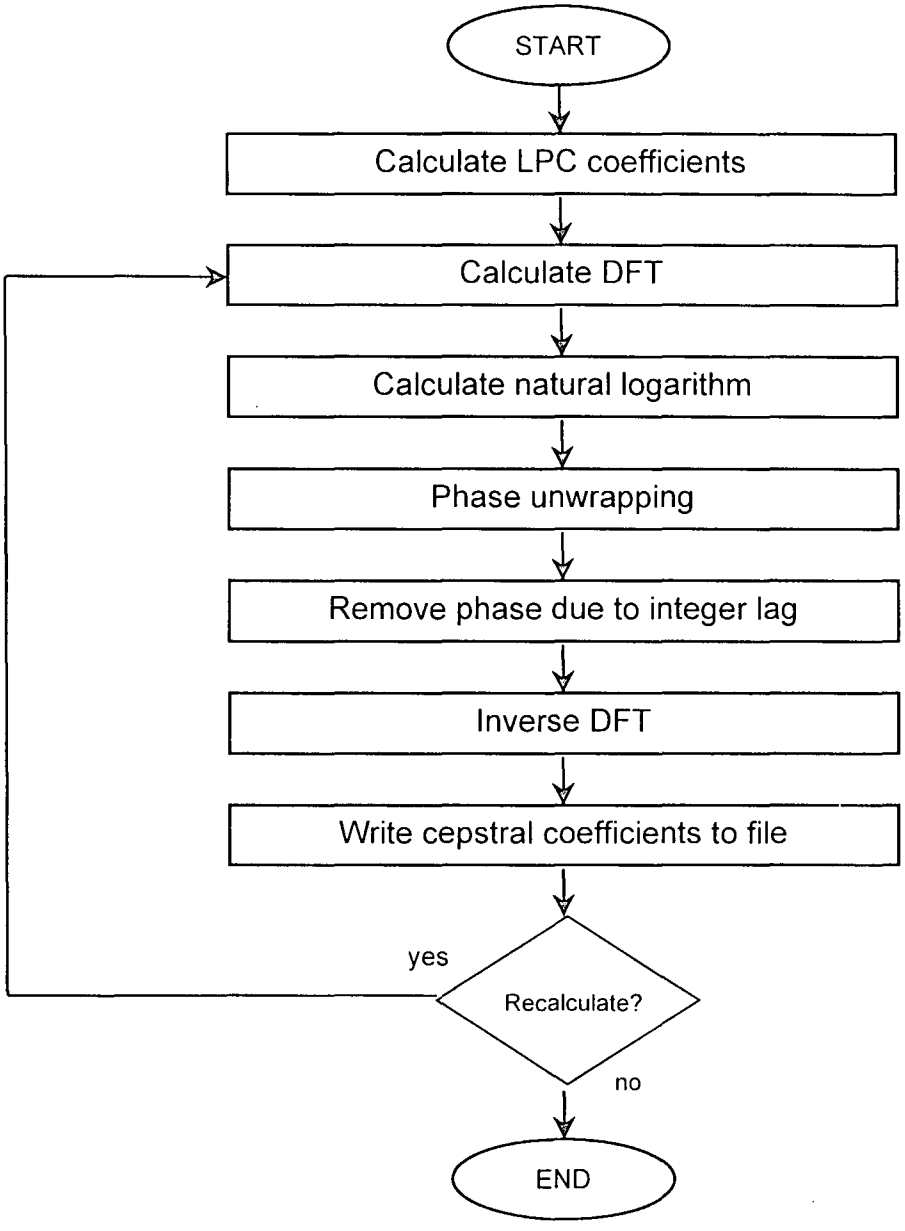


Figure 4.13 - Software flowchart of the Cepstral coefficient derivation

4.7 Neural Network Implementation

Artificial Neural Networks (ANNs), as described in Chapter Three, are intelligent systems that are related in some way to a simplified biological model of the human brain. They are fault tolerant, immune to noise and have applications in system identification, pattern recognition, classification, speech recognition, image processing, etc. For example, backpropagation neural networks have been used to identify bird species using recordings of birdsong [33].

The MATLAB neural network toolbox [25] contains a comprehensive list of neural network architectures that may be initialized, trained and simulated. This study has concentrated on the Multi-Layered Perceptron (MLP) feedforward neural network, with backpropagation training and the Self-Organizing Map (SOM) neural network architectures for speaker recognition [25]. In addition, a committee of neural networks was also implemented for speaker recognition in the MATLAB implementation.

4.7.1 Multi-layer perceptron (MLP) feedforward neural network implementation

A MLP neural network, consisting of a hidden layer of sigmoidal functions followed by a linear output layer, is used for the speaker recognition system [1]. The network was trained using the backpropagation method.

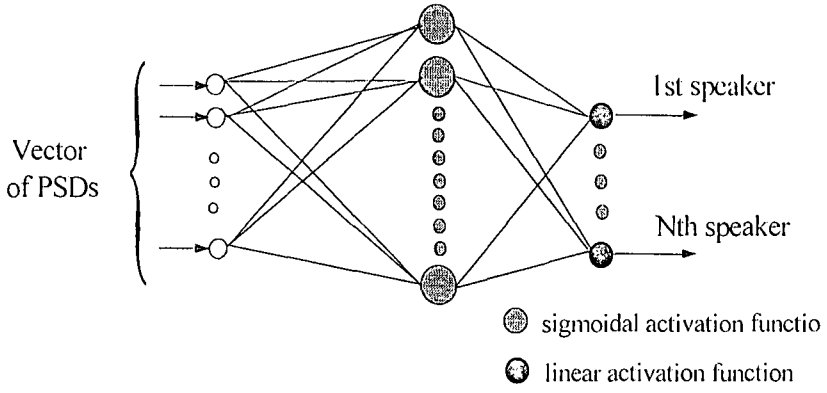
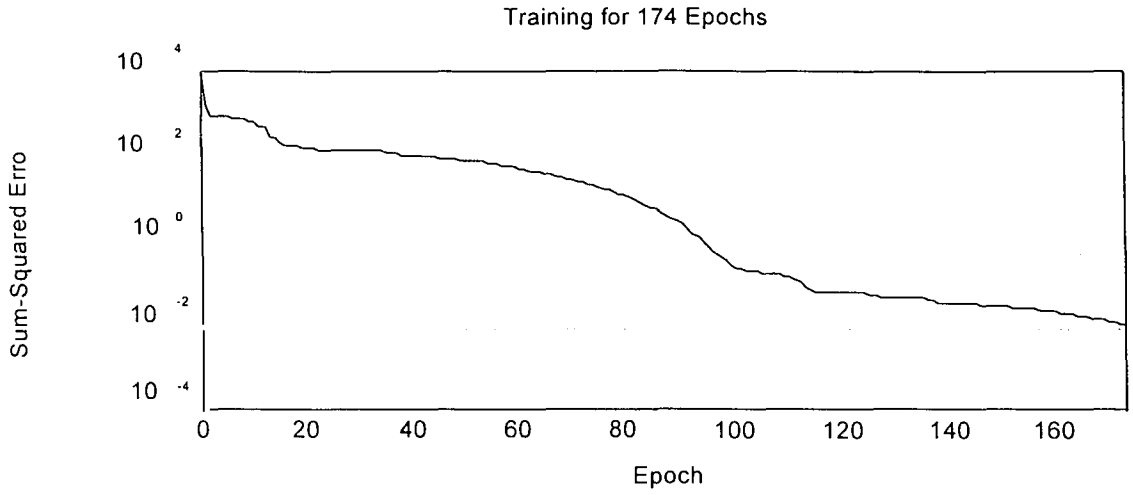


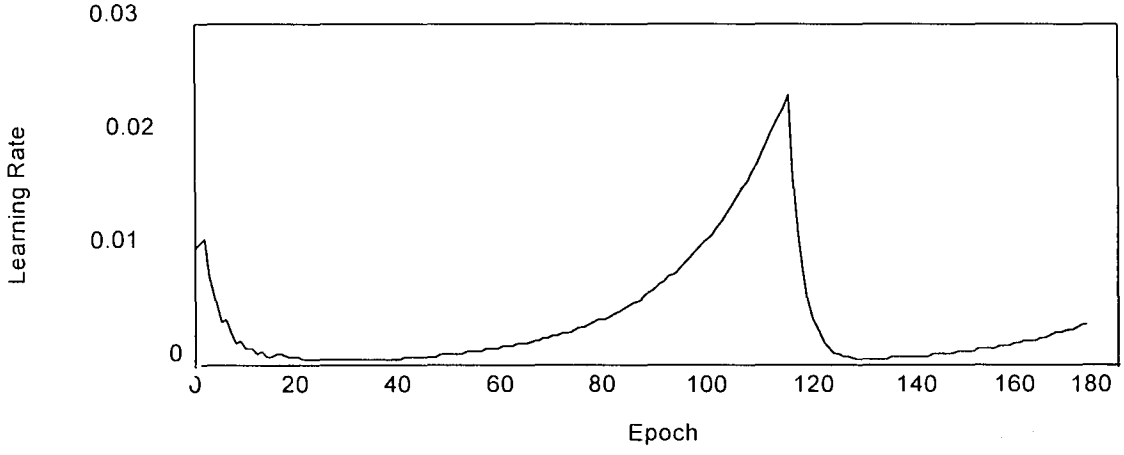
Figure 4.14 – MLP feedforward neural network

The output of the hidden layer, which enables the network to process non-linear data, is constrained from -1 to 1. It was therefore convenient to add the linear layer so that the output may now assume values ranging from negative infinity to infinity which are used as target values for each speaker. The target values for the speakers are now not constrained to values from –1 to 1. The MLP neural network based speaker recognition system was implemented to recognize five speakers.

Figure 4.15a shows the convergence of the SSE to the specified minimum error goal in the training of the MLP network. An adaptive learning rate is used to train the network. Figure 4.15b shows how the learning rate is adapted with each training cycle. The MATLAB code for the multi-layered network training is listed in appendix E.2.



(a) Convergence of SSE to error goal



(b) Variation of learning rate

Figure 4.15 - Training progress of the backpropagation network (Network 1)

Previous studies, using the same speaker database, have shown a 66% recognition success rate using PSDs as input to a MLP neural network [1]. Further, the success rate increased to 89% when the number of speakers in the group was reduced to three. This shows that increased group-size affects the success of the MLP based system negatively.

4.7.2 Learning vector quantization implementation

Although MLP feedforward neural networks are very well suited for function approximation they are not ideal for pattern classification tasks. It has been shown in [1] that the LVQ neural network, which is a supervised variant of the

Self-Organizing Map (SOM), is a neural network architecture that can be used for pattern recognition by the clustering of hidden-layer neurons towards particular inputs. Associations are made between the input vectors and certain neurons of the network. This network is very useful for the classification of input vectors into user specified categories. The objective of the LVQ network implementation is to improve the speaker recognition accuracy of 66% obtained with the MLP neural network.

A LVQ network was used to initialize, train and test the same 5-speaker group above. Instead of the elements of the output vector being real values, they are chosen as pre-defined categories (integers). The LVQ network, unlike the MLP network, does not approximate the output. This is illustrated below.

For example, Let P be the input matrix, given by

P=[vir1 vir2 vir3 sat1 sat2 sat3 kas1 kas2 kas3 suz1 suz2 suz3 kum1 ... kum2 kum3]

and the corresponding output vector C is taken as

C= [1 1 1 2 2 2 3 3 3 4 4 4 5 5 5]

where each column of the input matrix P is a feature vector of a particular speaker. The example above shows P having three samples each from speakers vir, sat, kas, suz and kum. These are the samples used in the training mode of the system.

The vector samples from each speaker are used to train the network to classify test vectors into one of five categories. Instead of specifying the maximum number of epochs, the number of training epochs is specified since there is no error goal in this network. The output will be one of the target classes and not an approximation. A matrix containing the known speaker input vectors were presented to the network during the training mode. The

target output vector for this matrix was also defined. That is, if a speech vector is fed to the neural network, the output of the network is a pre-assigned constant, representing a class, for each speaker. The **training** specifications that were specified to the network are specified in Table 4.5. Table 4.6 shows the success rates and training times for the networks with PSDs and LPCs.

Table 4.3 - Training Parameters for LVQ Network (network 2)

Number of epochs	10000
Initial learning rate	0.010
Group size (No. of speakers)	5
Number of training samples per speaker	7
Number of test samples per speaker	3
Number of neurons in the hidden layer	30
Number of inputs/ Size of input vectors	
- PSDs	500
- LPCs	20

Table 4.4 - Test results of the LVQ network (network 2)

Number of speakers	5
Approximate time to train network	
- PSDs	17 mins.
- LPCs	15 mins.
Recognition Success Rate ("unseen" vectors):	
500 pt. PSD	90%
20 pt. LPC	100%

Table 4.6 shows that LVQ neural networks significantly improve the performance of the speaker recognition system. The 66% accuracy of the multi-layered backpropagation neural network is improved to 90% by the LVQ network when using PSD inputs. Therefore, LVQ neural networks were used

as the pattern classifier for all further speaker trials. The MATLAB code used to generate the results shown in Table 4.6 is listed in Appendix E.2 and E.3.

4.8 Increasing the Group-Size

One of the objectives of this study is to increase the group-size whilst obtaining the 90%-100% recognition success rate achieved with the 5-speaker group. The network specifications used in the training process of the neural network are shown in table 4.7. Table 4.8 shows the performance of the LVQ system when the number of speakers in the group was increased to eight and then ten subjects. It also shows the success rates achieved when using each of the following characteristic feature vectors as input to the system: PSDs, LPCs, a hybrid feature vector consisting of a LPC vector appended to a PSD vector and the hybrid Cepstral-LPC coefficients. Hybrid feature vectors are employed to gain the benefit of different speaker characteristics rather than the use of single individual speaker characteristics.

Table 4.5 – Training parameters for 10-speaker network (network 3)

Type of neural network architecture	LVQ
Number of epochs	10000
Initial learning rate	0.010
Number of training samples per speaker	8
Number of test samples per speaker	2
Number of hidden layer neurons	30

Table 4.6 - LVQ Network test with different number of speakers

Number of speakers	**5** (%)	**8** (%)	**10** (%)
Recognition success using training vectors	100.0	100.0	100.0
Recognition success using PSD coefficients	90.0	68.8	45.0
Recognition success using LPC coefficients	100.0	75.0	70.0
Recognition success using PSD and LPC	100.0	75.0	70.0
Recognition success hybrid LPC and CCeps	100.0	93.8	90.0

Figure 4.16 also shows that 100% accuracy was achieved whenever training vectors were used to test the LVQ network, irrespective of the type of feature vector input used. All other results were obtained using unseen input vectors during the testing phase. The recognition rate of dropped to 45% when PSD coefficients were used as characteristic feature vector inputs to the 10-member group. Although PSD coefficients produce reasonable results, the LPC coefficients significantly improve the performance of the LVQ network. The use of a combination of feature vectors, comprising of a PSD vector simply appended to a LPC vector, did not noticeably enhance the performance of the system further. However, the use of a hybrid LPC-derived complex Cepstral coefficient (CCEPs) vector did improve the recognition rate to 90% with ten speakers in the group.

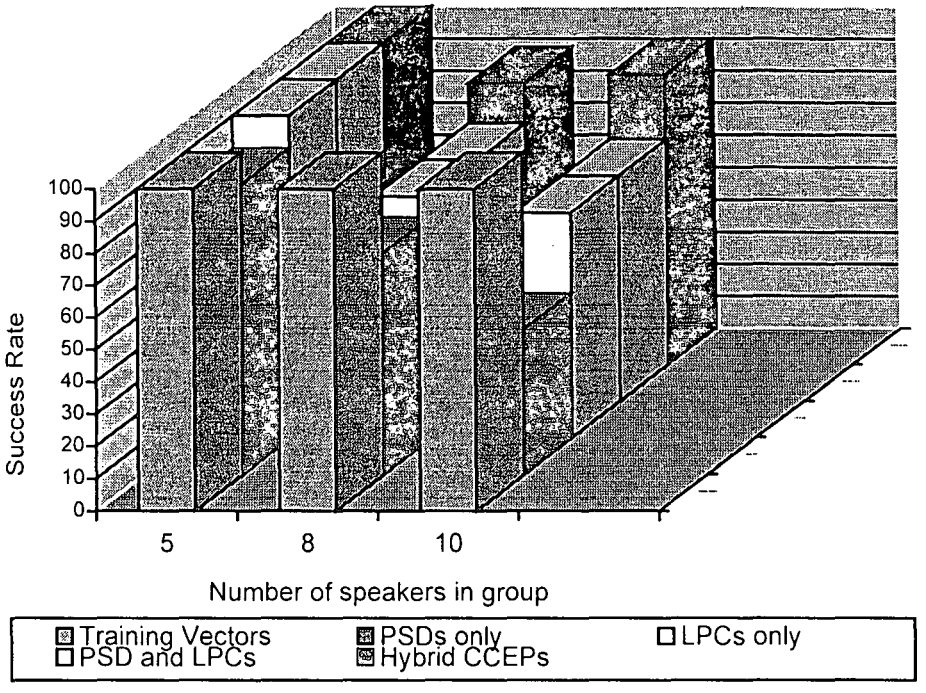


Figure 4.16 - Effect of group size

Section 4.9 describes the committee of neural network approach which maintains the accuracy of the system even when the group-size is increased to 20 speakers.

4.9 Committee Neural Network Implementation

The use of multiple neural networks, called the committee of neural network approach, has been proposed to improve the neural network decision making [7]. Using a committee of neural networks trained as a product of cross-validation can make improved predictions. The committee of neural networks is now used to improve the recognition accuracy of the system as the number of speakers in the group is increased from ten to twenty speakers.

Figure 4.17 shows a committee approach consisting of five LVQ member neural networks. Each member of the committee is trained and then tested

individually. The final decision was taken as the majority vote of the individual member networks.

Eight samples per subject were used to train the individual LVQ member networks. Two additional samples per subject were used to test the recognition success of the committee.

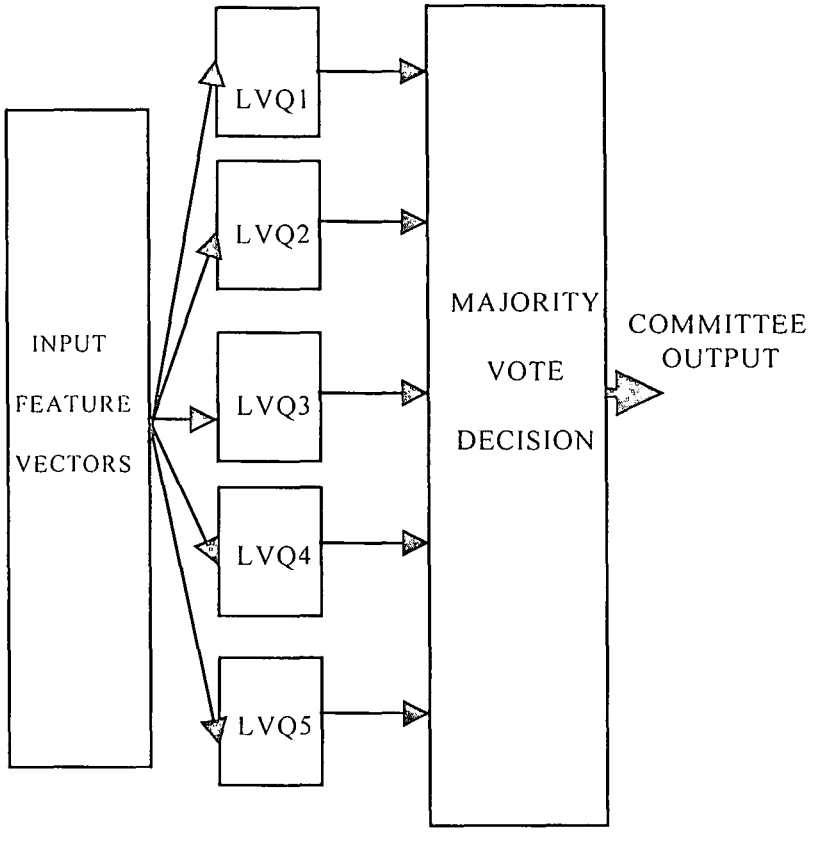


Figure 4.17 - Five-member committee neural network

The input vector, P = (R x 1), is the hybrid LPC-Cepstral feature vector per speech sample. This vector was scaled to 100 elements. The first 20 coefficients of a LPC vector contains the majority of the useful signal content. This placed a restriction on the minimum coefficients that can be used. The number of LPC coefficients was limited to 100 to maximize the speed of the speaker recognition system. These extracted feature vectors comprise fewer elements than the raw speech sample thus reducing the burden placed on the LVQ neural network.

The test results per input sample were similar for each of the member networks. This was due to the fact that all member networks were identical with the same network architecture, training data and weight initialization (vector W^1 as shown in figure 3.12). The resulting outputs of the member networks were similar since the LVQ training algorithm is identical in each case. This arrangement of the committee adds no value as compared to a single-network system. All individual networks behave identically to the test inputs and the committee decision is always unanimous.

Each member of the committee must process the input data using individual parameters. This results in outputs that can be cross-correlated to obtain the desired results from the committee. The S^1 parameter, shown in figure 4.16, is therefore defined differently for each member network. S^2 is the number of target classes in the linear layer. This value was fixed at 20 (number of speakers in the group) for all member networks.

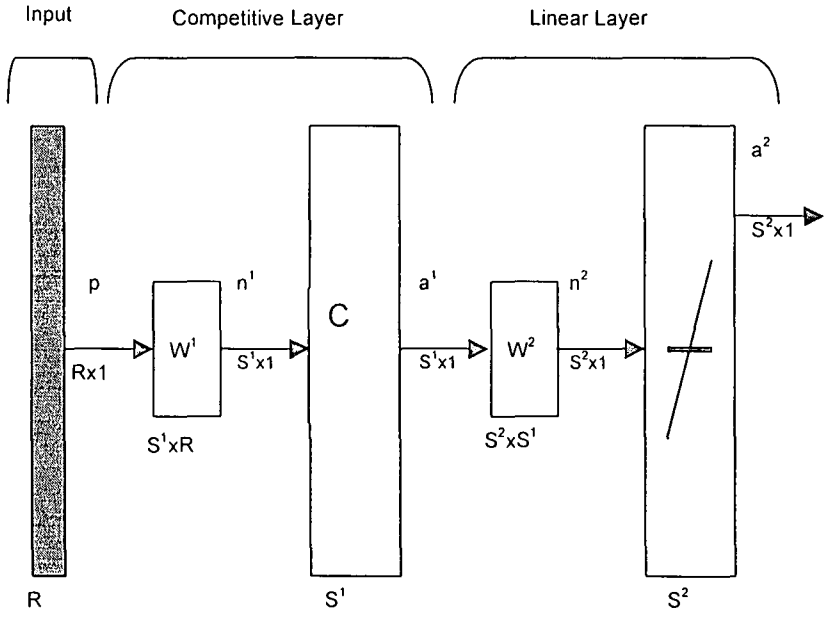


Figure 4.18 - LVQ specifications

The LVQ networks were trained with the training specifications shown in Table 4.7. The recognition accuracy of each individual LVQ network is given in Table 4.8.

Table 4.7- Committee neural network training specifications

Number of member networks	5
Initial learning rate (lr)	0.001
me (maximum epochs)	15000
Number of speakers/subjects	20
Training samples per subject	8
Test samples per subject	2
Total Test Samples	(20x2)=40

Table 4.8 - Recognition success of individual LVQ member networks

Member Network	**S^1**	**Recognition Success**	**Training Time (mins)**
LVQ1	20	90.0%	11
LVQ2	30	92.5%	14
LVQ3	40	90.0%	15
LVQ4	50	90.0%	17
LVQ5	60	77.5%	18

Although the overall success rate is similar for each individual LVQ member network (except LVQ5), the success rate per speaker sample differs. Particular samples are identified correctly by certain networks and incorrectly by others e.g. the other networks identify speaker sample bev1 correctly by LVQ1 and incorrectly by LVQ3. Committees of neural networks that consisted of three, four and five members have been described in Table 4.9 below. Appendix F lists the actual test results per speaker used to generate the results in Table 4.9 below.

Table 4.9 - Overall recognition success of the committee network

Committee	**Network Members**	**Recognition Success**
3-member	LVQ1, LVQ2, LVQ3	95%
4-member	LVQ1, LVQ2, LVQ3, LVQ4	95%
5-member	LVQ1, LVQ2, LVQ3, LVQ4, LVQ5	95%

Figure 4.19 shows that the committee of neural networks does improve the overall recognition success to 95%. In this case, with these specific speakers' samples, an increase in the number of members, from three LVQ networks to five, did not improve the recognition success rate any further.

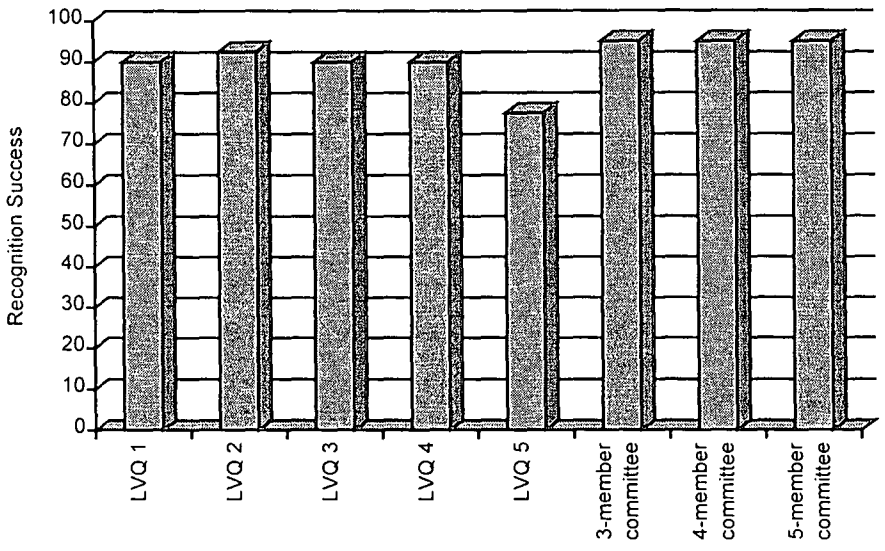


Figure 4.19 – Recognition success of the committee of neural networks

There is a significant increase in performance of the committee speaker recognition system than the individual LVQ5 network. The committee has produced a success of 95% compared to 77.5% obtained when using LVQ5 on its own.

Choosing the architecture of the committee more carefully can enhance the performance of the system further. LVQ 5, with S^1 equal to sixty, has an inferior success rate as compared to the other individual networks. It should not be chosen as a member of the committee, based on its individual performance, and should be substituted with another member network that can add greater benefit.

Practically, one should compare the speed of computation, too, not only ultimate accuracy of the system. A relative difference in accuracy of a few

percent can hardly be noticed in practice, whereas tiny speed differences during actual operation are very visible. A single LVQ network with only thirty neurons in the hidden layer produces 92.5% accuracy while a three-member committee with a total of ninety neurons in the competitive layer increases the output to 95%. Very few applications would compromise such a large expense of resources for this slight gain in accuracy.

Other methods of discrimination between the member networks have also been proposed in this work. The number of epochs in the training phase may be varied to generate individual outputs by the member networks. Different neural network types may also be adopted for each member network e.g. all members need not be LVQ networks. MLP networks could also be used as members of the committee. Although LVQ networks have been proven superior for pattern classification, a committee with one or more MLP members may add information specific to certain speakers that are not extracted by LVQ members.

Another alternative of changing the order in which the training samples are presented to each network was attempted. In this case, the number of neurons in the competitive layer, S^1, was kept constant for all networks. This did not affect the recognition success rate per sample. The order in which the training samples are presented to the individual networks does not matter for this application.

4.10 Detection of Impostors to the Group

The speaker recognition may also be required to identify impostors to group rather than just classify the test input to the closest match. Traditional systems have utilized techniques such as passwords that need to be input into keypads that enable the system to accept the speaker as a valid member of the group. Only then are the vocal properties of the speaker used to distinguish which member of the group he/she is.

This study has proposed two alternative methods of impostor detection depending on the type of neural network used for pattern classification and are described below:.

i) If the minimum difference between the output value of a backpropagation network and the target speaker values exceeds some pre-defined threshold then the system would recognize the sample as not being from the group of n reference speakers but an impostor. The actual threshold value can be varied to adjust the sensitivity of the system.

ii) A similar technique can be used if LVQ networks are used instead. In this case the minimum value of the output of the competitive layer, vector a1 shown in figure 4.16, is scrutinized. If this value exceeds the preset threshold then the speaker is an impostor.

4.11 YOHO Database Test Results

The YOHO database available to the public is used to obtain results on an expanded dataset. The success rate using LPCs alone was compared to that using the LPC derived Cepstral coefficients for group sizes of 20 and 49. Eight training samples were used and an additional seven samples per speaker were used for verification.

Table 4.10 - YOHO neural network training specifications

Number of neural networks	1
Initial learning rate (lr)	0.001
me (maximum epochs)	15000
Number of hidden layer neurons	80
Number of speakers/subjects	20, 49
Training samples per subject	8
Test samples per subject	7
Total Test Samples	(20x7, 49x7)=140, 343

Table 4.11- Recognition success using the YOHO database

Feature Parameter	Group Size	Recognition Success
LPC	20	75.7%
LPC-derived Cepstrum	20	90.0%
LPC	49	61.5%
LPC-derived Cepstrum	49	73.8%

Spectral envelopes reconstructed from Cepstral coefficients are much smoother than ones reconstructed from LPC coefficients. It therefore provides a more stable representation from one repetition to another of a particular speaker's utterances [43]. This is evident from the table above. The results also reflect the effect of decreased recognition accuracy as the speaker group size is increased.

4.11.1 Committee neural network results using the YOHO database

Table 4.12 shows the results obtained using individual and committee neural LVQ networks on the YOHO speaker database.

Table 4.12 - Recognition success of individual LVQ member networks against a committee of neural networks (YOHO 49-speakers)

Member Network	S^1	Recognition Success
LVQ1	60	74.0%
LVQ2	80	73.8%
LVQ3	100	70.0%
Committee (3-member)		75.5%

The three-member committee of neural networks increases the performance of the individual LVQ networks at the expense of higher overhead. The performance of the system cannot be improved by merely increasing the size of the hidden layer. The committee consisted of a total of two hundred and forty neurons. An interesting comparison between the committee and a single LVQ network, comprising of two hundred and forty neurons in the hidden

layer, reveals a greater recognition success rate using the committee on this dataset. The single LVQ network had an accuracy of only 61.5% compared to 75.5% obtained with a committee with the same number of neurons.

4.12 Summary

The speaker recognition system has been implemented in MATLAB. This system was to tune the variables of the system prior to implementation of the hardware target system.

Various characteristic feature parameters were used to train the pattern recognition system. The best system performance was obtained from a hybrid vector of LPC-derived Cepstral coefficients. Results have yielded that LVQ neural networks are better suited to the pattern classification task of speaker recognition than MLP networks, which are excellent for function approximation.

The group-size of the system was extended to 20 speakers with the use of a committee neural network approach.

The choice of a committee neural network, comprising of LVQ members, and a hybrid vector of LPC-derived Cepstral coefficients for the hardware target system implementation have been based on the results obtained from this MATLAB system implementation.

An expanded dataset test was conducted using 49 speakers from the YOHO database. An increase in the group size impacted negatively on the recognition success rate. The tests also revealed that LPC-derived Cepstrums perform better than LPCs on their own.

The next chapter describes the hardware implementation of the speaker recognition system on a digital signal processor. Chapter Five is focussed on the hardware-based ANSI-C implementation of the speaker recognition

system on a DSP. This is the ultimate objective of this thesis. The main advantage of a hardware implementation over the MATLAB implementation is the ability to carry out real-time testing.

CHAPTER 5

HARDWARE IMPLEMENTATION OF THE SPEAKER RECOGNITION SYSTEM

5.1 Introduction

The main objective of this thesis is to implement a practical automatic speaker recognition system on a hardware platform. This chapter describes the Digital Signal Processor (DSP) implementation considerations for an automatic speaker recognition system. The characteristic feature parameters and neural network specifications chosen for the implementation of the recognition system is based on results obtained from the MATLAB implementation of the speaker recognition system described in Chapter Four. A supervised Learning Vector Quantization (LVQ) neural network is implemented as the pattern classifier. Linear Predictive Coding (LPC) and Cepstral signal processing techniques are utilized to form hybrid feature parameter vectors as inputs to the pattern recognition system. The practical implementation of a committee of neural networks, on a five-speaker group, for pattern recognition rather than the conventional single-network decision system is also implemented.

The following section describes the various functional blocks of the hardware implementation.

5.2 Overview of the Target System Hardware

A block diagram of the hardware system is shown in figure 5.1. The ADC64 is a 32-bit PCI-compatible card (5V signaling environment). Applications for the card include data acquisition and analysis, intelligent digital control systems,

acoustic processing, and co-processing for the PC. The ADC64 block in the figure shown below now accomplishes most of the relative functions of the speaker recognition system:

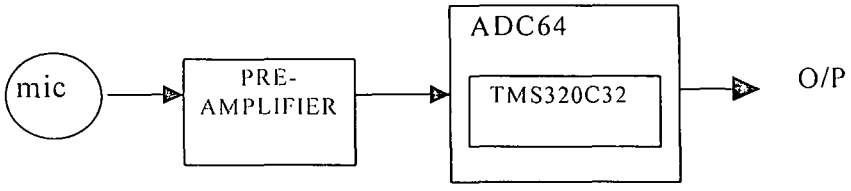


Figure 5.1 - Hardware implementation block diagram

Mic	– Conversion of sound waves to analog electrical signals
ADC64	– ADC conversion of the electrical signal to digital form
TMS320C32	– Feature extraction (LPCs and CCEPs)
TMS320C32	– Pattern templates and pattern matching (LVQ neural network implementation)

The characteristic feature extraction and pattern template and matching functions performed by the signal processing and neural network toolboxes in the MATLAB implementation are now implemented using ANSI C source code in the TMS320C3x environment.

The ADC64 data acquisition card, supplied by Innovative Integration, was chosen as the target system. The ADC64 is a high performance DSP-based (TMS320C32) computing engine. The main benefit of the system is its real-time performance capability. Using the ADC64, the host is freed up from time-critical events. This enables real-time performance, which is an afterthought on host PCs working under Windows and other operating systems.

The entire speaker recognition system is now developed on a single half-length PCI bus-compatible DSP data acquisition card incorporating a Texas Instrument's TMS320C32 floating point DSP processor. The code can now be embedded onto a stand-alone target system independent of the host platform. The ADC64 is used to acquire the training samples from each speaker. The

samples are then used to train the LVQ network off-line on the host PC using MATLAB to generate the final trained weights. A LVQ network is then initialized with these final trained weights on the DSP card. This network is now used to implement the training process on the ADC64. A block diagram of the ADC64 is shown in figure 5.2 below. The TMS320C32 processor core and peripherals represent the bulk of the onboard hardware implemented on the ADC64. The card is designed for a bank of 128Kx32 SRAM, 64 channels of instrumentation grade 16-bit analog I/O, 2 16-bit D/A channels, and 16 bits of bi-directional digital I/O. External interrupts are also pinned out to detect external events [35]. The analog I/O block is further expanded in figure 5.4.

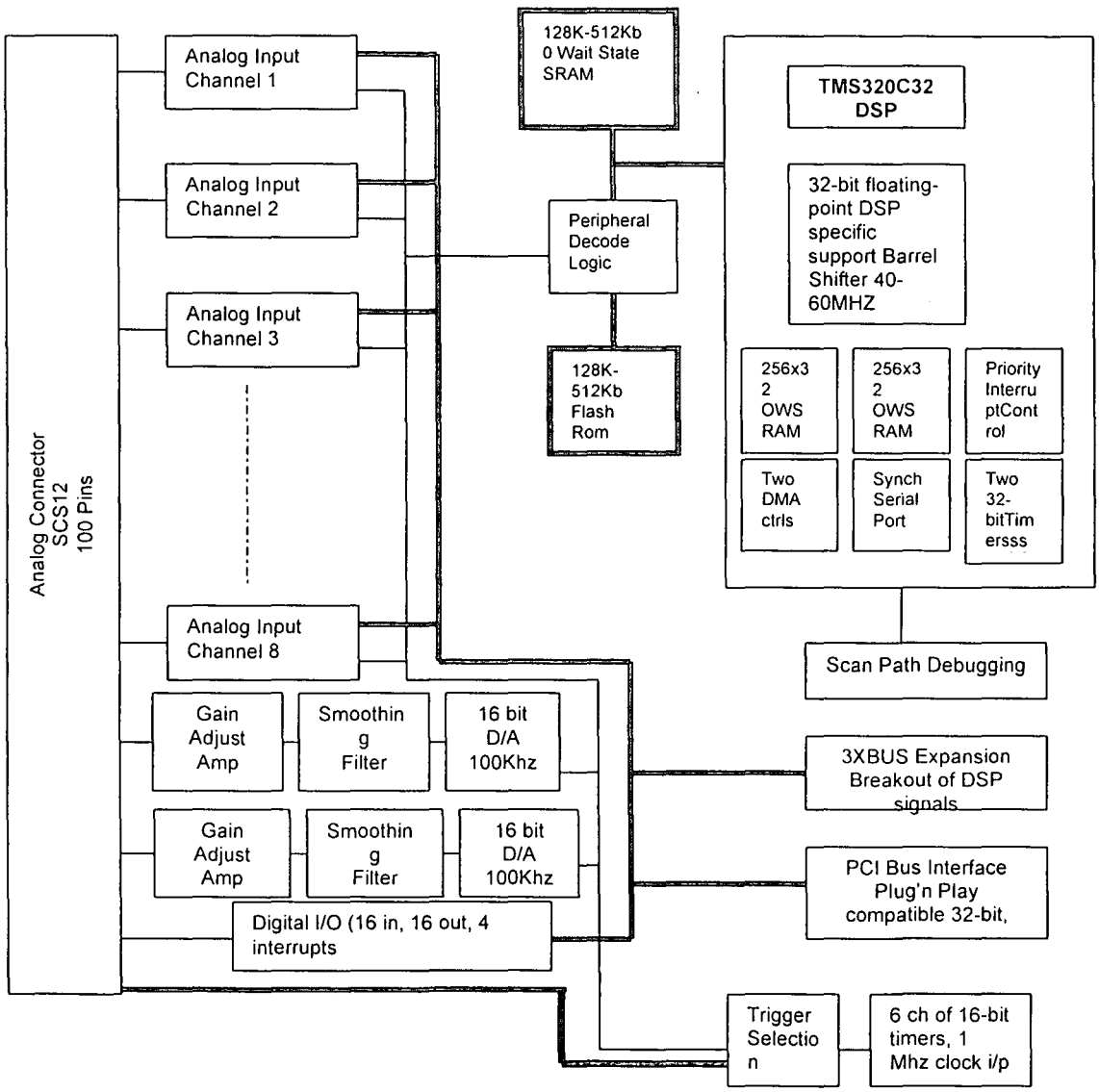


Figure 5.2 - ADC64 block diagram [35]

5.3 System Memory Map

The TMS320C32 processor is operated in bootloader (microcomputer) mode on the ADC64. The memory addresses generated by the TMS320C32 microprocessor are twenty-four bits wide resulting in a total 16 MWord address range. The memory map of the ADC64 is shown in Table 5.1. All variable formats (integers, float and double) are stored as 32-bit numbers. A TMS320C3x byte is 32 bits.

Table 5.1 - ADC64 memory map

Bootloader (Internal to DSP)	0x0 0xFFF
Flash ROM 128K Bytes (STRB0)	0x1000 0x1FFFF
Reserved	0x20000 0xFFFFF
Reserved Internal Peripherals	0x808000 0x8097FF
Reserved External Peripherals	0x810000 0x82FFFF
Internal SRAM 512x32	0x87FE00 0x87FFFF
Reserved	0x880000 0x8FFFFF
External SRAM 32-bit Physical	0x900000 0x91FFFF

The ADC64 directly supports up to 512 Kbytes of onboard zero-wait-state "program" SRAM (Static Random Access Memory). The onboard external SRAM (Static Random Access Memory) ranges from address 0x900000 to

0x91FFFF resulting in a total of 131071x32 bit locations (128 Kbytes of addressable memory). The ADC64 also supports a 512x32 bi-directional FIFO dual port memory block that may be accessed by either the ADC64 or the PC.

5.4 Software Development Environment of the ADC64

This section describes the software development environment for the ADC64 digital signal processor (DSP) card. The ADC64 environment is equipped with an ANSI C compliant Code Compiler, Assembler and Linker.

The C compiler supplied with the Developer's Package is the Texas Instruments (TI) floating-point C Compiler toolset for the TMS320C3x/4x family. The compiler runs under Windows as a cross compiler, generating executable applications for the DSP processor which is downloadable and executable using certain tools in the Developer's Package.

Typical application programs will consist of one or more C (.C), header (.H), and Assembly language (.ASM) source files, as needed. Additionally, target program generation requires use of a linker command file (.CMD) which specifies the memory map for the target and optionally includes commands defining the libraries to be linked into the final application.

5.4.1 The TMS320C3x/C4x optimizing C compiler

The TMS320C3x/C4x DSPs are fully supported by a complete set of code generations tools including an optimizing C compiler, an assembler, a linker, an archiver, a software simulator, a full-speed emulator, and a software development board.

Figure 5.3 illustrates the software development flowchart. Additional libraries provided with the Developer's System include C standard I/O and peripheral drivers for the A/D, D/A, bit-I/O and timers. The compiler accepts ANSI

standard C source code and produces assembly language source code for the TMS320C3x/4x devices. It was therefore not necessary to write the source code in assembly or machine language for this application of speaker recognition. The **'cl30'** shell program was used to automatically compile, assemble and link the source modules. Assembly language may also be used in conjunction with C code or the assembly language that the compiler produces may be modified where required. No part of this speaker recognition software was written using assembly language.

The assembler translates assembly language sources files to machine language object files. The machine language is based on common object file format (COFF).

The linker (**'lnk30'**) was used to combine object files into a single object module. The linker also allocates memory i.e. it defines the memory map. The 'lnk30' command invokes the TI linker to link several object modules to create a target executable (.OUT) file, consuming a linker command file (.CMD) as a parameter.

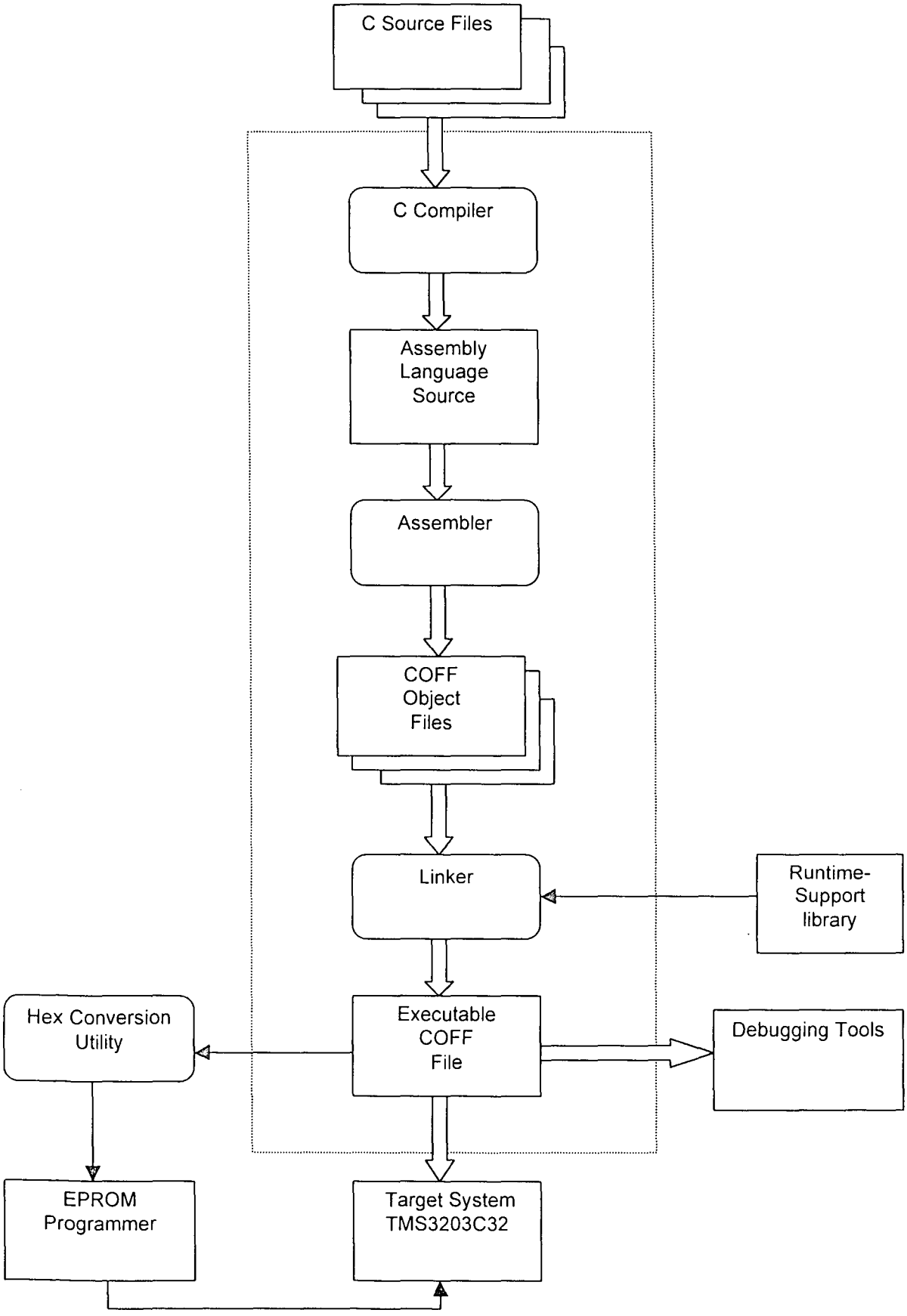


Figure 5.3 - Software development flowchart [38]

5.4.2 Memory models

The C compiler treats memory as a single linear block that is partitioned into sub-blocks of code and data. The linker defines the memory map and allocates code and data into target memory.

Two memory models are available on the TMS320C3x: the big memory model and the small memory model. The default small memory model was utilized in this application. This model requires that all external, global, static and compiler-generated constants fit into a single 64K-word long data page. This allows the compiler to access any of these objects without accessing the data page pointer (DP).

The big memory model removes the 64K restriction but it forces the compiler to reload the data page pointer before accessing certain variables and constants. The small memory model is used for the automatic speaker recognition implementation to avoid delays caused by the big memory model. The **block** attribute has been used in this application to make sure that the global and static variables memory section does not cross any 64K-page boundaries.

The TMS320C3x C optimizer analyses data flow to avoid memory accesses whenever possible. The volatile keyword is used for code that depends on memory accesses exactly as written in the C code.

5.5 System Analog Input Section

The analog to digital converters (A/D's) provides a means to digitize external analog signals and retrieve the resulting information for use in TMS320C32 software. Conversions can be triggered by either software requests (accesses to certain locations in the TMS320C32 memory map starts a conversion on

the A/D) or by hardware sources (on-board timers or external strobe sources cause conversions to begin). After a certain time period (the 5 μsec maximum conversion time of the A/D and front-end circuitry), the results of the conversion may be read using certain locations in the peripheral memory map [36].

There are eight 16-bit analog to digital converters (A/D) with a 5-microsecond maximum conversion time. Each A/D is a successive approximation type having an independent conversion path with independent filtering and input range.

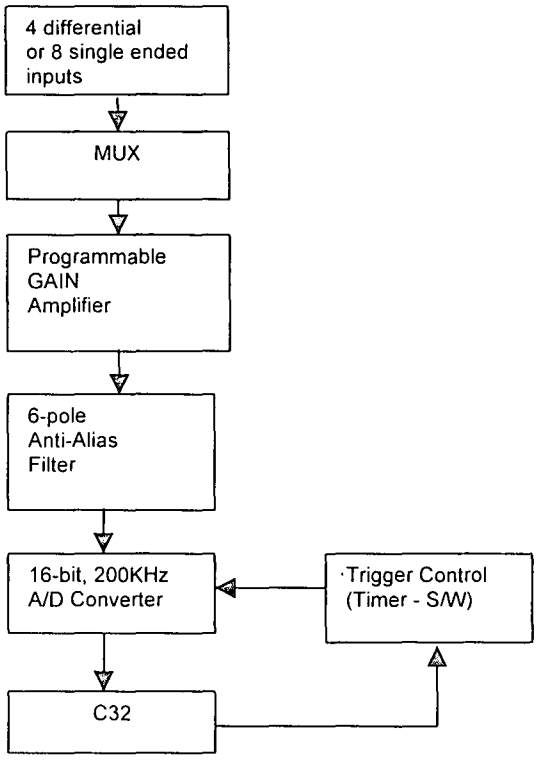


Figure 5.4 - Analog input system block diagram

The 16-bit analog input channels are configured with high input impedance amplifiers to permit direct connection to sensors and signal sources. The microphone of the system is connected to the ADC64 through the 100-pin SCS12 analog connector via a breakout terminal block and cable for the 100-pin connector. The ADC64 end connector P1 is a 100 pin female SCSI2 type

which provides access to all analog inputs, the D/A outputs, digital I/O, the external triggers, counter/timers and system power. The pin-outs of the connector are shown in Table G.1 of Appendix G.

The eight A/D's on the ADC64, may be triggered to sample simultaneously or independently as pairs. The default bipolar input range is ±10V. Custom input ranges may be achieved by changing the jumpers shown in Figure G.1 and Figure G.2 (in Appendix G.3). A simple block diagram of a single channel of the analog input system is given in Figure G.3 (in Appendix G.4).

5.5.1 System input sensitivity

The subject/speaker utters the training or test phrase into a 600Ω-impedance standard audio microphone whose output is approximately one millivolt. Table G.2 shows that the lowest selectable input range of the ADC64 is ±2.5V. The system sensitivity to the input is therefore very low, even when the input range is strapped to ±2.5V.

Each A/D has independent, software programmable gain. The standard gain selections are x1, x2, x4, or x8. The gain is selected by writing a number 0 to 3 to the programmable gain amps signifying the desired gain setting. This allows the gain and channel to be specified in one write cycle to the control logic. A gain of x2 was used in this application.

Although the analog input section of the ADC64 is equipped with this amplification circuitry, a very low Signal-to-Noise (SNR) is achieved due to the low level of the microphone output. It is therefore necessary for the signal to be amplified externally before being input to the analog circuitry of the ADC64. This was achieved using a standard audio pre-amplifier as shown in Figure 5.1. The circuit diagram of the pre-amplifier is shown in Figure G.4.

5.5.2 ADC64 sampling of the analog input

The A/D converters may be triggered for conversion by writes to the memory-mapped A/D, externally triggered by TTL signal, or triggered by the trigger selection matrix. It is preferred to trigger the conversion with the use of a timer, which results in the least jitter and timing ambiguity. This is especially important for audio applications. An on-board 10 MHz clock module clocks the timer channels. The 10 MHz timebase is very accurate and stable to 100 PPM since it is crystal driven [35]. Only ADC channel 0 was required for the sampling of each speaker's input speech sample, for both the training and test modes. The A/D convertors was triggered with the use of the (PIT0) on-board timer in this application of speaker recognition.

The input speech samples were limited to a time duration of 3 seconds. The sampling rate was chosen to be 16 KHz thereby satisfying the Nyquist criterion. As a result, the size of the speech signal is 48000 samples for one input training and test vector.

Digital processing requirements also need memory space for calculation of the LPCs (involves auto-correlation, Hermitian Toeplitz and matrix inversion) [7]. While the DSP acquisition card is excellent in terms of real-time computation one needs to ensure the memory requirements are met.

The ADC64 also has multiplexing capabilities, which were not necessary for this application. The software programmable gain is used to amplify the signal further. The ADC64 also provides independent 6-pole anti-alias filtering with offset for each A/D device.

After the analog peripherals are used to acquire the training samples from each speaker, the sampled input was then normalized to avoid amplitude variations of samples from the same speaker which will degrade the recognition performance of the system. A flowchart of the analog input and normalization process is shown in Figure 5.5 below. The actual C source code is listed in Appendix H.2.

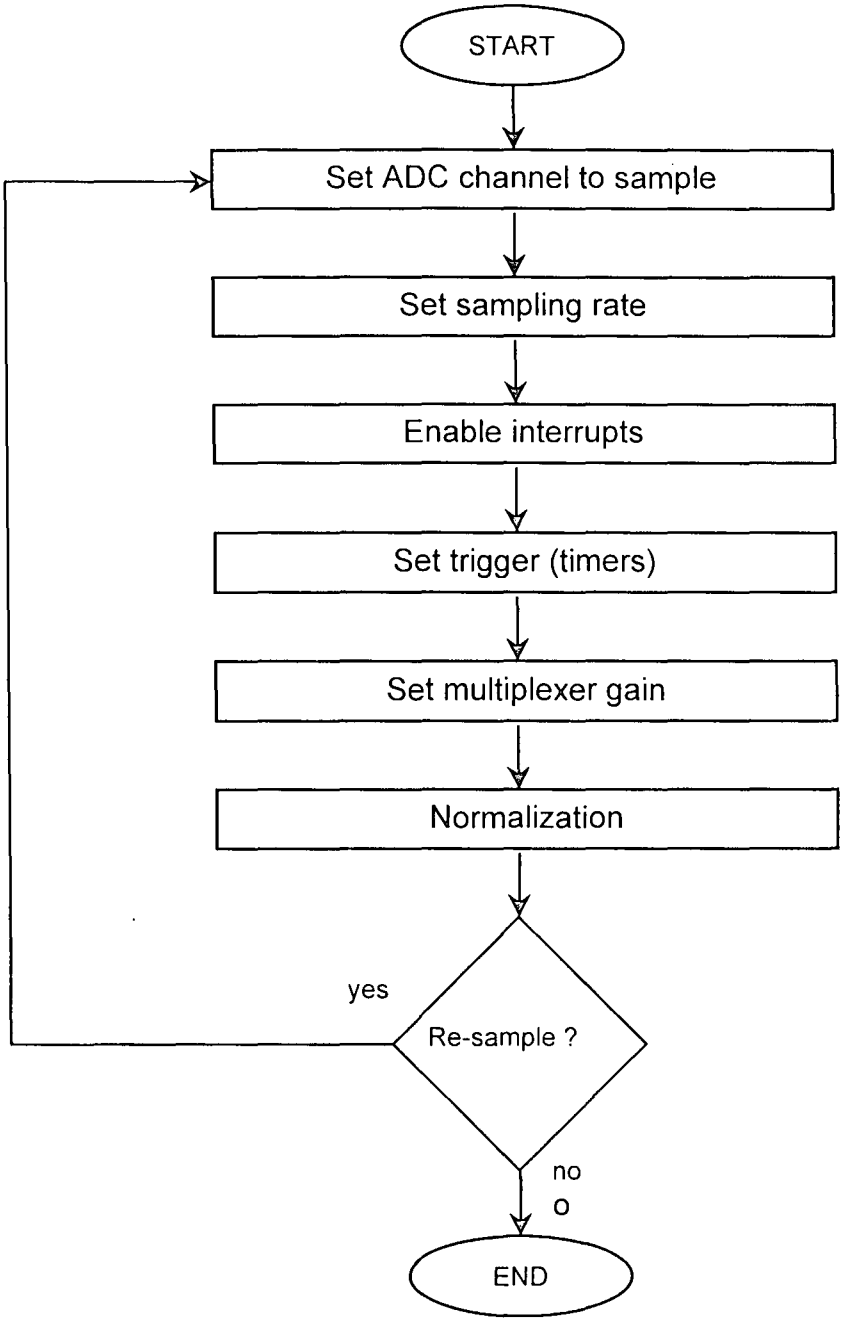


Figure 5.5 - ADC software flowchart

Figure 5.6 shows a normalized speech sample from speaker 'mum', using the target system, with a very low SNR. The high noise level is due to the low level of output from the microphone. The speaker also contributes to the SNR due to their actual volume of speech and placement of the microphone.

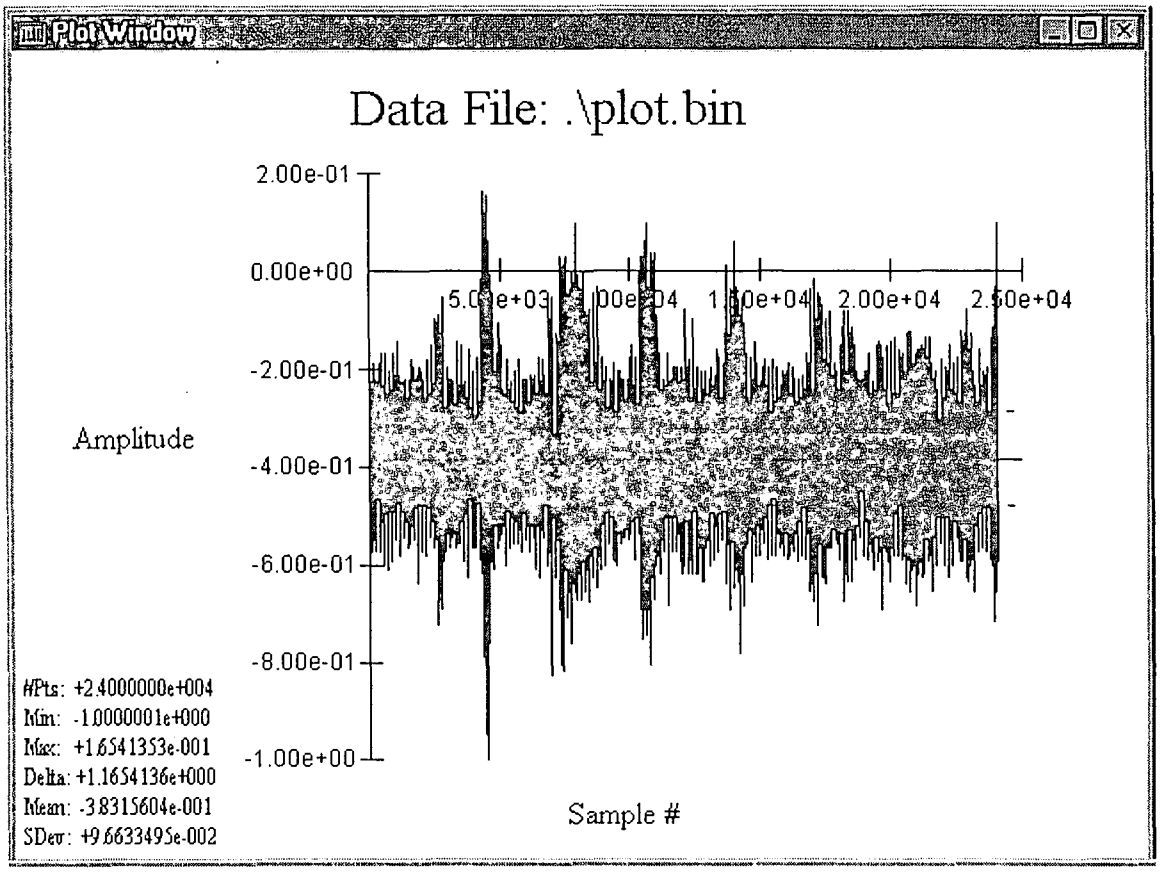


Figure 5.6 - Normalized noisy input speech of speaker 'mum1'

Figure 5.7 shows a slightly improved SNR as compared to figure 5.7. This could be due to either the positioning of the microphone or the loudness of the speaker.

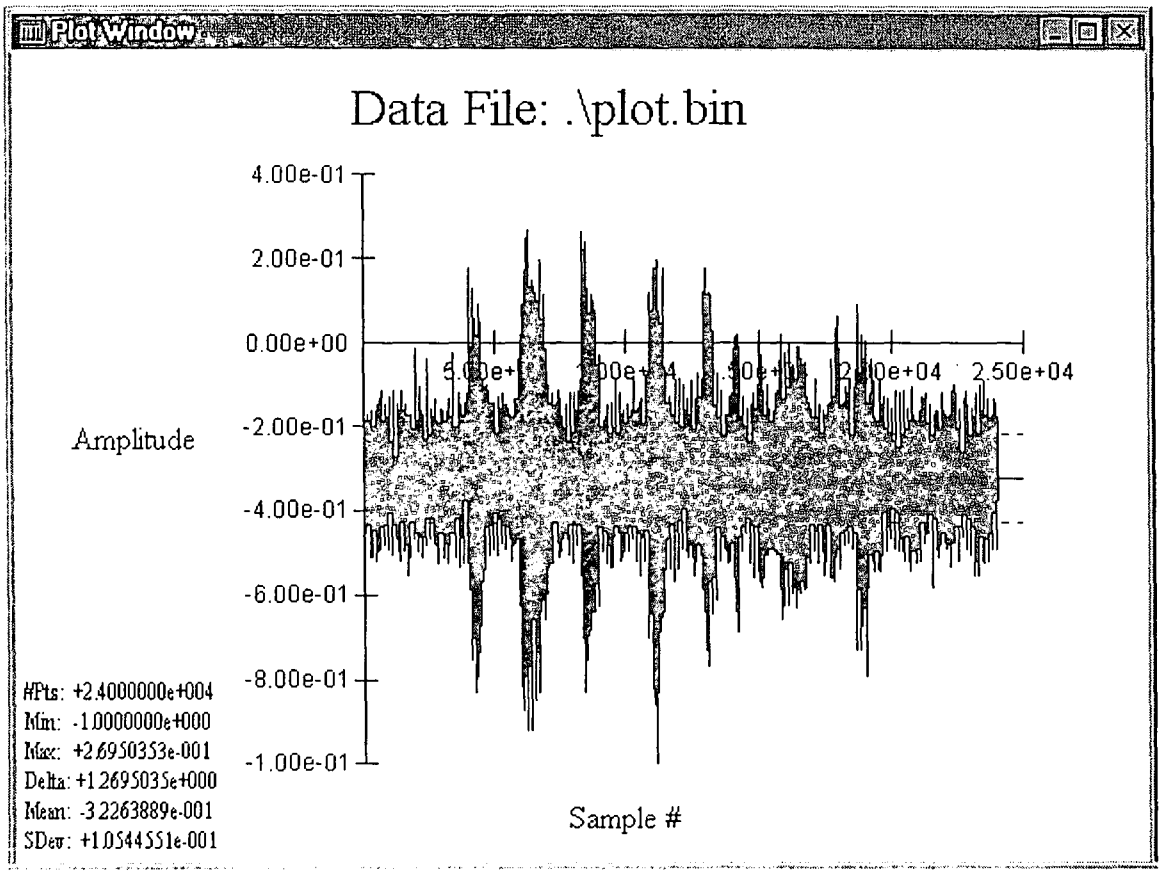


Figure 5.7 - Noisy normalized speech sample (speaker 'kas')

Figures 5.8 and 5.9 shows how the SNR is improved significantly with the use of a standard pre-amplifier prior to the signal being input to the analog section of the ADC64.

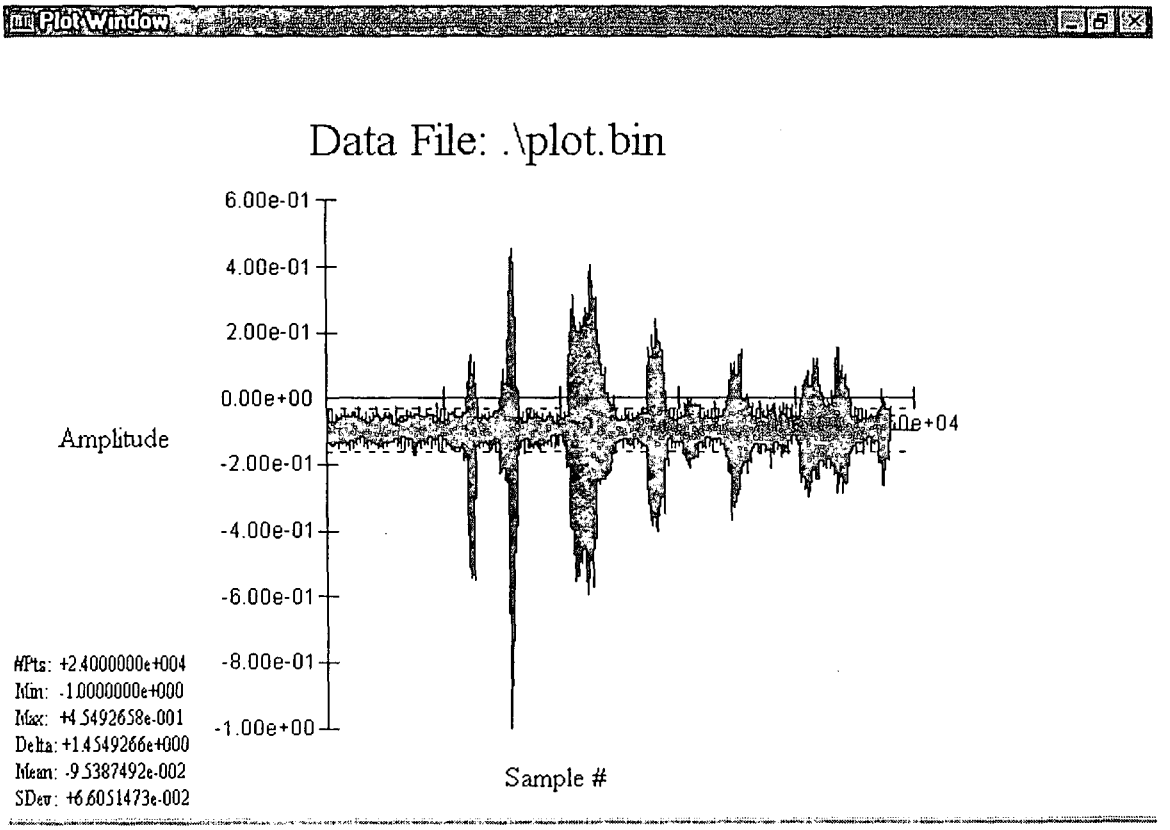


Figure 5.8 - Improved quality SNR speech sample from speaker 'dad'

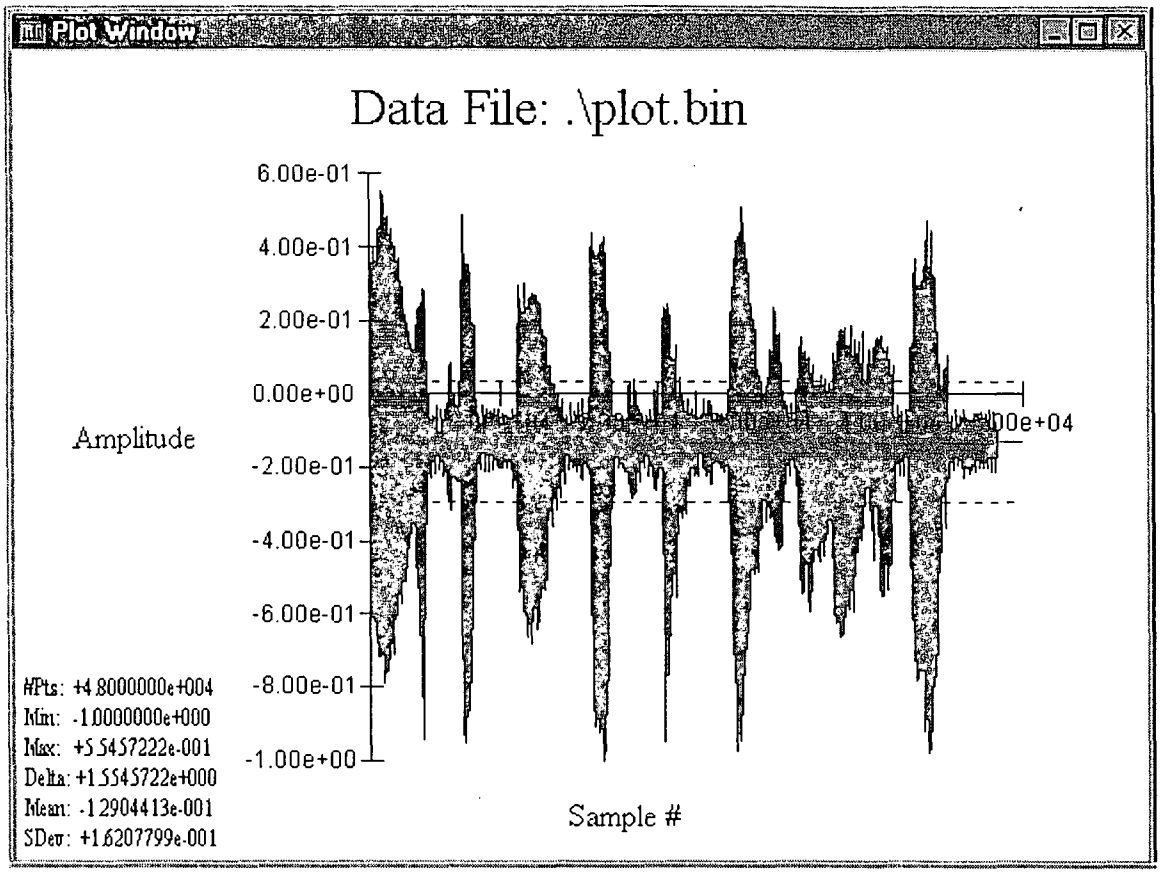


Figure 5.9 - Improved quality SNR speech from speaker 'vir'

5.6 Characteristic Feature Extraction using the ADC64

The next step, both during training and test modes of the system, is to extract the characteristic feature parameters of each speech sample, which would indicate the uniqueness of the speaker in terms of their voice signature. MATLAB contains functions that can be used to extract certain features e.g. Linear Prediction Coefficients (LPCs) and the complex Cepstrum (CCEPS). However, during the implementation on the target hardware all processes/functions need to be written from first principles using ANSI C source code. It is therefore necessary to understand how LPC and Cepstral coefficients can be derived from speech signals. Figure 5.10 and 5.11 shows the flowcharts of the method employed to calculate the LPC and Cepstral

coefficients respectively [25]. The first process in deriving LPCs is the autocorrelation of the recorded speech sample. The order of the LPC vector is determined by the maximum lag chosen for the autocorrelation process.

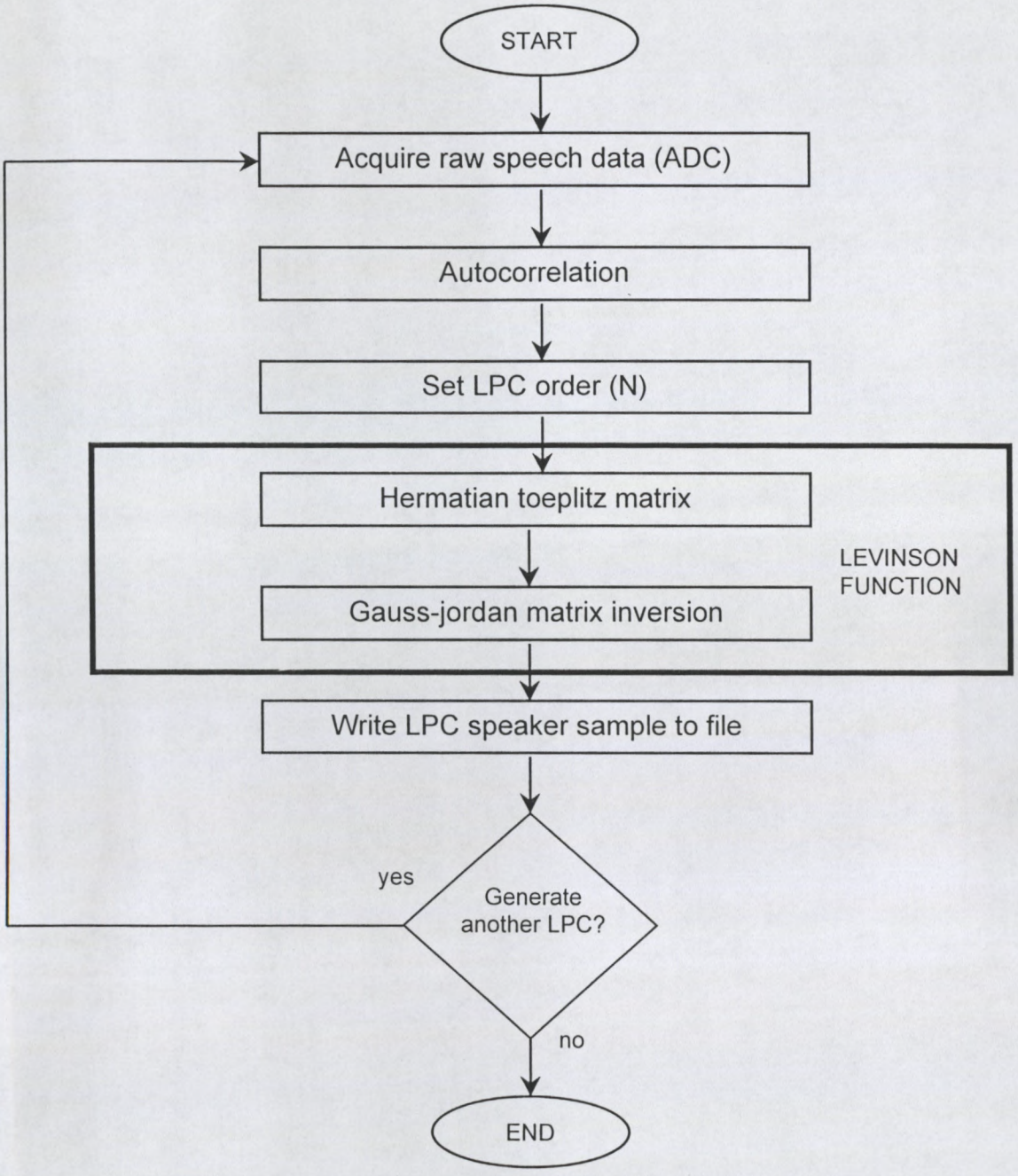


Figure 5.10 - Software flowchart for LPC coefficient derivation

Next, the Levinson function is applied by forming a Hermatian Toeplitz matrix and then finding its inverse using the Gauss-Jordan process of elimination. The LPC vector is then written to file for use during the training mode. Other

Speech samples may then be used to derive further LPC vectors if required. This is based on the condition of the loop block.

The source code described by the LPC flowchart shown above is listed in Appendix H.2. 32 LPC coefficients were chosen for this application due to memory constraints of target system. The Cepstral coefficients derived from these coefficients also constituted of 32 coefficients since the order of the Cepstral vector is dependent on the order of the LPC input vector.

The LPC coefficients may be used as the feature parameter to train and test the system as described in Chapter 4. In this case, the LPCs need to be written to file (host PC hard drive). The neural network to be implemented on the target system is trained off-line, using the MATLAB implementation, and the final trained weights are then loaded onto the target system for the test mode. This is necessary due to the memory constraints of the target system. There are only 128K of addressable SRAM locations. Each sampled speech segment alone requires 48000 locations to be stored. Memory is also required for the calculation of LPC coefficients and other temporary variables e.g. a duplicate input speech vector (further 48000 coefficients) for the autocorrelation process of LPC derivation. That is, more than 96000 (memory also required for the weights matrix of the committee of LVQ neural networks) of the available 128K of addressable SRAM locations are used for testing of a single speech sample. Memory would be required for 20x8 training speech samples if the training process were to be employed on the target system. Additional memory would now also be required for the calculation and adjustment of the neural network weight matrix during training.

The results of Chapter 4 have shown that the hybrid LPC-derived Cepstral coefficients provide much improved performance than the LPC coefficient feature vectors. Cepstral vectors were therefore derived from the LPC coefficients as shown in Figure 5.12.

The Cepstral coefficients were calculated by taking the inverse Fourier transform of the natural logarithm of the Fourier transform. The C source code is listed in Appendix H.2.

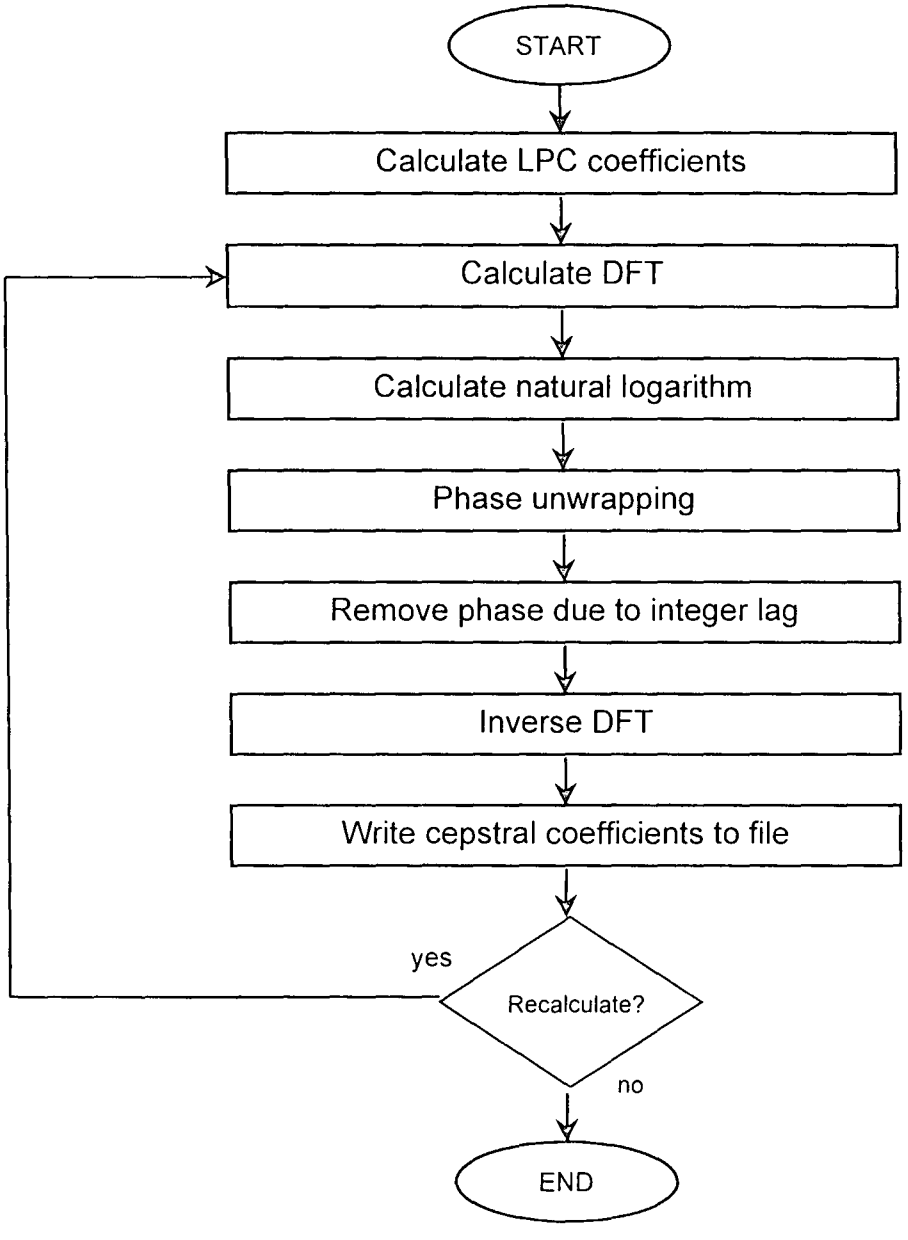


Figure 5.11 - Software flowchart of the Cepstral coefficient derivation

Figures 5.12 and 5.13 show plots of LPC coefficients that were generated by the target system. The LPC calculation stage was an intermediate stage of the

feature extraction process since the eventual parameters used were CCEP coefficients.

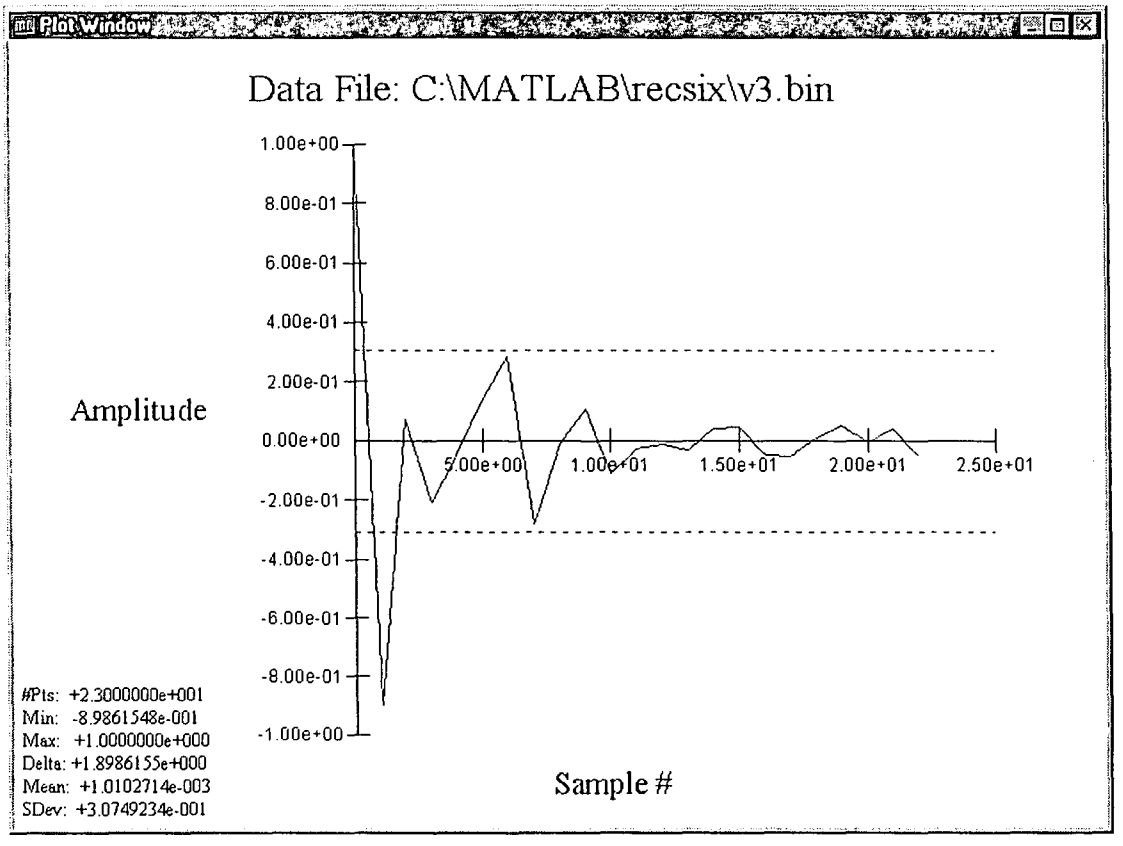


Figure 5.12 - LPC coefficients derived by the target system (speaker 'vir')

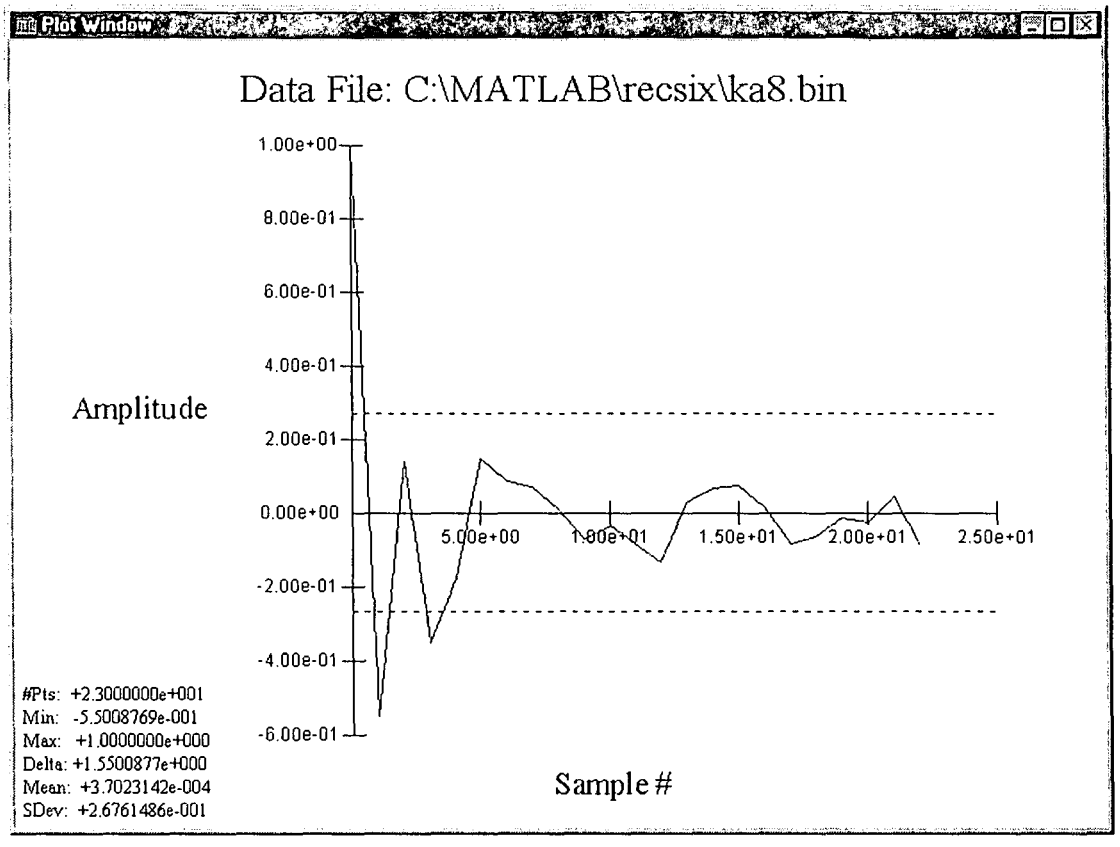


Figure 5.13 - LPC coefficients derived by the target system (speaker 'kas')

Figures 5.14 and 5.15 show plots of 32-point Cepstral coefficient feature vectors by two different speakers. These 32-point Cepstral coefficients are used as the feature parameters to train the neural network pattern classifier for implementation on the TMS320C32 DSP board.

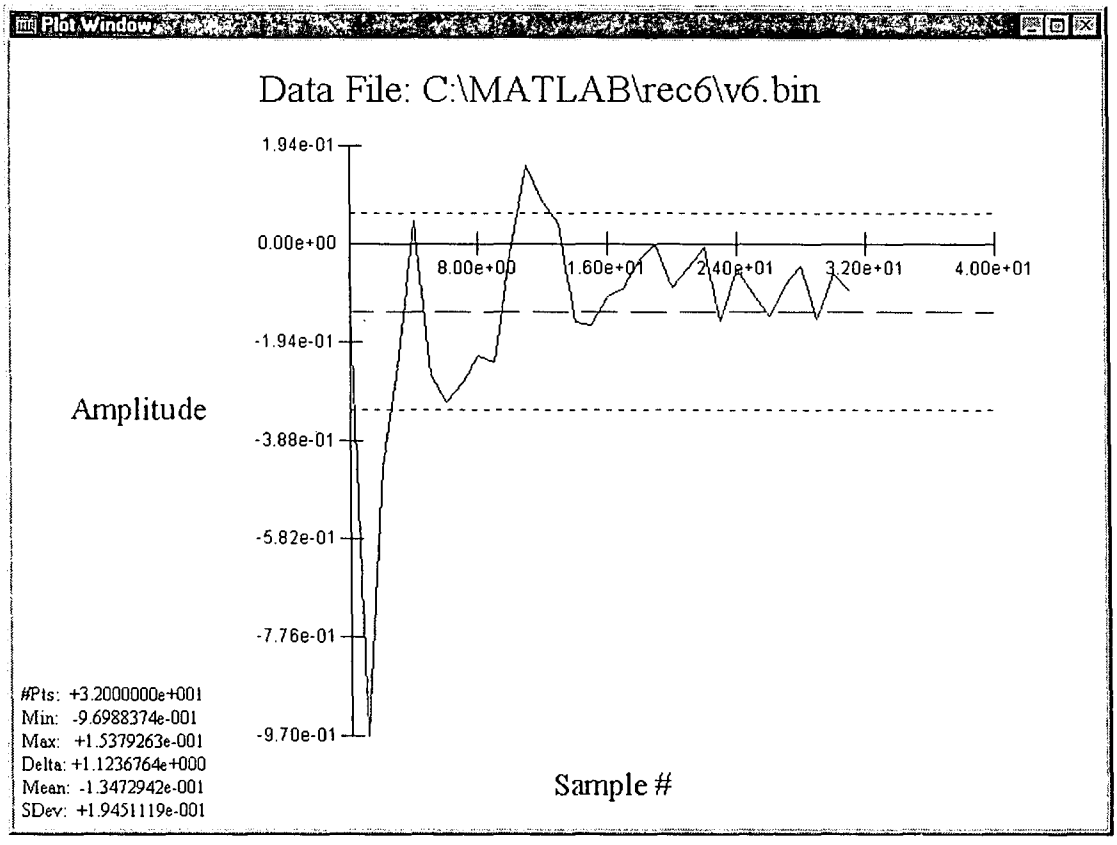


Figure 5.14 – Cepstral coefficients for speaker 'vir' computed on the DSP board

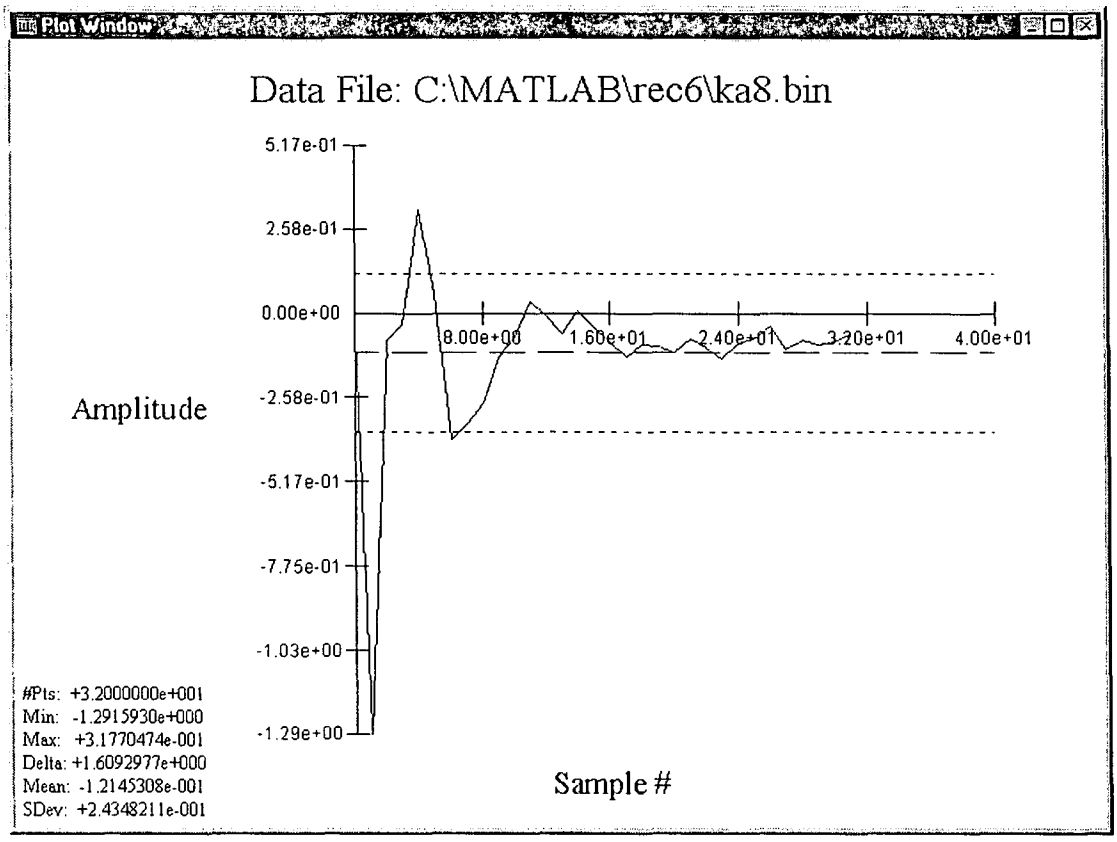


Figure 5.15 – *Cepstral coefficients for speaker 'kas' computed on the DSP board*

5.7 File Formats

Since the system is trained off-line, using the MATLAB implementation on a host PC and not the DSP board, and the final weights are then re-loaded onto the target system, it is necessary to have variables written in formats that are compatible to both the host and target systems. This is necessary if data is being passed to/from host software, since the 80x86/87 floating point units usually used in PC host systems use the IEEE-754 format.

When writing data from the target system to the host PC it is necessary to convert IEEE-754 format floating point number values into TMS320C3x/4x floating point format, returning it as the result. This is required when writing the LPCs or CCEPS derived using the target ADC64 system to the host PC hard drive for training of the neural network pattern recognition systems.

Channel variances, as described in Chapter 2, would be introduced if the CCEP parameters were calculated by the host system during the training phase and derived by the target system during the test phase. It is therefore necessary to have the target system process the speech samples and derive the Cepstral coefficients rather than the host system. The only function of the host PC is to train the neural network for implementation on the target platform. The SRAM constraints of the target don't permit the on-board training of the LVQ network. The test phase is now entirely implemented on the TMS320C32 kit in real-time as described below.

The opposite process is required when initializing the neural network on the target system with the trained weights generated by the host PC, since the 80x86/87 floating point units usually used in ISA host systems use the IEEE-754 format.

5.8 Neural Network Implementation on the Target System

Supervised Learning Vector Quantization (LVQ) neural networks are used since they are excellent pattern classifiers. The practical application of a committee of neural networks for pattern recognition rather than the conventional single-network decision system has been implemented in this work

Feature parameters, acquired using the target system to avoid channel variances, are read from the host PC files and used to train the system off-line (using MATLAB on the host PC).

The flowchart, shown in Figure 5.16, describes the training process to determine the finalized weights for the target system.

Appendix H.3 lists the source code used for the training mode. This code is also used to train each of the member networks during the committee implementation. The only difference being the S1 variable (number of neurons in the hidden layer). Each member of the committee will be initialized with a different number of neurons in the LVQ hidden layer.

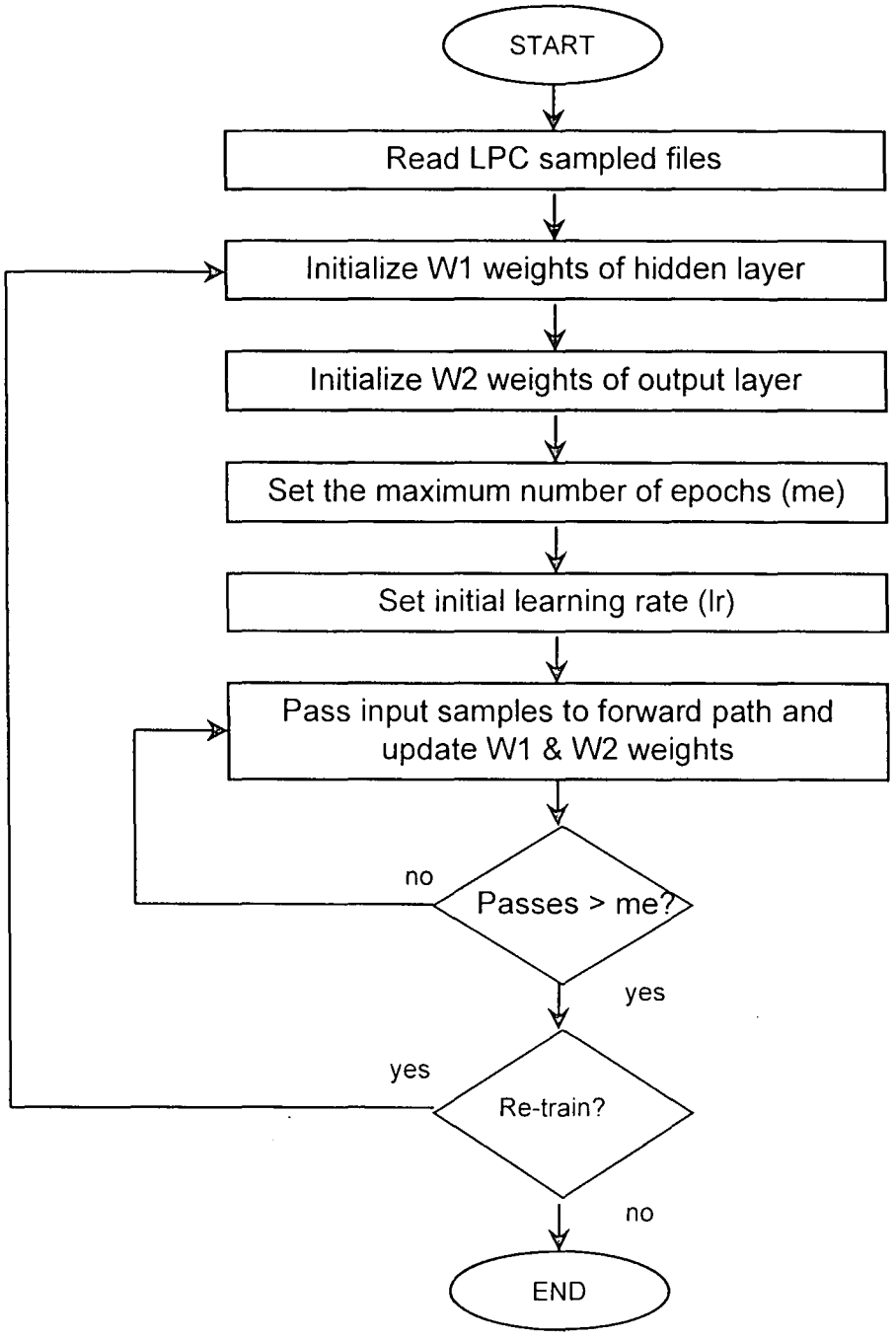


Figure 5.16 - Flowchart of system training process

Figure 5.17 shows the process followed in the test mode after the speaker recognition system is trained.

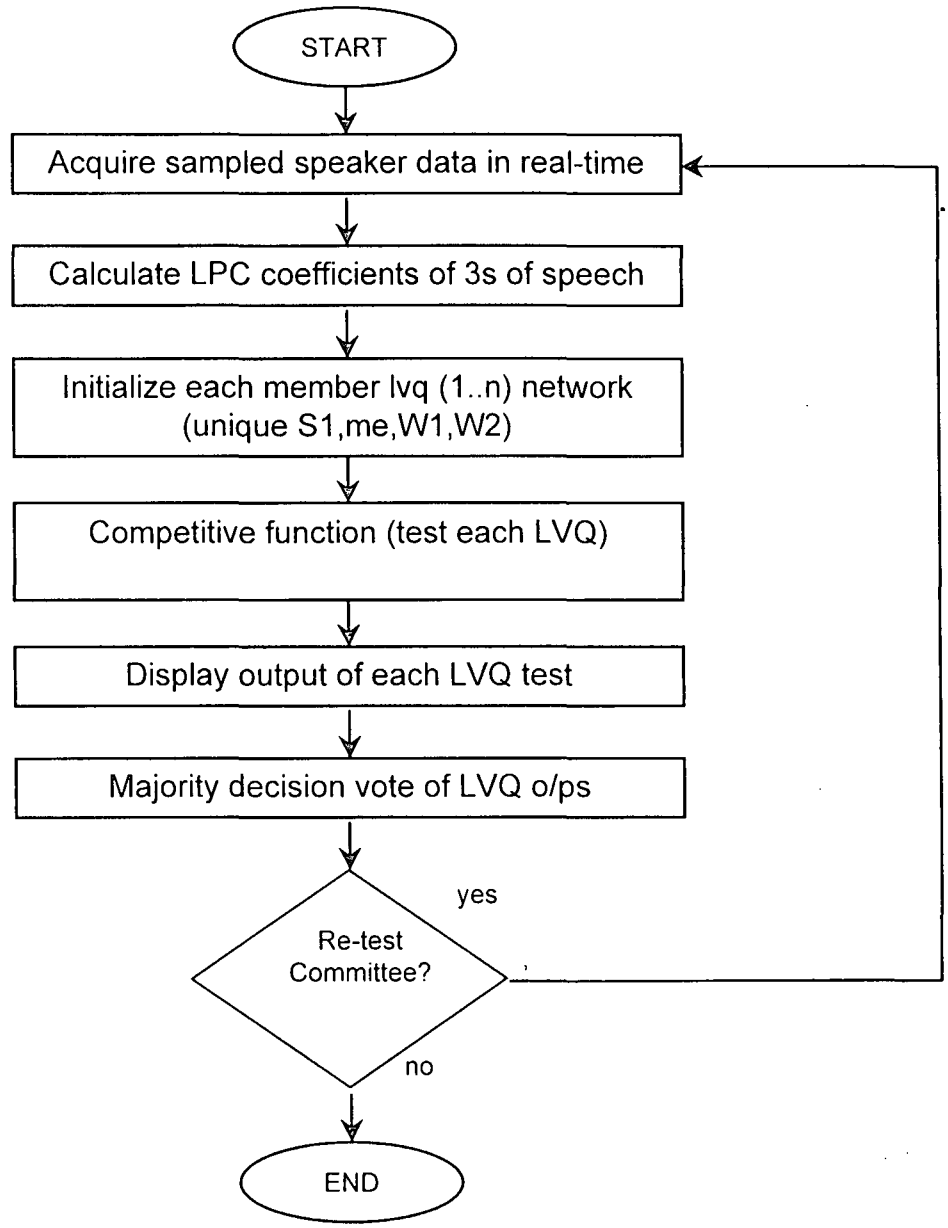


Figure 5.17 - Flowchart of target system test mode

5.9 System Front-End

Figures 5.18 and 5.19 show the 2-stage compiling process and the linking process of the system software. This is accomplished with the use of the TMS320C3x C compiler (cl30) and TMS320C3x linker (lnk30).

Figure 5.20 shows that the code was compiled with no errors. The complete software program is developed as a series of individual functions. The main program then calls each function of this modular program in order to maximize the use of the limited 128K word SRAM memory.

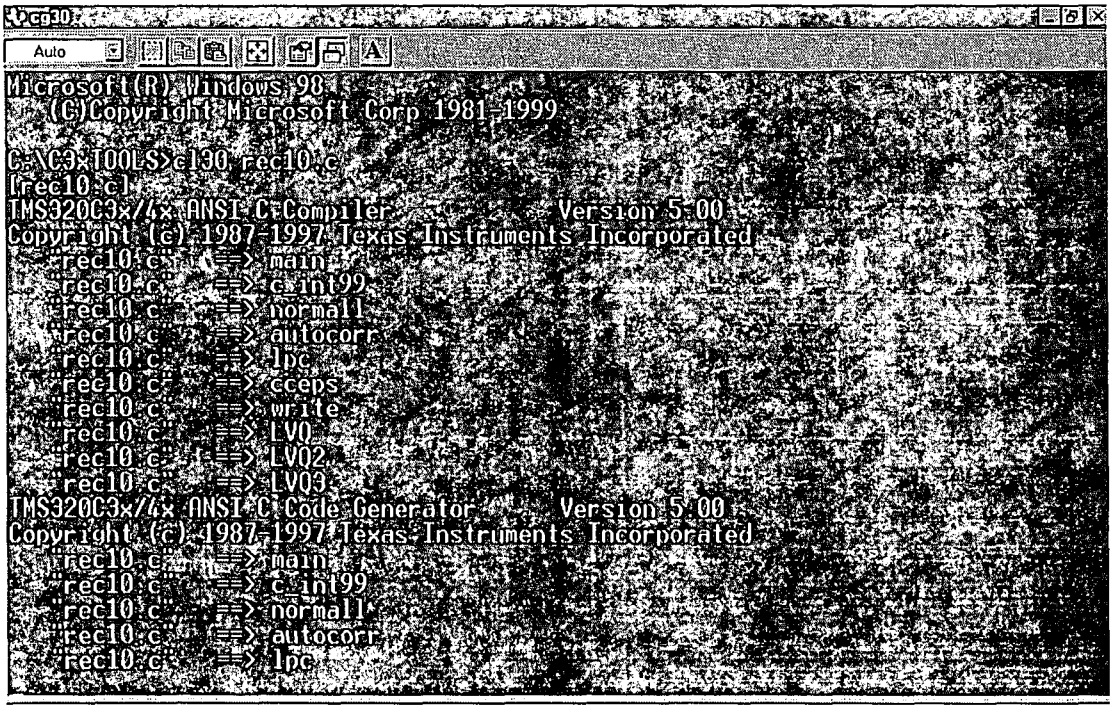


Figure 5.18 - Compiling the source code

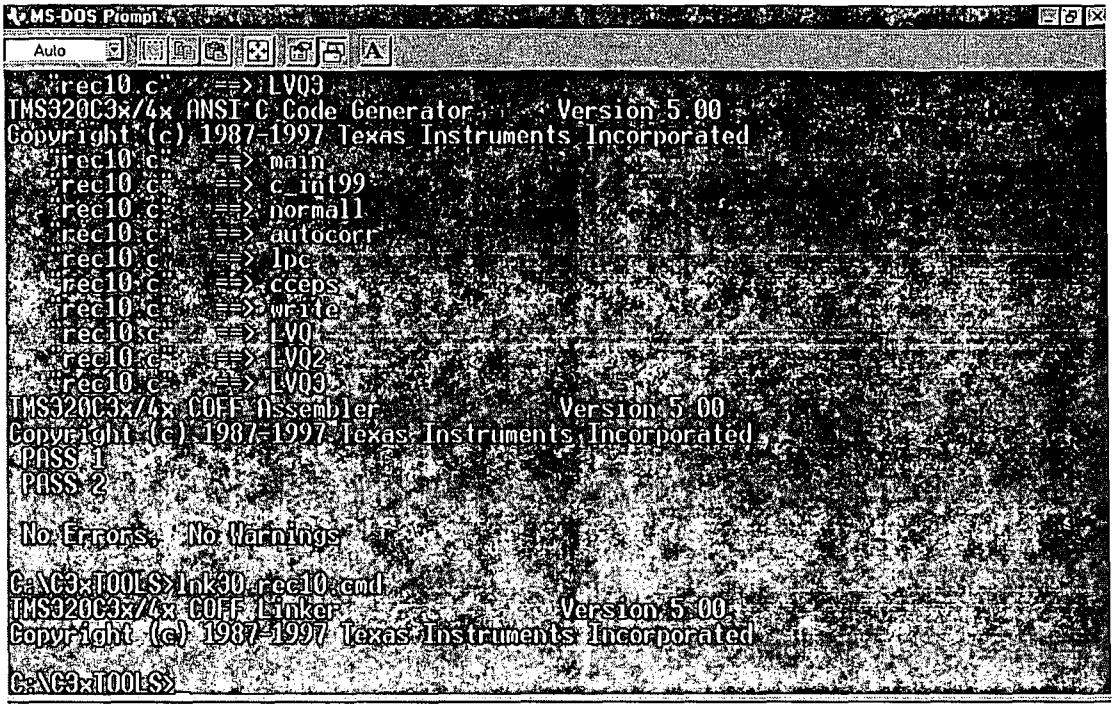


Figure 5.19 - Linking of the object file

The linking process dictates the memory layout of target system by controlling the memory allocation. Appendix I.2 shows the .CMD file consumed by the linker to generate the .OUT COFF file to be executed on the target system. Appendix I.3 shows the effect of the linking process on the memory distribution.

Figure 5.20 shows the target system prompt the speaker to strike a key on the keyboard and proceed to record a speech sample.

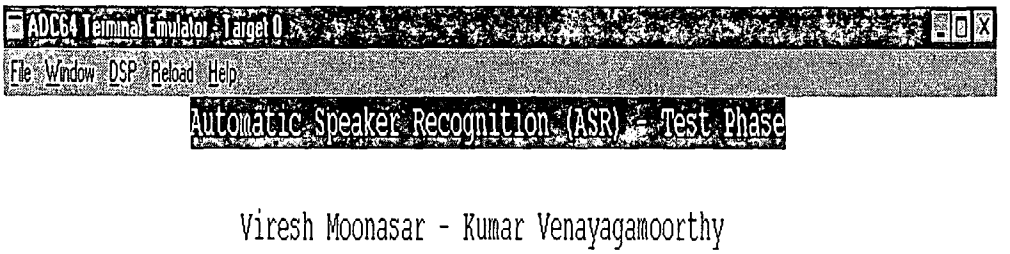


Figure 5.20 - Front-end of the system training mode

Figure 5.21 shows the output of the speaker recognition system test mode. All the three members of the LVQ committee network classify an input voice sample as speaker 4 of the group. The output of the competitive layer of each member network is also displayed. This value is an indication of the "confidence" of the system. A competitive function matches any input vector to the closest neuron, even if there is a huge Euclidean distance between the input vector and the closest matching neuron. It may also be used as a means to detect impostors if the value exceeds some predefined threshold.

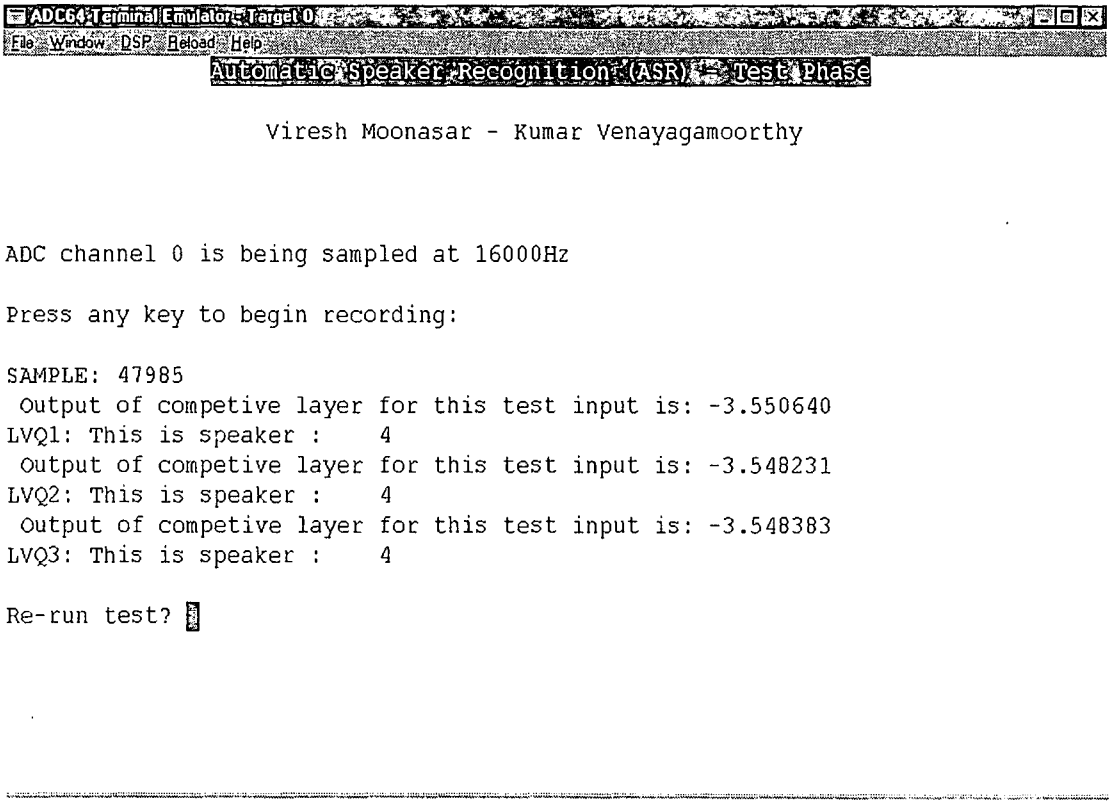


Figure 5.21 - Front-end of the test mode of the system

5.10 DSP-Target System Test Results

Table 5.2 shows the parameters used for the hardware target system implementation. Test results were obtained, using the DSP-target system, from a 5-subject group. The LVQ network was trained with ten training samples per speaker that were processed using the target system described above. This is necessary to avoid channel characteristic variances during the training and testing phases. Channel characteristic variance is a key factor that negatively affects the performance of automatic speaker recognition systems.

Table 5.2 - Target System Committee Network Parameters

lr (learning rate)	0.001
me (maximum epochs)	15000
Feature parameter	LPC-CCEP hybrid
Number of member networks for committee	3
S1 (neurons in the hidden layer)	LVQ1=10 LVQ2=15 LVQ3=20
Number of speakers/subjects	5
Training samples per subject	10
Test samples per subject	10
Total Test Samples	50

Figure 5.22 reflects the test results of a single LVQ network (LVQ2 with S1=15). The feature parameter vector comprised of 32-element LPC coefficients. The target system successfully recognizes 88% (44 of the 50 test samples were recognized successfully) of the test inputs. The low recognition success rate for speaker five can be attributed to the very low SNR of the

speakers samples used for the training and test modes of the system. Other contributing factors include the incorrect placement of the microphone.

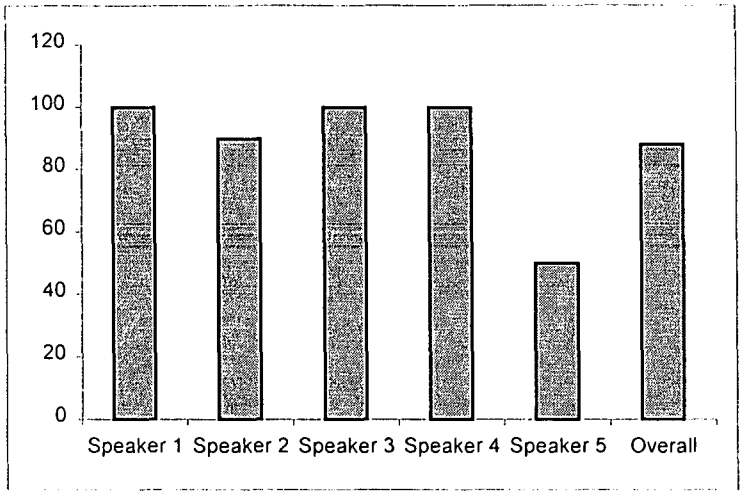


Figure 5.22 – Hardware results (LVQ2 with LPC input)

Figure 5.23 illustrates an increase in output performance from 88% to 92 % when LPC-CCEP hybrid feature inputs are used instead (LVQ2). Further a committee of three member LVQ networks increases the performance of the single LVQ2 network system to 94%.

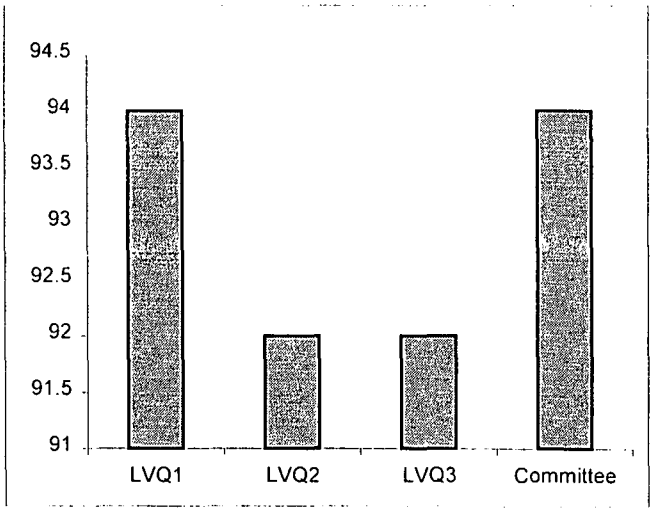


Figure 5.23 - Committee neural network results

Practically, one should compare the speed of computation, too, not only ultimate accuracies. A relative difference in accuracy of a few percent can hardly be noticed in practice (depending on application), whereas tiny speed differences during actual operation are very visible. A single LVQ network with only fifteen neurons in the hidden layer produces 92% accuracy while a three-member committee with a total of forty-five neurons in the competitive layer produces an output of 94%. Very few applications would compromise such a large expense of resources for slight gain in accuracy.

Figure 5.23 also shows that LVQ1 produces an output of 94%, which is equivalent to the output of the committee of neural networks. Although, the choice of ten S1 neurons is optimal for this particular set of input speaker training and test samples it may not be optimal when other test samples, from the same speakers, are used. The committee can add the benefit of obtaining the recognition success rate close to the best member network. After all, it is a majority vote system.

Committees of neural network can enhance the performance of certain individual LVQ neural networks, but at a cost. Processing time is increased and there is a much larger memory requirement. Target systems do have memory limitations e.g. 128Kbytes of external RAM in our system. Memory is also required for the storage of the actual speech samples and other program variables, constants and program code.

5.11 Summary

The automatic speaker recognition system has been implemented successfully on a simple target system with some degree of accuracy. Similar results were obtained during implementations on the host PC. Further improvements will result from the improvement of the Signal to Noise Ratio (SNR).

The SNR of the input to the system also affects the performance of the system greatly. The type and placement of the microphone is crucial. Another solution is to use an array of microphones to achieve greater SNR. The SNR can be improved even further with the use of a pre-amplifier, which increases the input signal level to the ADC64 target system.

A committee of neural networks has been implemented to enhance the performance of the system, but at a cost. Processing time is increased and there is a much larger memory requirement. Target systems do have memory limitations e.g. 128K words of external SRAM in this system.

The 94% recognition rate achieved for a five speaker group, with the three-member committee network, using the target system correlates well with the results achieved with the MATLAB implementation. This implies that favourable results may be achieved with the target system when the group size is extended to 20 members.

CHAPTER 6

CONCLUSION

6.1 Introduction

This thesis has investigated the practical considerations when implementing an automatic speaker recognition system using single and committee LVQ neural networks on a real-time target system that is independent of any host Personal Computer (PC). Software has been developed for automatic speaker recognition in MATLAB and in real-time on a DSP-based hardware system.

This chapter is divided into three parts. The first part presents the major conclusions and results from each of the previous chapters. The second part summarizes the major findings of this thesis in the application of ANNs to automatic speaker recognition. The third and final part suggests some of the many possible areas of further research in the field of automatic speaker recognition.

6.2 Chapter Summaries

Chapter One describes the application of automatic speaker recognition, the layout, objectives and main contributions of this thesis.

Natural Language Processing, which includes speaker recognition, is discussed in Chapter Two. The fundamental concepts of human speech production, the human auditory system and implementation of an automated machine-based system are described. Also included are topics such as characteristic feature extraction, representing the uniqueness of a speakers voice signature, and factors that hinder the performance of speaker recognition systems.

Chapter Three discusses the theory and application of ANNs to speaker recognition. The suitability of the LVQ neural network for pattern recognition over backpropagation neural networks, which are better suited to function approximation, is stressed in this chapter. This chapter also introduces the concept of committee neural networks and their value in pattern recognition.

The MATLAB implementation of the speaker recognition system is described in Chapter Four. This chapter proves that using the concept of using a hybrid feature vector comprising a combination of characteristic feature vectors does improve the performance of the recognition system substantially. Committee of neural networks are shown to improve the accuracy of the system even further but at a high computational resource cost.

Chapter Five describes the practical implementation and measured results of a DSP-based real-time system. The results generated show that speaker recognition systems can be implemented on stand-alone, PC independent hardware systems.

6.3 Main Conclusions

The final conclusions to be drawn from this investigation are that:

i) The hybrid LPC-derived Cepstral coefficients are good characteristic feature representation of the speakers' voice.

ii) LVQ neural networks are excellent for pattern recognition.

iii) The use of the committee of neural network does enhance the performance of the system.

iv) Finally, the entire speaker recognition system, including the committee approach, can be practically implemented on a simple target system with reasonable accuracy.

6.4 Suggestions for Further Research

This thesis has been the first step in the practical implementation of an automatic speaker recognition system. Although the hardware implementation has generated reasonable practical success there are several areas that could be investigated and improved.

Specific areas for future research, which follow on from the work of this thesis, are listed below:

a) Implementation of all three modes of the speaker recognition system on the hardware system alone.

b) Comparison of the ANN based system with a probabilistic based system such as the HMM or GMM-based systems. Versus resource expense should also be made.

c) Expand the group size of the system even further (greater than twenty subjects) whilst still maintaining or improving the accuracy achieved by this system.

d) Investigate the use of a hybrid template-based and probabilistic-based system e.g. HNNs as proposed in chapter 3.

e) Investigate the use of other characteristic feature parameters e.g. the use of the time-dependent Fourier transform (Spectrograms).

6.5 Summary

This chapter has summarized the major results and conclusions obtained in the previous chapters and suggest areas in which further research could be undertaken.

The remainder of this thesis consists of appendices and references referred to in the previous chapters.

APPENDIX A

SOUND FILE FORMAT

A.1 Introduction

The file format used to store the recorded speech samples from the various speakers, for both the MATLAB implementation training and test modes of the speaker recognition system is described in this section. Other sound file formats are also discussed.

A.2 File Formats

The following sound file formats are supported by the "Cooledit" sound recording software which, was used during the MATLAB implementation of the speaker recognition system:

A.2.1. WAV - Windows PCM waveform

Consists of Pulse Code Modulation (PCM) waveform data, which is just the exact amplitude of each sample.

A.2.2. WAV - Microsoft ADPCM waveform

This compressed waveform data consists of 4-bit per channel compressed data. Each sample is expanded when loaded. This compression results in distortion. Distortion is more intense at frequencies greater than 2 kHz.

A.2.3. WAV - IMA ADPCM waveform

This standard compresses 16-bit waves to 4-bit using a different method than Microsoft ADPCM. Distortion is less in the high end compared to Microsoft ADPCM, but is spread out evenly through the entire frequency range. Slightly lower quality.

A.2.4. WAV - CCITT mu-law and A-law waveforms

Compresses 16-bit to 8-bit audio. Quality is somewhere between 8 and 16-bits. They thus have a higher signal to noise ratio (s/n) than ADPCM formats.

A.2.5. AIF - Apple AIFF format

This is Apple's standard wave format. We are not using a Apple platform and thus did not make use of this format.

A.2.6. AU - Next/Sun CCITT mu-Law, A-Law and PCM format

Similar to WAV - CCITT mu-Law and A-Law. Most common use for the AU file format is compression of 16-bit to 8-bit mu-law data.

A.2.7. PCM - Raw PCM data

This is simply the dump of the PCM waveform data without any header information. Sample rate, resolution and number of channels will have to be input when loading.

A.2.8. MPG - MPEG Layer 2

The MPEG (Motion Picture Expert Group) format, layer 2, will allow the saving of CD quality (16-bit 44.1 kHz stereo) with no noticeable loss in quality. High compression ratios, like 10:1 are attainable with little loss in fidelity [25].

APPENDIX B

MATLAB IMPLEMENTATION MEMORY CONSTRAINTS

B.1 Introduction

This section describes the memory limitations of the MATLAB implementation (PC) system that constrained the total number of training samples that could be presented to the pattern recognition neural network.

B.2 System Memory Constraints

If the duration of each speech sample averaged 3 seconds then the memory required to store each sample is:

Memory required (M) = Sampling rate (F_s)×Resolution (R)×duration of speech (t)

$$M = F_s \times R \times t \qquad \text{(B.2.1)}$$

If resolution of 16 bits/sample is used then using equation (B.2.1):

$$M = 16\,000 \text{ samples/second} \times 16 \text{ bits/sample} \times 3 \text{ seconds}$$

M = 768 kbits

Since 1 byte = 8 bits,

M = 96 kbytes

This is not large to store on the hard drive of a computer. The problem arises when these samples have to be processed. The MATLAB digital signal processing software was used and during training modes all the training samples are needed to be loaded into the Random Access Memory (RAM) of the PC. Furthermore, additional RAM may be required by the software for intermediate processing steps.

The computer on which the study has been carried out had 40 Mbytes of RAM.

Maximum number of training samples that can be processed = 40 M / 0.096 M

$\approx$ 416

If 10 training samples per speaker are used to train the network and a further sample is needed per speaker to test the network then the maximum number of speakers allowed to train this network is :

Maximum number of speakers per network = 416 / 11

$\approx$ 37

In practice the maximum number of speakers permissible is considerably less than this since additional memory is needed to store (pattern recognition) network parameters (weights and biases).

APPENDIX C

CHARACTERISTIC FEATURE PARAMETER PROGRAM CODE

C.1 Introduction

The feature parameters used during the MATLAB implementation of the speaker recognition system were derived using the MATLAB Signal Processing Toolbox. This appendix lists the code for the different feature parameters used during the MATLAB implementation.

C.2. Fourier Transforms

The Discrete Fourier Transforms (DFT) of the acquired voice samples that was calculated and plotted in MATLAB using the FFT "built-in" function. The MATLAB code used to generate the FFT is given below :

```
x=auread('ku1.au') ;                    % reads stored speech sample
x=fft(x,2048) ;                         % calculate  2048 pt. FFT
x=abs(x) ;                              % magnitude of FFT
plot(2,1,1);                            %subplot 1
plot(x);                                % plot FFT
axis([0 2500 0 4]);                     % scaling of axes
title('2048 Point FFT of Speaker 1');   % labeling of axes
ylabel('Magnitude');
y=auread('sa1.au');
```

```
y=fft(y,2048);
y=abs(y);
subplot(2,1,2);
plot(y);
axis([0 2500 0 4]);
title('2048 Point FFT of Speaker 2');
ylabel('Magnitude');
xlabel('FFT points');
```

C.3. Power Spectral Density estimate code

The function, Pxx = PSD (X, NFFT, Fs, WINDOW), was used to generate all PSD feature features during the MATLAB implementation of the system. The Power Spectral Density of signal vector X is estimated using Welch's averaged periodogram method. X is divided into overlapping sections, each of which is detrended, then windowed by the WINDOW parameter, then zero-padded to length NFFT. The magnitude-squared of the length NFFT DFTs of the sections are averaged to form Pxx. Pxx is length NFFT/2+1 for NFFT even, (NFFT+1)/2 for NFFT odd, or NFFT if the signal X is complex. If you specify a scalar for WINDOW, a Hanning window of that length is used. Fs is the sampling frequency which doesn't effect the spectrum estimate but is used for scaling of plots.

[Pxx,F] = PSD(X,NFFT,Fs,WINDOW,NOVERLAP) returns a vector of frequencies the same size as Pxx at which the PSD is estimated, and overlapsthe sections of X by NOVERLAP samples.

The following syntax has been used for the PSD function [32]:

% compute PSD:

```
x = x(:);                                   % Make sure x is a column vector
window = window(:);
n = length(x);                              % Number of data points
nwind = length(window);                     % length of window
if n < nwind                                % zero-pad x if it has length less
                                                  than the window length
 x(nwind)=0;  n=nwind;
end
k = fix((n-noverlap)/(nwind-noverlap));     % Number of windows
                                            % (k = fix(n/nwind) for noverlap=0)
if 0
   disp(sprintf('  x       = (length %g)',length(x)))
   disp(sprintf('  y       = (length %g)',length(y)))
   disp(sprintf('  nfft    = %g',nfft))
   disp(sprintf('  Fs      = %g',Fs))
   disp(sprintf('  window  = (length %g)',length(window)))
   disp(sprintf('  noverlap = %g',noverlap))
   if ~isempty(p)
      disp(sprintf('  p       = %g',p))
   else
      disp('  p       = undefined')
   end
   disp(sprintf('  dflag   = "%s"',dflag))
   disp('  --------')
   disp(sprintf('  k       = %g',k))
end

index = 1:nwind;
KMU = k*norm(window)^2; % Normalizing scale factor ==> asymptotically
unbiased
```

```
% KMU = k*sum(window)^2;% alt. Nrmlzng scale factor ==> peaks are about
right

Spec = zeros(nfft,1); Spec2 = zeros(nfft,1);
for i=1:k
   if strcmp(dflag,'linear')
      xw = window.*detrend(x(index));
   elseif strcmp(dflag,'none')
      xw = window.*(x(index));
   else
      xw = window.*detrend(x(index),0);
   end
   index = index + (nwind - noverlap);
   Xx = abs(fft(xw,nfft)).^2;
   Spec = Spec + Xx;
   Spec2 = Spec2 + abs(Xx).^2;
end

                                                    % Select first half
if ~any(any(imag(x)~=0)),                           % if x is not complex
   if rem(nfft,2),                                  % nfft odd
      select = (1:(nfft+1)/2)';
   else
      select = (1:nfft/2+1)';
   end
   Spec = Spec(select);
   Spec2 = Spec2(select);

else
   select = (1:nfft)';
end
```

```
Spec = Spec*(1/KMU);
freq_vector = (select - 1)*Fs/nfft;

% find confidence interval if needed
if (nargout == 3)|((nargout == 0)&~isempty(p)),
   if isempty(p),
      p = .95;    % default
   end
   confid = zeros(size(Spec));
   if k > 1
      c = (k.*Spec2-abs(Spec).^2)./(k-1);
      c = max(c,zeros(size(Spec)));
      confid = sqrt(c);
   end
   ff = sqrt(2)*erfinv(p);  % Equal-tails.
   confid = (ff.*confid)*(1/KMU);

   % compute variance using Welch's equation
   % (assumes underlying Gaussian process)

   if noverlap > 0
      disp('Warning: confidence intervals inaccurate for NOVERLAP > 0.')
   end
end

% set up output parameters
if (nargout == 3),
   Pxx = Spec;
   Pxxc = confid;
   f = freq_vector;
elseif (nargout == 2),
```

```
  Pxx = Spec;
  Pxxc = freq_vector;
elseif (nargout == 1),
  Pxx = Spec;
elseif (nargout == 0),
  if ~isempty(p),
     P = [Spec max(Spec-confid,0) Spec+confid];
  else
     P = Spec;
  end
  newplot;
  plot(freq_vector,10*log10(abs(P))), grid
  xlabel('Frequency'), ylabel('Power Spectrum Magnitude (dB)');
end
```

The PSD plots in figures 4.5 and 4.6 were then produced using the following code:

```
sat1=auread('sat1.au');                    % read au file from disk
[Psat1,F]=PSD(sat1,1024,16000,hanni        % calculate PSD, 1024 pt , sampling
ng(256),128);                              rate=16kHz, hanning
subplot(2,1,1)                             % window(size),overlap
plot(F,10*log(Psat1));                     % plot PSD
ylabel('Pow. Spec. Mag. (dB)');
title('PSD of sat1');

sat2=auread('sat2.au');
[Psat2,F] = PSD(sat2,                      % calculate psd
1024,16000,hanning(256), 128);
subplot(2,1,2)
plot(F,10*log(Psat2));
```

```
ylabel('Pow. Spec. Mag. (dB)');
xlabel('Frequency (Hz)');
%axis([0 90 0 0.7]);
title('PSD of sat2');
figure
sat3=auread('sat3.au');
[Psat3,F] = PSD(sat3,                    % calculate psd
1024,16000,hanning(256), 128);
subplot(2,1,1)
plot(F,10*log(Psat3));
%axis([0 90 0 0.7]);
title('PSD of sat3');
ylabel('Power Spec. Mag. (dB)');
kum1=auread('kum1.au');
[Pkum1,F] = PSD(kum1,                    % calculate psd
1024,16000,hanning(256), 128);
subplot(2,1,2)
plot(F,10*log(Pkum1));
%axis([0 90 0 0.7]);
title('PSD of kum1');
ylabel('Power Spec. Mag. (dB)');
xlabel('Frequency (Hz)');
```

C.4. Linear Predictive Coefficient Code

A = LPC(X,N) finds the coefficients of an Nth order auto-regressive process that models the time series X as follows:

$$X(n) = -A(2)*X(n-1) - A(3)*X(n-2) - \ldots - A(N+1)*X(n-N-1)$$

Here, X is the real input time series (a vector), and N is the order of denominator polynomial A(z), i.e. A = [1 A(2) ... A(N+1)]. If you leave N unspecified, LPC uses a default N = LENGTH(X)-1. LPC uses the autocorrelation method, also known as the maximum entropy method (MEM) of spectral estimation.

```
error(nargchk(1,2,nargin))
if nargin<2, N = length(h)-1; end

if (N>length(h)-1),        % disp('Warning: zero-padding short input sequence')
h(N+1)=0;
end
R = xcorr(h);
M=length(h);
R(1:M-1) = [];
a = levinson(R,N);
```

LPC calls the LEVINSON function:

A = LEVINSON(R,N) solves a symmetric toeplitz system of equations using the Levinson-Durbin recursion. R is a vector of autocorrelation coefficients, starting with lag 0 as the first element. N is the order of the recursion; A will be a length N+1 row, with A(1) = 1.

The equations solved are of the form:

```
[ R(1)   R(2) ... R(N)   ]   [ A(2)   ]     = [ -R(2)   ]
[ R(2)   R(1) ... R(N-1) ]   [ A(3)   ]     = [ -R(3)   ]
[ .      .        .      ]   [ .      ]     = [ .       ]
[ R(N-1) R(N-2) ... R(2) ]   [ A(N)   ]     = [ -R(N)   ]
[ R(N)   R(N-1) ... R(1) ]   [ A(N+1) ]     = [ -R(N+1) ]
```

If N is not large, LEVINSON will use the \ (Backslash or left division) function to solve this system, which is faster than the Levinson-Durbin recursion because of its higher overhead:

A\B is the matrix division of A into B, which is roughly the same as INV(A)*B , except it is computed in a different way. If A is an N-by-N matrix and B is a column vector with N components, or a matrix with several such columns, then X = A\B is the solution to the equation A*X = B computed by Gaussian elimination. A warning message is printed if A is badly scaled or nearly singular. A\EYE(SIZE(A)) produces the inverse of A.

R is the auto correlation vector, R(1) = E(h(t)h*(t)),
R(2) = E(h(t+1)h*(t)),...

Code for the Levinson function:

```
error(nargchk(1,2,nargin))
if nargin < 2, N = length(R)-1; end
if length(R)<(N+1), error('Correlation vector too short.'), end

if (N>195),          % use higher overhead, but asymptotically faster algorithm
a = -R(2)/R(1);
V = R(1) - R(2)^2/R(1);
for n = 1:N-1,
alfa = [1 a.']*R(n+2:-1:2);
rho = -alfa / V;
V = V + rho*alfa;
a = [ a + rho*flipud(a); rho ];
end
a = [1; a].';
```

```
else   % use the good old \ command
      b=-R(2:N+1);
      a = [ 1;  toeplitz(R(1:N))\(b(:)) ]';
   end
```

The Levinson function now calls the Toeplitz function. TOEPLITZ(C) is a symmetric (or Hermitian) Toeplitz matrix. The code for toeplitz follows:

```
if nargin < 2,
  c(1) = conj(c(1)); r = c; c = conj(c);          % set up for Hermitian Toeplitz
else
  if r(1) ~= c(1)
      disp(' ')
      disp('Column wins diagonal conflict.')
  end
end
r = r(:);                                         % force column structure
p = length(r);
m = length(c);
x = [r(p:-1:2) ; c(:)];                           % build vector of user data
cidx = (0:m-1)';
ridx = p:-1:1;
t = cidx(:,ones(p,1)) + ridx(ones(m,1),:);        % Toeplitz subscripts
t(:) = x(t);                                      % actual data
```

The LPC function was then used during the MATLAB implementation to calculate the 30-point LPCs of the input speech signals shown in figure 4.8:

```
suz1=auread('suz1.au');                           % read file from disk
kas1=auread('kas1.au');
a=lpc(suz1,30);                                   % calculate 30 LPC coefficients
```

```
b=lpc(kas1,30);
subplot(2,1,1);
plot(a);                                    % plot coefficients
title('LPC coefficients of suz1');
ylabel('Magnitude.');
subplot(2,1,2);
plot(b);
ylabel('Magnitude.');
xlabel('Coefficient number');
title('LPC coefficients of kas1');
```

C.5 Cepstral Coefficient Code

The complex cepstrum is defined as the IFFT of the natural logarithm of the FFT of the original signal:

```
function xhat = cceps(x)                    % calling function
h = fft(x);                                 % the FFT of the original
                                               signal(h)
logh = log(abs(h))+sqrt(-1)*rcunwrap(angle(h));   %log of the magnitude and
                                                    unwrap phase
xhat = real(ifft(logh));                    % real part of the IFFT
```

The cceps function calls the rcunwrap function. Function y = rcunwrap(x). Phase unwrap utility used by CCEPS. RCUNWRAP(X) unwraps the phase and removes phase corresponding to integer lag:

```
n = max(size(x));
y = unwrap(x);
```

```
nh = fix((n+1)/2);                              %round towards zero
y(:) = y(:)' - pi*round(y(nh+1)/pi)*(0:(n-1))/nh;
```

The rcunwrap function call the unwrap function. Function q = unwrap(p, cutoff). Unwrap phase angle in radians. UNWRAP(P) unwraps radian phases P by changing absolute jumps greater than pi to their 2*pi complement. It unwraps columnwise with matrices. UNWRAP(P,TOL) uses a jump tolerance of TOL rather than the default TOL = pi:

```
if nargin < 2, cutoff = pi; end                 % Original UNWRAP used pi*170/180.

[m, n] = size(p); oldm = m;
if m == 1, p = p(:); [m, n] = size(p); end      % Column orientation.

pmin = min(p); pmin = pmin(ones(m, 1), :);  % To force REM to behave.
p = rem(p - pmin, 2 .* pi) + pmin;              % Phases modulo 2*pi.

b = [p(1, :); diff(p)];                         % Differentiate phases.
c = -(b > cutoff); d = (b < -cutoff);           % Locations of jumps.
e = (c + d) .* 2 .* pi;                         % Array of 2*pi jumps.
f = cumsum(e);                                  % Integrate to get corrections.

q = p + f;                                      % Phases + corrections.
if oldm == 1, q = q.'; end                      % Reorient.
```

APPENDIX D

PSD CROSS-CORRELATIONS

D.1 Introduction

Cross-correlation was used as an initial method to evaluate how closely the PSDs of different samples of the same speaker's voice are related to each other. This section describes the cross-correlation results and the code used. All cross correlation results were computed at zero lag between the two signals.

Table D.1- Cross-correlation results of the same speaker

Speaker 1	Speaker 2	Cross Correlation Result
vir1	vir1	1.0000
vir1	vir2	0.9871
vir1	vir3	0.9605
vir1	vir4	0.9605
vir2	vir3	0.9821
kas1	kas1	1.0000
kas1	kas2	0.9746
kas1	kas3	0.9736
kas1	kas4	0.9739
sat1	sat1	1.0000
sat1	sat2	0.9904
sat1	sat3	0.9901
sat1	sat4	0.9853
suz1	suz1	1.0000
suz1	suz2	0.9895
suz1	suz3	0.9839
suz1	suz4	0.9866

Table D.1 reveals that there is very close correlation between PSDs of the same speaker. Now the samples of different speakers are calculated to check how significant the differences between them are. Table D.2. shows these results.

Table D.2 - Cross-correlation results of different speakers

Speaker 1	Speaker 2	Cross Correlation Result
sat5	kum3	0.6448
suz1	vir3	0.5764
suz2	kum4	0.7324
suz3	sat4	0.6058
vir1	sat1	0.9812
vir1	sat3	0.9605
vir1	kum1	0.9660
vir1	suz1	0.9875
vir2	sat2	0.8401
vir3	sat3	0.7569

The results in Table D.2 show very high levels of correlation between some samples of different speakers. Therefore cross-correlation cannot be used as a method of recognition with a high degree of success. The PSDs are instead fed as inputs to a neural network classifier.

D.2 Cross-Correlation Code

```
x=auread('suz3.au');                          % read file from disk
[Pxx,F] = PSD(x, 1024,16000,hanning(256),     % compute PSD
128);
y=auread('sat4.au');
[Pyy,F] = PSD(y, 1024,16000,hanning(256),
128);
c=xcorr(Pxx,Pyy,'coeff');                     % compute cross correlation
c=c(513)                                      % cross correlation at zero lag
```

Above code was repeated for each cross correlation computation.

APPENDIX E

MATLAB CODE FOR THE NEURAL NETWORK IMPLEMENTATION

E.1 Introduction

This appendix lists the MATLAB source code for the neural network pattern classifiers used in the MATLAB implementation of the speaker recognition system.

E.2 Multi-Layer Perceptron (MLP) Neural Network

This code calculates the PSD coefficients and trains a MLP network with the coefficients as input.

```
vir3=auread('vir3.au');          % read each file from disk
vir3=psd(vir3,10000);            % compute 10000 pt. PSD
sat3=auread('sat3.au');           PSD variables are named same as
sat3=psd(sat3,10000);               au format variables, saves space
vir4=auread('vir4.au');             in RAM.
vir4=psd(vir4,10000);
sat4=auread('sat4.au');
sat4=psd(sat4,10000);
vir5=auread('vir5.au');
vir6=auread('vir6.au');
vir5=psd(vir5,10000);
```

```
vir6=psd(vir6,10000);
vir7=auread('vir7.au');
vir7=psd(vir7,10000);
vir8=auread('vir8.au');
vir9=auread('vir9.au');
vir10=auread('vir10.au');
vir8=psd(vir8,10000);
vir9=psd(vir9,10000);
vir10=psd(vir10,10000);
sat5=auread('sat5.au');
sat6=auread('sat6.au');
sat5=psd(sat5,10000);
sat6=psd(sat6,10000);
sat7=auread('sat7.au');
sat7=psd(sat7,10000);
sat8=auread('sat8.au');
sat8=psd(sat8,10000);
sat9=auread('sat9.au');
sat10=auread('sat10.au');
sat9=psd(sat9,10000);
sat10=psd(sat10,10000);
kum3=auread('kum3.au');
kum3=psd(kum3,10000);
kum4=auread('kum4.au');
kum4=psd(kum4,10000);
kum5=auread('kum5.au');
kum6=auread('kum6.au');
kum5=psd(kum5,10000);
kum6=psd(kum6,10000);
kum7=auread('kum7.au');
```

```
kum7=psd(kum7,10000);
kum8=auread('kum8.au');
kum9=auread('kum9.au');
kum10=auread('kum10.au');
kum8=psd(kum8,10000);
kum9=psd(kum9,10000);
kas3=auread('kas3.au');
kas3=psd(kas3,10000);
kas4=auread('kas4.au');
kas4=psd(kas4,10000);
kas5=auread('kas5.au');
kas6=auread('kas6.au');
kas5=psd(kas5,10000);
kas6=psd(kas6,10000);
kas7=auread('kas7.au');
kas7=psd(kas7,10000);
kas8=auread('kas8.au');
kas9=auread('kas9.au');
kas10=auread('kas10.au');
kas8=psd(kas8,10000);
kas9=psd(kas9,10000);
kas10=psd(kas10,10000);

P=[vir8  kas8 suz8 kum8 kum4 sat8 vir9  kas9      % input matrix with column
suz9 vir10 kas10 suz10 kum5 sat9 sat10 kum6       training vectors
sat4 sat5 vir4  kas4 suz4 kum7 sat6 kum9 vir5
kum10 vir6 kas6 suz6 sat7 vir7 kas5 suz5 kas7
suz7];
T=[11 41 51  31 31 21 11 41 51 11 41 51 31 21 21  % output target vector
31 21 21 11 41 51 31 21 31 11 31 11 41 51 21  11
```

```
41 51 41 51];                                   each value corresponds
                                                to an input vector
plot(P,T,'+');                                  % plot training progress
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
s1=30;                                          % 30 neurons in hidden
                                                layer
[w1,b1,w2,b2]=initff(P,s1,'tansig',T,'purelin'); % initialize network and
                                                weights. Sigmoid followed
                                                by linear layer.

df=10;                                          % display frequency
me=8000;                                        % maximum epochs
eg=0.01;                                        % error goal
lr=0.01;                                        % learning rate
tp=[df me eg lr];                               % training parameters
[w1,b1,w2,b2,ep,tr]=trainbpx(w1,b1,'tansig',w2,b2,' % train network
purelin',P,T,tp);
plottr(tr,eg);
P=sa2
a=simuff(P,w1,b1,'tansig',w2,b2,'purelin')      % test network
```

E.3 LVQ Network (PSD inputs)

This code calculates the PSD coefficients and trains a LVQ network with the coefficients as input.

```
vir3=auread('vir3.au');          % read file from disk
vir3=psd(vir3,500);              % compute 500 pt. PSD
vir4=auread('vir4.au');
vir4=psd(vir4,500);
sat3=auread('sat3.au');
sat3=psd(sat3,500);
sat4=auread('sat4.au');
sat4=psd(sat4,500);
vir5=auread('vir5.au');
vir6=auread('vir6.au');
vir5=psd(vir5,500);
vir6=psd(vir6,500);
vir7=auread('vir7.au');
vir7=psd(vir7,500);
vir8=auread('vir8.au');
vir9=auread('vir9.au');
vir10=auread('vir10.au');
vir8=psd(vir8,500);
vir9=psd(vir9,500);
vir10=psd(vir10,500);
kum3=auread('kum3.au');
kum3=psd(kum3,500);
kum4=auread('kum4.au');
kum4=psd(kum4,500);
kum5=auread('kum5.au');
kum6=auread('kum6.au');
kum5=psd(kum5,500);
kum6=psd(kum6,500);
kum7=auread('kum7.au');
kum7=psd(kum7,500);
```

```
kum8=auread('kum8.au');
kum9=auread('kum9.au');
kum10=auread('kum10.au');
kum8=psd(kum8,500);
kum9=psd(kum9,500);
kum10=psd(kum10,500);
sat5=auread('sat5.au');
sat6=auread('sat6.au');
sat5=psd(sat5,500);
sat6=psd(sat6,500);
sat7=auread('sat7.au');
sat7=psd(sat7,500);
sat8=auread('sat8.au');
sat8=psd(sat8,500);
sat9=auread('sat9.au');
sat10=auread('sat10.au');
sat9=psd(sat9,500);
sat10=psd(sat10,500);
suz3=auread('suz3.au');
suz3=psd(suz3,500);
suz4=auread('suz4.au');
suz4=psd(suz4,500);
suz5=auread('suz5.au');
suz6=auread('suz6.au');
suz5=psd(suz5,500);
suz6=psd(suz6,500);
suz7=auread('suz7.au');
suz7=psd(suz7,500);
suz8=auread('suz8.au');
suz9=auread('suz9.au');
```

```
suz10=auread('suz10.au');
suz8=psd(suz8,500);
suz9=psd(suz9,500);
suz10=psd(suz10,500);
kas3=auread('kas3.au');
kas3=psd(kas3,500);
kas4=auread('kas4.au');
kas4=psd(kas4,500);
kas5=auread('kas5.au');
kas6=auread('kas6.au');
kas5=psd(kas5,500);
kas6=psd(kas6,500);
kas7=auread('kas7.au');
kas7=psd(kas7,500);
kas8=auread('kas8.au');
kas9=auread('kas9.au');
kas10=auread('kas10.au');
kas8=psd(kas8,500);
kas9=psd(kas9,500);
kas10=psd(kas10,500);
```

```
P=[vir3 kum3 suz3 sat3 vir8 suz8 kum4 kas3
   suz4 sat8 kas4 kas5 kum5 vir9 vir10 kas6
   kas7 kas8 kum6 suz5 sat9 kum7 sat10
  suz6 sat4 kum8 sat5 vir4 suz7 kum9 sat6
  suz9 vir5 suz10 kum9 vir6 sat7 vir7 kum10
  kas9 kas10];
```

% input training matrix of 5 speakers. Each speech sample is a column vector.

```
C=[1 4 2 3 1 2 4 5 2 3 5 5 4 1 1 5 5 5 4      % output vector with categories
   2 3 4 3 2 3 4 3 1 2 4 3 2 1 2 4 1 3 1      for each input column vector
   4 5 5];
T=ind2vec(C);                                 % plot training progress
colormap(hsv)
plotvec(P,C)
alabel('p(1)','p(2)','input vectors')
S=30;                                         % number of neurons in hidden layer
[W1,W2]=initlvq(P,S,T);                       % initialize weights
hold on
plot(W1(1,1),W1(1,2),'ow')                    % plot training progress
alabel('p(1),w(1)','p(2),w(3)','input/weig
ht vectors')
df=20;                                        % display frequency
me=10000;                                     % maximum epochs
lr=0.001;%load r;                             % learning rate
tp=[df me lr];                                % training parameters
[W1,W2] = TRAINLVQ(W1,W2,P,T,tp)              % train network
%save r
P=vir8;
a=simulvq(P,W1,W2)                            % test network
```

E.4 LVQ Network (LPC inputs)

This program is similar to the one above except, LPC is computed instead of PSD.

```
vir3=auread('vir3.au');                       % read file from disk
vir3=lpc(vir3,20);                            % compute 20 pt. LPC vector
vir3=vir3';                                   % convert to column vector
```

```
vir4=auread('vir4.au');
vir4=lpc(vir4,20);
vir4=vir4';
sat3=auread('sat3.au');
sat3=lpc(sat3,20);
sat3=sat3';
sat4=auread('sat4.au');
sat4=lpc(sat4,20);
sat4=sat4';
vir5=auread('vir5.au');
vir6=auread('vir6.au');
vir5=lpc(vir5,20);
vir5=vir5';
vir6=lpc(vir6,20);
vir6=vir6';
vir7=auread('vir7.au');
vir7=lpc(vir7,20);
vir7=vir7';
vir8=auread('vir8.au');
vir9=auread('vir9.au');
vir10=auread('vir10.au');
vir8=lpc(vir8,20);
vir8=vir8';
vir9=lpc(vir9,20);
vir9=vir9';
vir10=lpc(vir10,20);
vir10=vir10';
kum3=auread('kum3.au');
kum3=lpc(kum3,20);
kum3=kum3';
```

```
kum4=auread('kum4.au');
kum4=lpc(kum4,20);
kum4=kum4';
kum5=auread('kum5.au');
kum6=auread('kum6.au');
kum5=lpc(kum5,20);
kum5=kum5';
kum6=lpc(kum6,20);
kum6=kum6';
kum7=auread('kum7.au');
kum7=lpc(kum7,20);
kum7=kum7';
kum8=auread('kum8.au');
kum9=auread('kum9.au');
kum10=auread('kum10.au');
kum8=lpc(kum8,20);
kum8=kum8';
kum9=lpc(kum9,20);
kum9=kum9';
kum10=lpc(kum10,20);
kum10=kum10';
sat5=auread('sat5.au');
sat6=auread('sat6.au');
sat5=lpc(sat5,20);
sat5=sat5';
sat6=lpc(sat6,20);
sat6=sat6';
sat7=auread('sat7.au');
sat7=lpc(sat7,20);
sat7=sat7';
```

```
sat8=auread('sat8.au');
sat8=lpc(sat8,20);
sat8=sat8';
sat9=auread('sat9.au');
sat10=auread('sat10.au');
sat9=lpc(sat9,20);
sat9=sat9';
sat10=lpc(sat10,20);
sat10=sat10';
suz3=auread('suz3.au');
suz3=lpc(suz3,20);
suz3=suz3';
suz4=auread('suz4.au');
suz4=lpc(suz4,20);
suz4=suz4';
suz5=auread('suz5.au');
suz6=auread('suz6.au');
suz5=lpc(suz5,20);
suz5=suz5';
suz6=lpc(suz6,20);
suz6=suz6';
suz7=auread('suz7.au');
suz7=lpc(suz7,20);
suz7=suz7';
suz8=auread('suz8.au');
suz9=auread('suz9.au');
suz10=auread('suz10.au');
suz8=lpc(suz8,20);
suz8=suz8';
suz9=lpc(suz9,20);
```

```
suz9=suz9';
suz10=lpc(suz10,20);
suz10=suz10';
kas3=auread('kas3.au');
kas3=lpc(kas3,20);
kas3=kas3';
kas4=auread('kas4.au');
kas4=lpc(kas4,20);
kas4=kas4';
kas5=auread('kas5.au');
kas6=auread('kas6.au');
kas5=lpc(kas5,20);
kas5=kas5';
kas6=lpc(kas6,20);
kas6=kas6';
kas7=auread('kas7.au');
kas7=lpc(kas7,20);
kas7=kas7';
kas8=auread('kas8.au');
kas9=auread('kas9.au');
kas10=auread('kas10.au');
kas8=lpc(kas8,20);
kas8=kas8';
kas9=lpc(kas9,20);
kas9=kas9';
kas10=lpc(kas10,20);
kas10=kas10';
P=[vir3 kum3 suz3 sat3 vir8 suz8 kum4 kas3
   suz4 sat8 kas4 kas5 kum5 vir9 vir10 kas6
   kas7 kas8 kum6 suz5 sat9 kum7 sat10
```

% input training matrix of 5 speakers. Each speech sample is a column vector.

```
suz6
    sat4 kum8 sat5 vir4 suz7 kum9 sat6 suz9
    vir5 suz10 kum9 vir6 sat7 vir7 kum10 kas9
     kas10];
C=[1 4 2 3 1 2 4 5 2 3 5 5 4 1 1 5 5 5 4        % output vector with categories
    2 3 4 3 2 3 4 3 1 2 4 3 2 1 2 4 1 3 1       for each input column vector
    4 5 5];
T=ind2vec(C);                                   % plot training progress
colormap(hsv)
plotvec(P,C)
alabel('p(1)','p(2)','input vectors')
S=30;                                           % number of neurons in hidden layer
[W1,W2]=initlvq(P,S,T);                         % initialize weights
hold on
plot(W1(1,1),W1(1,2),'ow')                      % plot training progress
alabel('p(1),w(1)','p(2),w(3)','input/weig
ht vectors')
df=20;                                          % display frequency
me=10000;                                       % maximum epochs
lr=0.001;%load r;                               % learning rate
tp=[df me lr];                                  % training parameters
[W1,W2] = TRAINLVQ(W1,W2,P,T,tp)                % train network
%save r
P=vir8;
a=simulvq(P,W1,W2)                              % test network
```

APPENDIX F

COMMITTEE OF NEURAL NETWORK RESULTS (MATLAB IMPLEMENTATION)

F.1 Introduction

This appendix presents the results of the five-member LVQ committee of neural networks used during the MATLAB implementation of the speaker recognition system.

Twenty speakers were used to determine the success rate of the system. The target and test values are shown for each speaker sample. The shaded blocks, in the tables that follow, indicate incorrect classifications.

The success rate of the system is determined by the percentage of correct classifications per total test samples e.g. In section F.6, the success rate of the system is (36/40 * 100)= 90%

F.2 Test Results of Member Network LVQ1

SAMPLE	TARGET	TEST
vir1	1	1
vir2	1	1
suz1	2	2
suz2	2	2
sat1	3	3
sat2	3	3
kum1	4	4
kum2	4	4
kas1	5	5
kas2	5	5

SAMPLE	TARGET	TEST
mag1	6	6
mag2	6	6
dad1	7	7
dad2	7	7
mum1	8	8
mum2	8	8
kub1	9	9
kub2	9	9
tam1	10	10
tam2	10	10

SAMPLE	TARGET	TEST
aub1	11	20
aub2	11	11
goo1	12	12
goo2	12	12
lus1	13	13
lus2	13	13
val1	14	14
val2	14	14
bev1	15	15
bev2	15	18

SAMPLE	TARGET	TEST
see1	16	16
see2	16	16
vin1	17	17
vin2	17	17
sha1	18	18
sha2	18	18
anb1	19	17
anb2	19	17
sag1	20	20
sag2	20	20

F.3 Test Results of Member Network LVQ2

SAMPLE	TARGET	TEST
vir1	1	1
vir2	1	1
suz1	2	2
suz2	2	2
sat1	3	3
sat2	3	3
kum1	4	4
kum2	4	4
kas1	5	5
kas2	5	5

SAMPLE	TARGET	TEST
mag1	6	6
mag2	6	6
dad1	7	7
dad2	7	7
mum1	8	8
mum2	8	8
kub1	9	9
kub2	9	9
tam1	10	10
tam2	10	10

SAMPLE	TARGET	TEST
aub1	11	15
aub2	11	13
goo1	12	12
goo2	12	12
lus1	13	13
lus2	13	13
val1	14	14
val2	14	14
bev1	15	15
bev2	15	3

SAMPLE	TARGET	TEST
see1	16	16
see2	16	16
vin1	17	17
vin2	17	17
sha1	18	18
sha2	18	18
anb1	19	19
anb2	19	17
sag1	20	20
sag2	20	20

F.4 Test Results of Member Network LVQ3

SAMPLE	TARGET	TEST
vir1	1	1
vir2	1	1
suz1	2	2
suz2	2	2
sat1	3	3
sat2	3	3
kum1	4	4
kum2	4	4
kas1	5	5
kas2	5	5

SAMPLE	TARGET	TEST
mag1	6	6
mag2	6	6
dad1	7	7
dad2	7	7
mum1	8	8
mum2	8	8
kub1	9	9
kub2	9	9
tam1	10	10
tam2	10	10

SAMPLE	TARGET	TEST
aub1	11	15
aub2	11	13
goo1	12	12
goo2	12	12
lus1	13	13
lus2	13	13
val1	14	14
val2	14	14
bev1	15	15
bev2	15	3

SAMPLE	TARGET	TEST
see1	16	16
see2	16	16
vin1	17	17
vin2	17	17
sha1	18	18
sha2	18	18
anb1	19	19
anb2	19	17
sag1	20	20
sag2	20	20

F.5 Test Results of Member Network LVQ4

SAMPLE	TARGET	TEST
vir1	1	1
vir2	1	1
suz1	2	2
suz2	2	2
sat1	3	3
sat2	3	3
kum1	4	4
kum2	4	4
kas1	5	5
kas2	5	5

SAMPLE	TARGET	TEST
mag1	6	6
mag2	6	6
dad1	7	7
dad2	7	7
mum1	8	8
mum2	8	8
kub1	9	9
kub2	9	9
tam1	10	10
tam2	10	10

SAMPLE	TARGET	TEST
aub1	11	15
aub2	11	13
goo1	12	12
goo2	12	12
lus1	13	13
lus2	13	13
val1	14	14
val2	14	14
bev1	15	15
bev2	15	3

SAMPLE	TARGET	TEST
see1	16	16
see2	16	16
vin1	17	17
vin2	17	17
sha1	18	18
sha2	18	18
anb1	19	19
anb2	19	17
sag1	20	20
sag2	20	20

F.6 Test Results of Member Network LVQ5

SAMPLE	TARGET	TEST
vir1	1	1
vir2	1	1
suz1	2	2
suz2	2	2
sat1	3	3
sat2	3	3
kum1	4	4
kum2	4	4
kas1	5	5
kas2	5	5

SAMPLE	TARGET	TEST
mag1	6	6
mag2	6	6
dad1	7	7
dad2	7	7
mum1	8	8
mum2	8	8
kub1	9	9
kub2	9	9
tam1	10	10
tam2	10	10

SAMPLE	TARGET	TEST
aub1	11	15
aub2	11	13
goo1	12	12
goo2	12	12
lus1	13	13
lus2	13	13
val1	14	14
val2	14	14
bev1	15	15
bev2	15	13

SAMPLE	TARGET	TEST
see1	16	16
see2	16	16
vin1	17	17
vin2	17	17
sha1	18	18
sha2	18	18
anb1	19	19
anb2	19	17
sag1	20	20
sag2	20	20

APPENDIX G

ADC64 SCHEMATICS

G.1 Introduction

Appendix G contains details of the ADC64 hardware [35] used for the DSP implementation of the speaker recognition system.

G.2 Analog Connector Pin-out and Jumper Settings

Table G.1 - Analog connector pin-out

Pin	Signal Name	Description	Pin	Signal Name	Description
1	-12V	-12 power from PC bus	51	+12V	+12 power from PC bus
2	TCLK1	Timer/counter IO from DSP	52	TCLK0	Timer/counter 0 on DSP
3	EXT TRIG3	External Trigger Input 3	53	EXT TRIG2	External Trigger Input 2
4	EXT TRIG1	External Trigger Input 1	54	EXT TRIG0	External Trigger Input 0
5	GATE5*	Gate to 82C54 timer counter 5	55	EXT TRIG4	External Trigger Input 4
6	TMR5	O/p of timer 5	56	TMR_CLK5	Clock input to 82C54 timer counter 5
7	DX15	Digital IO bit 15	57	DX14	Digital IO bit 14
8	DX13	Digital IO bit 13	58	DX12	Digital IO bit 12
9	DX11	Digital IO bit 11	59	DX10	Digital IO bit 10
10	DX9	Digital IO bit 9	60	DX8	Digital IO bit 8
11	DX7	Digital IO bit 7	61	DX6	Digital IO bit 6
12	DX5	Digital IO bit 5	62	DX4	Digital IO bit 4
13	DX3	Digital IO bit 3	63	DX2	Digital IO bit 2
14	DX1	Digital IO bit 1	64	DX0	Digital IO bit 0
15	DVCC	+5V from PC bus	65	DGND	Digital Ground
16	-	Not used	66	-	Not used

17	AGND	Analog Ground	67	DAC0	Output from D/A 0
18	AGND	Analog Ground	68	DAC1	Output from D/A 1
19	IN1	Analog Input 1	69	IN0	Analog Input 0
20	IN3	Analog Input 3	70	IN2	Analog Input 2
21	IN5	Analog Input 5	71	IN4	Analog Input 4
22	IN7	Analog Input 7	72	IN6	Analog Input 6
23	IN9	Analog Input 9	73	IN8	Analog Input 8
24	IN11	Analog Input 11	74	IN10	Analog Input 10
25	IN13	Analog Input 13	75	IN12	Analog Input 12
26	IN15	Analog Input 15	76	IN14	Analog Input 14
27	IN17	Analog Input 17	77	IN16	Analog Input 16
28	IN19	Analog Input 19	78	IN18	Analog Input 18
29	IN21	Analog Input 21	79	IN20	Analog Input 20
30	IN23	Analog Input 23	80	IN22	Analog Input 22
31	IN25	Analog Input 25	81	IN24	Analog Input 24
32	IN27	Analog Input 27	82	IN26	Analog Input 26
33	IN29	Analog Input 29	83	IN28	Analog Input 28
34	IN31	Analog Input 31	84	IN30	Analog Input 30
35	IN33	Analog Input 33	85	IN32	Analog Input 32
36	IN35	Analog Input 35	86	IN34	Analog Input 34
37	IN37	Analog Input 37	87	IN36	Analog Input 36
38	IN39	Analog Input 39	88	IN38	Analog Input 38
39	IN41	Analog Input 41	89	IN40	Analog Input 40
40	IN43	Analog Input 43	90	IN42	Analog Input 42
41	IN45	Analog Input 45	91	IN44	Analog Input 44
42	IN47	Analog Input 47	92	IN46	Analog Input 46
43	IN49	Analog Input 49	93	IN48	Analog Input 48
44	IN51	Analog Input 51	94	IN50	Analog Input 50
45	IN53	Analog Input 53	95	IN52	Analog Input 52
46	IN55	Analog Input 55	96	IN54	Analog Input 54
47	IN57	Analog Input 57	97	IN56	Analog Input 56
48	IN59	Analog Input 59	98	IN58	Analog Input 58
49	IN61	Analog Input 61	99	IN60	Analog Input 60
50	IN63	Analog Input 63	100	IN62	Analog Input 62

G.3 ADC64 Jumper Settings

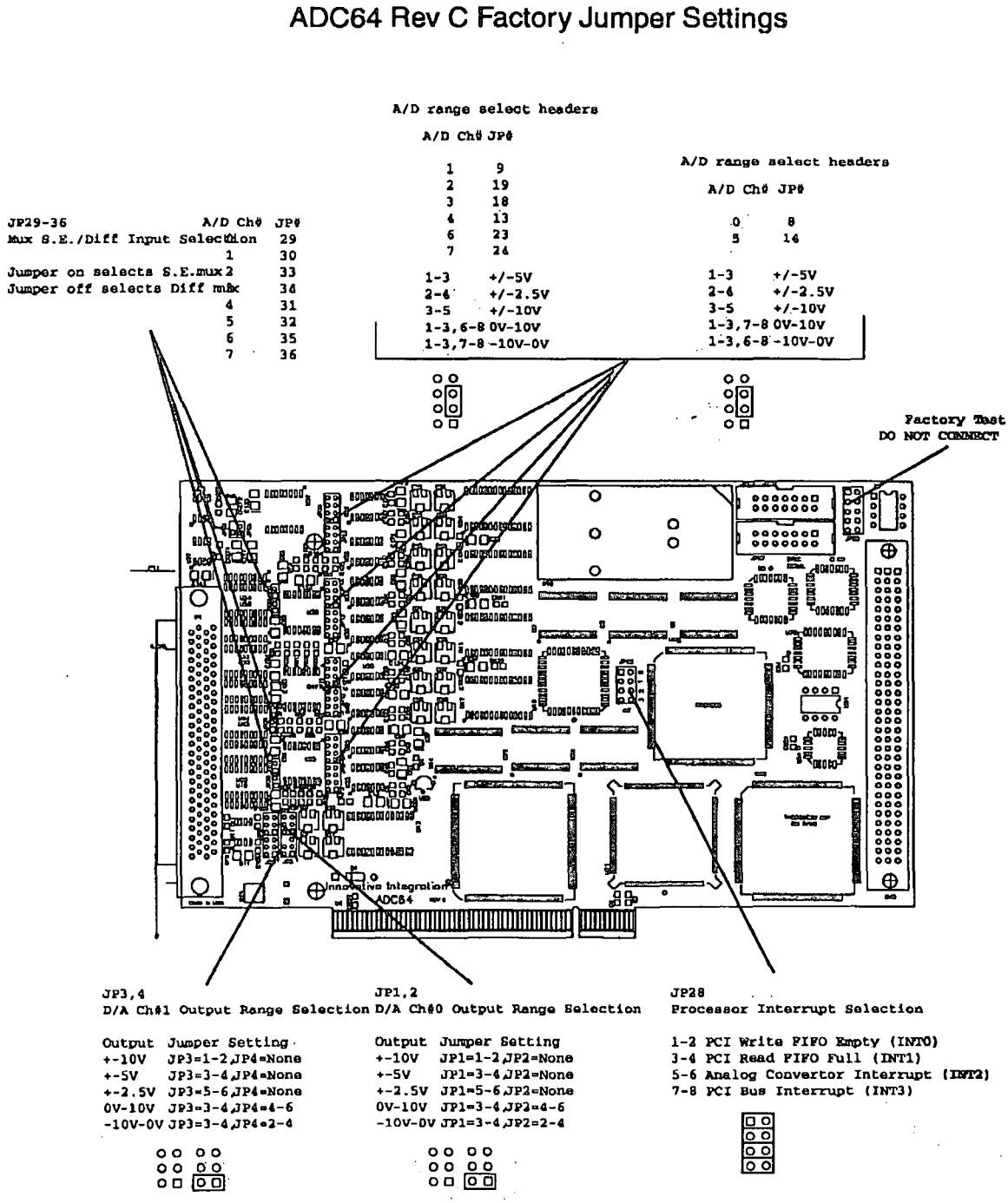


Figure G.1 - ADC64 (Rev C) jumper settings

ADC64 Rev F Factory Jumper Settings

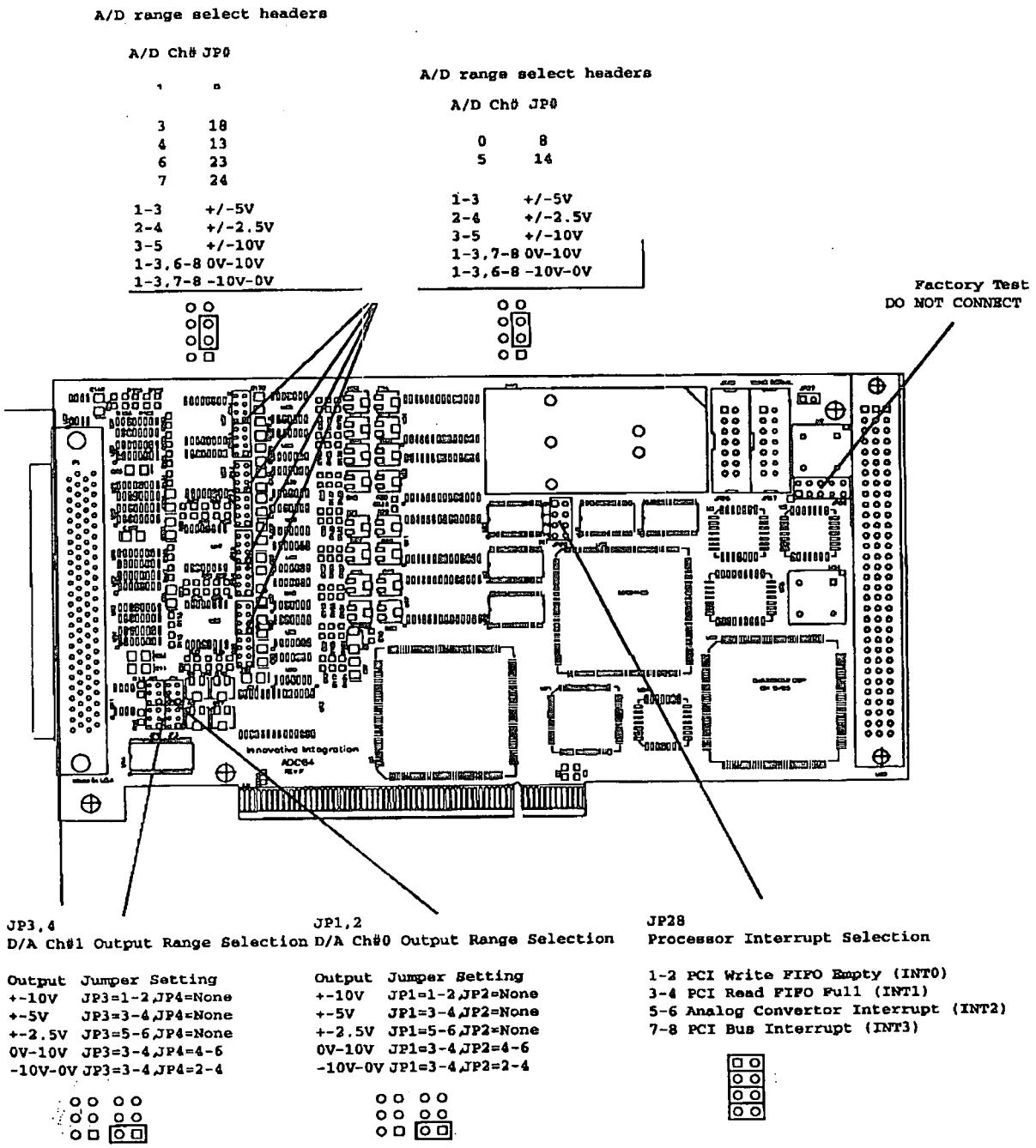


Figure G.2- ADC64 (Rev F) jumper settings

G.4 ADC64 Single-Channel Amplifier

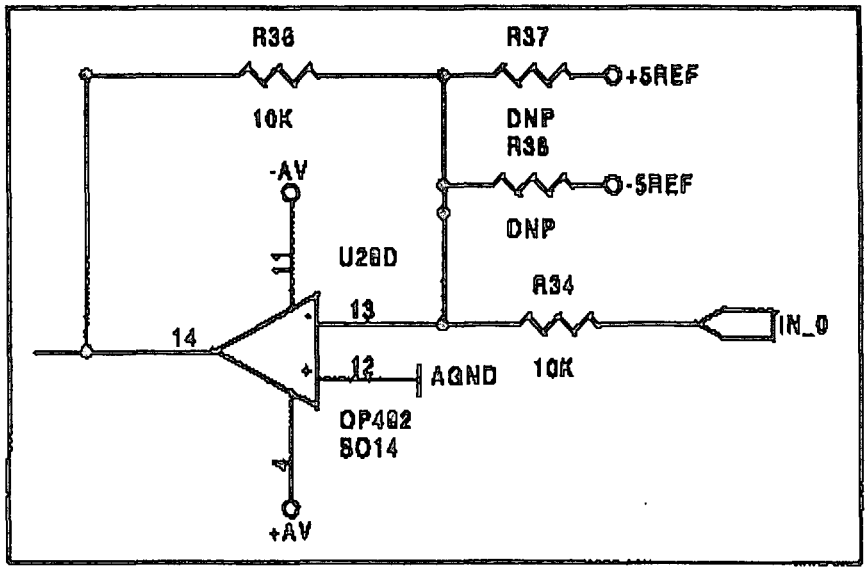


Figure G.3 - Single channel analog amplification

G.5 ADC Input Range Selection

Table G.2 - ADC input range selection

A/D Channel	Input Span	Jumper	Setting	Input Resistor
0	+/-10V	JP8	3-5	R36 = 10K
0	+/-5V	JP8	1-3	R38 = 20K
0	+/-2.5V	JP8	2-4	R40 = 40K
0	0-10V	JP8	1-3	R38 = 20K
0	-10-0V	JP8	1-3	R38 = 20K
1	+/-10V	JP9	3-5	R41= 10K
1	+/-5V	JP9	1-3	R43 = 20K
1	+/-2.5V	JP9	2-4	R44 = 40K
1	0-10V	JP9	1-3	R43 = 20K
1	-10-0V	JP9	1-3	R43 = 20K
2	+/-10V	JP19	3-5	R85 = 10K
2	+/-5V	JP19	1-3	R87 = 20K
2	+/-2.5V	JP19	2-4	R88 = 40K
2	0-10V	JP19	1-3	R87 = 20K
2	-10-0V	JP19	1-3	R87 = 20K
3	+/-10V	JP18	3-5	R81 = 10K
3	+/-5V	JP18	1-3	R83 = 20K
3	+/-2.5V	JP18	2-4	R84 = 40K
3	0-10V	JP18	1-3	R83 = 20K
3	-10-0V	JP18	1-3	R83 = 20K
4	+/-10V	JP13	3-5	R59 = 10K
4	+/-5V	JP13	1-3	R61 = 20K
4	+/-2.5V	JP13	2-4	R62 = 40K
4	0-10V	JP13	1-3	R61 = 20K
4	-10-0V	JP13	1-3	R61 = 20K
5	+/-10V	JP14	3-5	R63 = 10K
5	+/-5V	JP14	1-3	R65 = 20K
5	+/-2.5V	JP14	2-4	R66 = 40K
5	0-10V	JP14	1-3	R65 = 20K
5	-10-0V	JP14	1-3	R65 = 20K
6	+/-10V	JP23	3-5	R102 = 10K
6	+/-5V	JP23	1-3	R104 = 20K
6	+/-2.5V	JP23	2-4	R105 = 40K
6	0-10V	JP23	1-3	R104 = 20K
6	-10-0V	JP23	1-3	R104 = 20K
7	+/-10V	JP24	3-5	R106 = 10K
7	+/-5V	JP24	1-3	R108 = 20K
7	+/-2.5V	JP24	2-4	R109 = 40K
7	0-10V	JP24	1-3	R108 = 20K
7	-10-0V	JP24	1-3	R108 = 20K

G.6 Pre-Amplification

A pre-amplifier was necessary to increase the SNR due to the low output level of the microphone. The circuit diagram of the pre-amplifier used is shown on the next page.

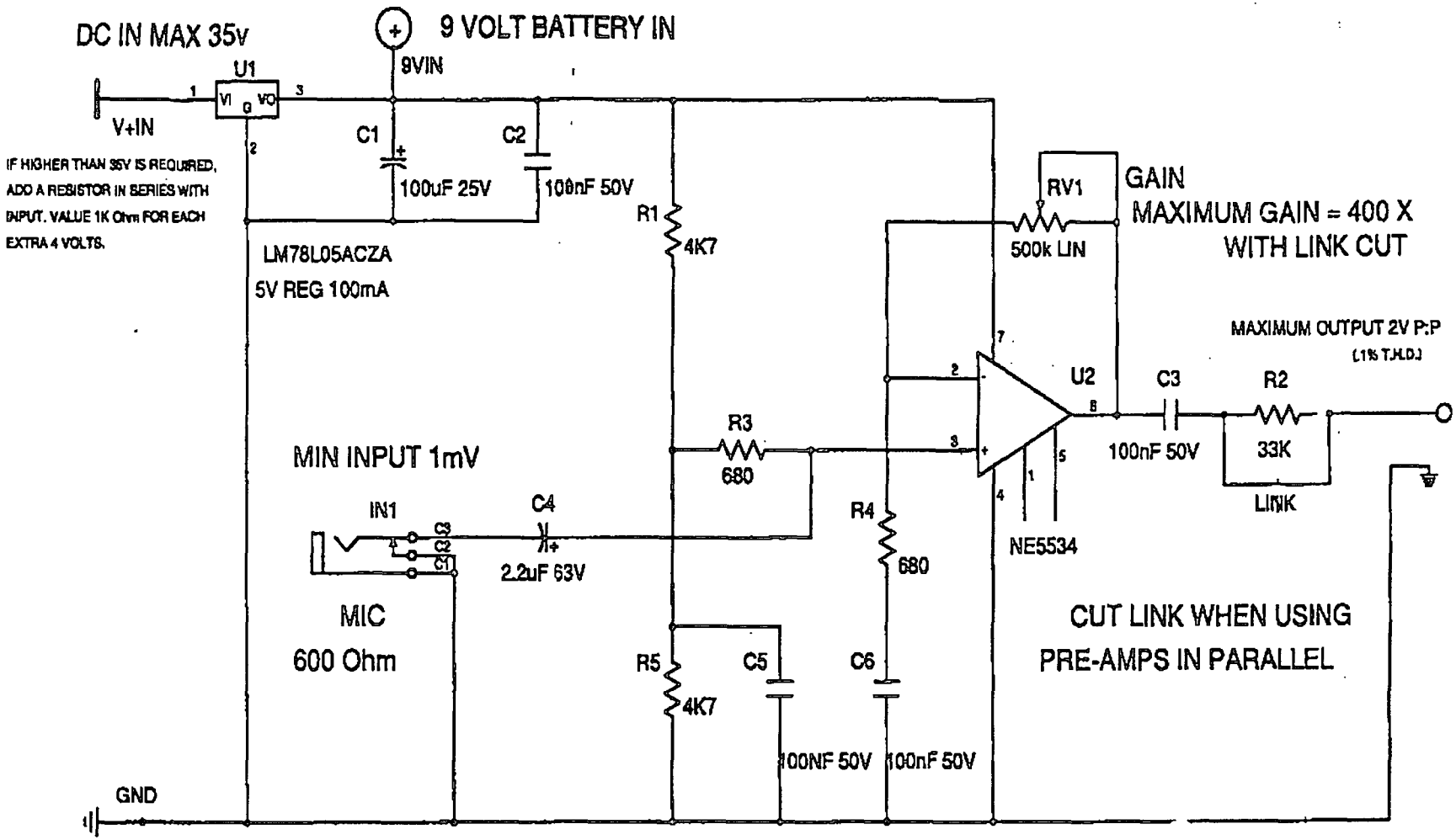


Figure G.4 - System pre-amplifier

APPENDIX H

DSP IMPLEMENTATION SOURCE CODE

H.1 Introduction

This section describes the source code used for the DSP implementation of the speaker recognition system. ANSI-C was used for the sample acquisition, feature parameter extraction and the test mode of the system. MATLAB was used to generate the final trained weights of the pattern recognition network during the training mode of the system.

Despite the diversity of applications, the C used by its practitioners has remained consistent. By 1983, however, the maturity of the C language and its adoption by a community of users beyond its original environment, dictated the need for a standard. ANSI formed the X3J11 committee to provide a definition for the C language independent of any particular implementation.

C is general purpose programming language. It was designed as the systems programming language for the Unix operating system. It is not, however tied to any operating system and is very adaptable. In addition to being used to implement operating systems, language compilers and software tools, it has been used to write major numerical, text processing and database management systems.

H.2 Implemented C Source Code

```
// Viresh Moonasar - 29 September 2001 //
// Post Graduate Research Lab - ML Sultan Technikon //
// ADC64 speech sample acquistion //

#include <math.h>
#include <stddef.h>
#include "periph.h"
#include "stdio.h"
#include "dsp.h"

// Prototypes */
#pragma CODE_SECTION(c_int99, ".onchip");
#define      sample_isr c_int99
void         sample_isr();

#define SIZE 48000        // 2 secs of recorded speech -> memory constraint of
                             ADC64
                          // Tally of ISR executions
                          // Number of samples can be set using this
                          // 4 KHz speech, sampled at 16000
                             samples/sec(Nyquist) implies 48000 ISR executions
                              in 3 secs

#define N 31              // N = number of LPC co-efficients restricted by stack
                             limitation in .cmd linking file , else LPC too high?
#define pi 3.1416         // for cceps and fft/ifft
#define S1 15             // number of neurons in hidden LVQ1 layer
#define S2 5              // number of neurons in linear output LVQ1 i.e. number
of speaker                      classes

volatile float * buffer = (volatile float *) 0x904000;
volatile float R[N] ;
```

```
float a[N+1];
float C[N+1];
volatile int   point;
volatile int   adc_channel;

main()
{
        volatile int   i;
        int            row, col;
        int            rate;

        enable_cache();
        enable_monitor();
        EnableInterrupts();

start:
        clrscr();
        gotoxy(15, 0);
        bold();
        printf("\7Automatic Speaker Recognition (ASR) - Test Phase\n\n");
        normal();
        gotoxy(19, 2);
         printf("\7Viresh Moonasar \n\n");

        timer(PIT0_TIMER, 0);
        //printf("\n\nADC channel to sample (0-7): ");
        //scanf("%d", &adc_channel);
        printf("\n\nADC channel 0 is being sampled at 16000Hz ");
        adc_channel=0;
        rate=16000;
        //printf("\n\nDesired acquisition rate in Hz (<100000): ");
        //scanf("%d", &rate);
        printf("\n\nPress any key to begin recording: ");
```

```
getchar();
point = 0;
enable_analog(BASEBOARD, NULL);

  write_mux(BASEBOARD, 0, 1);
  write_gain(BASEBOARD,0, 1);
  write_mux(BASEBOARD,1, 1);
  write_gain(BASEBOARD,1, 1);

// enable ADC interrupt only
write_analog_interrupt_mask(BASEBOARD, 1 << (adc_channel/2));

  trigger(PIT0_TIMER, adc_channel/2);
  InstallIntVector(sample_isr, EINT2_INTERRUPT);
  EnableInterrupt(EINT2_INTERRUPT);    // Enable ADC interrupt

  printf("\n\nSAMPLE: ");
  wherexy(&col, &row);

  timer(PIT0_TIMER, rate);

// Discard first reading
while (point < 2)
;
point = 0;

   do
{
        gotoxy(col, row);
        printf("%d", point);
  }
  while (point < SIZE);
```

```
        timer(PIT0_TIMER, 0);
        DisableInterrupt(EINT2_INTERRUPT);   // Disable ADC interrupt
        DeinstallIntVector(EINT2_INTERRUPT);

// call the following functions

      normall();
      autocorr();
      lpc();
      cceps();
      write();
      LVQ();
      LVQ2();
      LVQ3();

//for(i=0;i<=(N-1);i++)
//printf("\n R %d %f", i, R[i]);
//getchar();

for(i=0;i<=(N-1);i++)
printf("\n LPC %d %f", i, a[i]);
getchar();
      view((void*)(buffer), SIZE);
//       view((a), N);

   printf("\n\nRe-run test? ");
   if ( tolower(getchar()) == 'y')
   {
        clrscr();
      goto start;
   }
}
```

```
//
// Analog I/O interrupt routine
//

void sample_isr()
{
volatile int i;
        if (point < SIZE)
                {buffer[point++] = (float)read_adc(BASEBOARD, adc_channel);
                        }
}

// normailze input speech - normal conflicts with text
// command

normall()
{
volatile float max;
volatile int i;

for(i=0;i<=(SIZE-1);i++)
 { if(abs(buffer[i]) > max)
    max=abs(buffer[i]);
 }

for(i=0;i<=(SIZE-1);i++)
buffer[i]=(buffer[i]/max);
 return;
}

// autocorelates speech , 1st stage of LPC calculation

autocorr()
```

```
{
//volatile float * prodi ;
volatile float * prodi = (volatile float *) 0x910000;                    //line 194 9E00
 volatile int i,lag;
 volatile float sum;

for(i=0; i<=(SIZE-1); i++)          /*initialize R and Sum*/
  prodi[i]=0;

for(i=0;i<=(N-1);i++)
 R[i]=0;

for(lag=0; lag<=N; lag++)
 {
   sum=0;

   for(i=0; i<=((SIZE-1)-lag); i++)
    {
      prodi[i]=buffer[i]*(buffer[i+lag]);
      sum=(sum+prodi[i]);
    }

  R[lag]=sum;
 }
return;
}

// toeplitz & inversion
// 2nd and 3rd phase of lpc calc

lpc()
{
volatile float toe[N][N];
```

```
volatile float I[N][N];
volatile float * b = (volatile float *) 0x91BC00;          //90FC00
volatile float z;
volatile int i,j,diff,lag;

/* toeplitz: takes bottom half of R and produces NxN hermertian toeplitz matrix
*/

for(i=0; i<=(N-1); i++)
{
 for(j=0; j<=(N-1); j++)
  {
   if(i<j)
     diff=(j-i);
     else
     diff=(i-j);

toe[i][j]=R[diff];
  }
}
/* b = -R(2:N+1)    */

for(i=1; i<(N+1); i++)
b[i-1]=-R[i];

/* LPC=[1; toe[][]\b]]    \ backslash or left division which is roughly the
                          same as INV(toe[][])*b[] */
/* this section uses the Gauss Jordan */
/* method of elimination to calculate */
/* the invert of the square matrix, toe    */

/* initialize unit matrix I */
for(i=0;i<N;i++)
```

```
{for(j=0;j<N;j++)
  {if(i==j)
   I[i][j]=1;
   else
   I[i][j]=0;
  }
 }

/* Gauss Jordan method of Elimination - step one: triangularize bottom half of
matrix
  Perform same functions on the unit matrix I, (ref. Advanced Engineering
  Mathemethics, Erwin Krezig
*/
for(lag=0;lag<(N-1);lag++)          /* lag is just a reused variable, name has
no significance*/
{ for(i=lag;i<(N-1);i++)
 { z=toe[i+1][lag]/toe[lag][lag];
  for(j=0;j<N;j++)
   {
    toe[i+1][j]=(toe[i+1][j])/z;
    toe[i+1][j]=toe[lag][j]-toe[i+1][j];
    I[i+1][j]=(I[i+1][j])/z;
    I[i+1][j]=I[lag][j]-I[i+1][j];
   }
 }
}

/* Gauss Jordan method - step two: triangularize top half of matrix
  Perform same functions on the unit matrix I
*/
for(lag=(N-1);lag>0;lag--)
{
 for(i=lag;i>0;i--)
```

```
{
  z=toe[i-1][lag]/toe[lag][lag];
   for(j=0;j<(N);j++)
   {
    toe[i-1][j]=(toe[i-1][j])/z;
    toe[i-1][j]=toe[i-1][j]-toe[lag][j];

    I[i-1][j]=(I[i-1][j])/z;
    I[i-1][j]=I[i-1][j]-I[lag][j];
   }

 }
}
/* Gauss Jordan elimination step 3 -  normalize toe */

for(i=0;i<(N);i++)
{    z=toe[i][i];
 for(j=0;j<(N);j++)
 {
  I[i][j]=I[i][j]/z;
 toe[i][j]=toe[i][j]/z;
 }
}

/* LPC=[1; toe[][]\b]]   \ backslash or left division which is roughly the
                          same as INV(toe[][])*b[]
                          i.e. I[][]*b[] each row of I * b summed*/
a[0]=1;                /* first element of LPC vector always 1*/

for(i=0;i<N;i++)
{
 z=0;
 for(j=0;j<N;j++)
```

```
{  z=z+(l[i][j]*b[j]);
}
 a[i+1]=z;                  /*this is the LPC co-efficients (N+1) */
}
return;
}

cceps()
{

float real[N+1],imag[N+1], angle[N+1],min,D[N+1];
int k,n,nh;

// calculates the DFT of the input signal
for(k=0;k<=(N);k++)
{
  real[k]=0;
  imag[k]=0;

 for(n=0;n<=(N);n++)
  {
  real[k]=real[k]+(a[n]*cos(2*pi*n*k/(N+1)));
  imag[k]=imag[k]-(a[n]*sin(2*pi*n*k/(N+1)));
  }

 if(imag[k]<0.000001)
  {
   if(imag[k]>0)
     {
      imag[k]=-imag[k];
      goto viresh;
     }
   }
```

```
 if(imag[k]>-0.000001)
 {
 if(imag[k]<0)
  imag[k]=-imag[k];
 }

viresh:
    min=imag[k];
}

// calculate the natural log of the DFT with phase umwrapping
// i.e log of a complex no.

for(k=0;k<=(N);k++)
{
//  Limag[k]=atan(-imag[k]/real[k]);
   angle[k]=atan2(imag[k],real[k]);          // angle of the DFT
 real[k]=log(sqrt((real[k]*real[k])+(imag[k]*imag[k])));
}

// unwrap angle (Limag)     unwrap angle in radians, jumps of greater than pi
are change to their 2pi complements
min=100;              //initialize var
for(k=0;k<=(N);k++)     //find min of angle
{
 if(angle[k]<min)
  min=angle[k];
}

for(k=0;k<=(N);k++)     //phases modulo 2pi
{
 if((angle[k]-min)>(2*pi))
 {                          // angle[k]=((angle[k]-min)%(2*pi))+min;
```

```
 n=(angle[k]-min)/(2*pi);
 angle[k]=((angle[k]-min)-(2*n*pi))+min;
 }
 if((angle[k]-min)<(-2*pi))
 {
 n=(angle[k]-min)/(2*pi);
 angle[k]=((angle[k]-min)+(2*n*pi))+min;
 }
}
min=0;
imag[0]=angle[0];
 D[0]=0;
if(imag[0]>(pi))
    D[0]=(-2*pi);
if(imag[0]<-(pi))
    D[0]=(2*pi);

C[0]=angle[0];
for(k=1;k<=(N);k++)                    // diff phases
 {
  imag[k]=(angle[k]-angle[k-1]);
  D[k]=0;
  if(imag[k]>(pi))
    D[k]=(-2*pi);
  if(imag[k]<(-pi))
    D[k]=(2*pi);

  min=0;                               // cumulative sum
  for(n=0;n<=k;n++)
    min=D[n]+min;
  C[k]=min;
 }
```

```
for(k=0;k<=(N);k++)
imag[k]=angle[k]+C[k];

//rcunwrap      unwrap utility for cceps
// unwraps and removes the phase due to integer lag
n=(N+1)%2;
if(n=0)
nh=(N+1)/2;                    //nh=N/2 if N even else =(N+1)/2

if(n=1)
nh=(N+2)/2;

k=imag[nh]/pi;
min=imag[nh]/pi;
if((min-k)>=0.5)
 k=k+1;                        // round

for(n=0;n<=(N);n++)
imag[n]=imag[n]-(pi*k*n/nh);

// ifft of the log to find the ccep co-efficients
for(n=0;n<=(N);n++)
{
 C[n]=0;
 for(k=0;k<=(N);k++)
   C[n]=C[n]+((real[k]*cos(2*pi*n*k/(N+1)))-(imag[k]*sin(2*pi*n*k/(N+1))))/(N+1);
// complex number multiplication j*j=-1
                                                    //only interested in real
part of ifft
}
return;
}
```

```
write()
{
  int lpcoeff[N+1];
  int Ccep[N+1];
  FILE *stream;
  volatile int i;

stream = fopen("reclpc.bin", "w+t");          /* open file test.bin */
for(i=0;i<=(N);i++)
lpcoeff[i]=to_ieee(a[i]);
fwrite(lpcoeff, 4, (N+1), stream);              /* write  to file , 4 bytes per float
-                          SIZE elements */
fclose(stream);                              /* close file */

stream = fopen("rectccep.bin", "w+t");          /* open file test.bin */
for(i=0;i<=(N);i++)
Ccep[i]=to_ieee(C[i]);
fwrite(Ccep, 4, (N+1), stream);              /* write  to file , 4 bytes per float -
SIZE elements */
fclose(stream);                              /* close file */
return;
}

LVQ()                    /* tests LVQ network */
{
  int i=0,k=0,l=0;                    // must not be volatile
 volatile float x=0,sum=0;
 volatile float max=-100;
 int W[S1][N+1];
 volatile float W1[S1][N+1];
 volatile float W2[S2][S1];              // must be float?
 volatile float n[S1];
//int ka[N+1];
```

```
// float ka10[N+1];
 FILE * stream;
// read trained weights//

stream=fopen("W4.bin", "rb");
fread(W,4,sizeof(W),stream);
fclose(stream);

// stream=fopen("ka10.bin", "rb");
//fread(ka,4,sizeof(ka),stream);
//fclose(stream);

//for(k=0;k<=(N);k++)
//     { ka10[k]=from_ieee(ka[k]);
//      printf("\n ka10 %d  %f", k,ka10[k]);
//     }
//getchar();

//convert W and V to TMS320C32 format
 for(i=0;i<=(S1-1);i++)
 {
  for(k=0;k<=(N);k++)
     { W1[i][k]=from_ieee(W[i][k]);
//       printf("\n W1 %d %d  %f", i,k,W1[i][k]);
     }
  }
//getchar();

for(i=0;i<=(S2-1);i++)
 {
  for(k=0;k<=(S1-1);k++)
      {
       W2[i][k]=0;
```

```
    }
}
W2[0][0]=1;
W2[0][1]=1;
W2[0][2]=1;
W2[1][3]=1;
W2[1][4]=1;
W2[1][5]=1;
W2[2][6]=1;
W2[2][7]=1;
W2[2][8]=1;
W2[3][9]=1;
W2[3][10]=1;
W2[3][11]=1;
W2[4][12]=1;
W2[4][13]=1;
W2[4][14]=1;

// for(i=0;i<=(S2-1);i++)
// {
//  for(k=0;k<=(S1-1);k++)
//      {
//      printf("\n W2 %d %d  %f", i,k,W2[i][k]);
//      }
// }

/* dist() Calculates negative Ecludean distances between input vector and W1
vectors */

for(i=0;i<=(S1-1);i++)
 {
  for(k=0; k<=(N); k++)
  {
```

```
    x=(W1[i][k]-C[k])*(W1[i][k]-C[k]);
    sum=sum+x;
    }
   n[i] = -sqrt(sum);
   sum=0;
   printf("n[%d] %f", i, n[i]);
 }
getchar();

/* compet() hidden competitive transfer function */

for(i=0;i<=(S1-1);i++)
 { if(n[i] > max)
    { max=n[i];
      l=i;
    }
 }                    /* n[i] is the output of the competitive layer */

printf("\n Output of competive layer for this test input is: %f", max);

k=0;                              /*  2nd Linear Layer   */
for(k=0; k<=(S2-1); k++)
  if(W2[k][l]==1)
     i=k;
  printf("\nLVQ1: This is speaker :    %d",(i+1));    /* array elements count from
0 */
return;
}
  LVQ2()                          /* tests LVQ network */

{                              //S1=10
  int i=0,k=0,l=0;                          // must not be volatile
 volatile float x=0,sum=0;
```

```
volatile float max=-100;
int W[10][N+1];
volatile float W1[10][N+1];
volatile float W2[S2][10];                    // must be float?
volatile float n[10];
//int ka[N+1];
//float ka10[N+1];

FILE * stream;

// read trained weights//

stream=fopen("W5.bin", "rb");
fread(W,4,sizeof(W),stream);
fclose(stream);

// stream=fopen("ka10.bin", "rb");
//fread(ka,4,sizeof(ka),stream);
//fclose(stream);

//for(k=0;k<=(N);k++)
//     { ka10[k]=from_ieee(ka[k]);
//       printf("\n ka10 %d  %f", k,ka10[k]);
//     }
//getchar();

//convert W and V to TMS320C32 format
 for(i=0;i<=(10-1);i++)
 {
  for(k=0;k<=(N);k++)
     { W1[i][k]=from_ieee(W[i][k]);
//       printf("\n W1 %d %d  %f", i,k,W1[i][k]);
     }
```

```
 }
//getchar();

for(i=0;i<=(S2-1);i++)
 {
  for(k=0;k<=(10-1);k++)
      {
       W2[i][k]=0;
      }
 }
 W2[0][0]=1;
 W2[0][1]=1;
 W2[1][2]=1;
 W2[1][3]=1;
 W2[2][4]=1;
 W2[2][5]=1;
 W2[3][6]=1;
 W2[3][7]=1;
 W2[4][8]=1;
 W2[4][9]=1;

// for(i=0;i<=(S2-1);i++)
// {
//  for(k=0;k<=(10-1);k++)
//      {
//      printf("\n W2 %d %d  %f", i,k,W2[i][k]);
//      }
// }

/* dist() Calculates negative Ecludean distances between input vector and W1
vectors */

 for(i=0;i<=(10-1);i++)
```

```
{
  for(k=0; k<=(N); k++)
  {
    x=(W1[i][k]-C[k])*(W1[i][k]-C[k]);
    sum=sum+x;
    }
   n[i] = -sqrt(sum);
   sum=0;
   printf("n[%d] %f", i, n[i]);
 }
getchar();

/* compet() hidden competitive transfer function */
for(i=0;i<=(10-1);i++)
 { if(n[i] > max)
    { max=n[i];
      l=i;
    }
 }                    /* n[i] is the output of the competitive layer */

printf("\n Output of competive layer for this test input is: %f", max);

k=0;                              /*  2nd Linear Layer   */
for(k=0; k<=(S2-1); k++)
  if(W2[k][l]==1)
     i=k;
  printf("\nLVQ2: This is speaker :    %d",(i+1));    /* array elements count from
0 */
return;
}

LVQ3()                          /* tests LVQ network */
{                              //S1=20
```

```
 int i=0,k=0,l=0;                    // must not be volatile
volatile float x=0,sum=0;
volatile float max=-100;
int W[20][N+1];
volatile float W1[20][N+1];
volatile float W2[S2][20];              // must be float?
volatile float n[20];
//int ka[N+1];
//float ka10[N+1];
FILE * stream;

// read trained weights//

stream=fopen("W6.bin", "rb");
fread(W,4,sizeof(W),stream);
fclose(stream);
// stream=fopen("ka10.bin", "rb");
//fread(ka,4,sizeof(ka),stream);
//fclose(stream);

//for(k=0;k<=(N);k++)
//     { ka10[k]=from_ieee(ka[k]);
//      printf("\n ka10 %d  %f", k,ka10[k]);
//     }
//getchar();

//convert W and V to TMS320C32 format
 for(i=0;i<=(20-1);i++)
 {
  for(k=0;k<=(N);k++)
     { W1[i][k]=from_ieee(W[i][k]);
//       printf("\n W1 %d %d  %f", i,k,W1[i][k]);
     }
```

```
 }
//getchar();

for(i=0;i<=(S2-1);i++)
 {
  for(k=0;k<=(20-1);k++)
      {
       W2[i][k]=0;
      }
 }
 W2[0][0]=1;
 W2[0][1]=1;
 W2[0][2]=1;
 W2[0][3]=1;
 W2[1][4]=1;
 W2[1][5]=1;
 W2[1][6]=1;
 W2[1][7]=1;
 W2[2][8]=1;
 W2[2][9]=1;
 W2[2][10]=1;
 W2[2][11]=1;
 W2[3][12]=1;
 W2[3][13]=1;
 W2[3][14]=1;
 W2[3][15]=1;
 W2[4][16]=1;
 W2[4][17]=1;
 W2[4][18]=1;
 W2[4][19]=1;

// for(i=0;i<=(S2-1);i++)
// {
```

```
// for(k=0;k<=(20-1);k++)
//     {
//     printf("\n W2 %d %d  %f", i,k,W2[i][k]);
//     }
// }
```

/* dist() Calculates negative Ecludean distances between input vector and W1 vectors */

```
for(i=0;i<=(20-1);i++)
{
  for(k=0; k<=(N); k++)
  {
    x=(W1[i][k]-C[k])*(W1[i][k]-C[k]);
    sum=sum+x;
    }
   n[i] = -sqrt(sum);
   sum=0;
   printf("n[%d] %f", i, n[i]);
 }
getchar();
```

/* compet() hidden competitive transfer function */

```
for(i=0;i<=(20-1);i++)
 { if(n[i] > max)
    { max=n[i];
      l=i;
    }
  }                    /* n[i] is the output of the competitive layer */

printf("\n Output of competive layer for this test input is: %f", max);
```

```
k=0;                              /*  2nd Linear Layer   */
for(k=0; k<=(S2-1); k++)
  if(W2[k][l]==1)
     i=k;
  printf("\nLVQ3: This is speaker :    %d",(i+1));    /* array elements count from
0 */
return;
}
```

H.3 Training Mode Source Code

```
% read stored feature vectors from file
fid=fopen('v1.bin','rb');
v1=fread(fid,'float');
fclose(fid);

fid=fopen('v2.bin','rb');
v2=fread(fid,'float');
fclose(fid);

fid=fopen('v3.bin','rb');
v3=fread(fid,'float');
fclose(fid);

fid=fopen('v4.bin','rb');
v4=fread(fid,'float');
fclose(fid);

fid=fopen('v5.bin','rb');
v5=fread(fid,'float');
fclose(fid);
```

```
fid=fopen('v6.bin','rb');
v6=fread(fid,'float');
fclose(fid);

fid=fopen('v7.bin','rb');
v7=fread(fid,'float');
fclose(fid);

fid=fopen('v8.bin','rb');
v8=fread(fid,'float');
fclose(fid);

fid=fopen('v9.bin','rb');
v9=fread(fid,'float');
fclose(fid);

fid=fopen('v10.bin','rb');
v10=fread(fid,'float');
fclose(fid);

fid=fopen('sa1.bin','rb');
sa1=fread(fid,'float');
fclose(fid);

fid=fopen('sa2.bin','rb');
sa2=fread(fid,'float');
fclose(fid);

fid=fopen('sa3.bin','rb');
sa3=fread(fid,'float');
fclose(fid);

fid=fopen('sa4.bin','rb');
```

```
sa4=fread(fid,'float');
fclose(fid);

fid=fopen('sa5.bin','rb');
sa5=fread(fid,'float');
fclose(fid);

fid=fopen('sa6.bin','rb');
sa6=fread(fid,'float');
fclose(fid);

fid=fopen('sa7.bin','rb');
sa7=fread(fid,'float');
fclose(fid);

fid=fopen('sa8.bin','rb');
sa8=fread(fid,'float');
fclose(fid);

fid=fopen('sa9.bin','rb');
sa9=fread(fid,'float');
fclose(fid);

fid=fopen('sa10.bin','rb');
sa10=fread(fid,'float');
fclose(fid);

fid=fopen('da1.bin','rb');
da1=fread(fid,'float');
fclose(fid);

fid=fopen('da2.bin','rb');
da2=fread(fid,'float');
```

```
fclose(fid);

fid=fopen('da3.bin','rb');
da3=fread(fid,'float');
fclose(fid);

fid=fopen('da4.bin','rb');
da4=fread(fid,'float');
fclose(fid);
fid=fopen('da5.bin','rb');
da5=fread(fid,'float');
fclose(fid);

fid=fopen('da6.bin','rb');
da6=fread(fid,'float');
fclose(fid);

fid=fopen('da7.bin','rb');
da7=fread(fid,'float');
fclose(fid);

fid=fopen('da8.bin','rb');
da8=fread(fid,'float');
fclose(fid);

fid=fopen('da9.bin','rb');
da9=fread(fid,'float');
fclose(fid);

fid=fopen('da10.bin','rb');
da10=fread(fid,'float');
fclose(fid);
```

```
fid=fopen('mu1.bin','rb');
mu1=fread(fid,'float');
fclose(fid);

fid=fopen('mu2.bin','rb');
mu2=fread(fid,'float');
fclose(fid);

fid=fopen('mu3.bin','rb');
mu3=fread(fid,'float');
fclose(fid);

fid=fopen('mu4.bin','rb');
mu4=fread(fid,'float');
fclose(fid);

fid=fopen('mu5.bin','rb');
mu5=fread(fid,'float');
fclose(fid);

fid=fopen('mu6.bin','rb');
mu6=fread(fid,'float');
fclose(fid);

fid=fopen('mu7.bin','rb');
mu7=fread(fid,'float');
fclose(fid);

fid=fopen('mu8.bin','rb');
mu8=fread(fid,'float');
fclose(fid);

fid=fopen('mu9.bin','rb');
```

```
mu9=fread(fid,'float');
fclose(fid);

fid=fopen('mu10.bin','rb');
mu10=fread(fid,'float');
fclose(fid);

fid=fopen('ka1.bin','rb');
ka1=fread(fid,'float');
fclose(fid);

fid=fopen('ka2.bin','rb');
ka2=fread(fid,'float');
fclose(fid);

fid=fopen('ka3.bin','rb');
ka3=fread(fid,'float');
fclose(fid);

fid=fopen('ka4.bin','rb');
ka4=fread(fid,'float');
fclose(fid);

fid=fopen('ka5.bin','rb');
ka5=fread(fid,'float');
fclose(fid);

fid=fopen('ka6.bin','rb');
ka6=fread(fid,'float');
fclose(fid);

fid=fopen('ka7.bin','rb');
ka7=fread(fid,'float');
```

```
fclose(fid);

fid=fopen('ka8.bin','rb');
ka8=fread(fid,'float');
fclose(fid);

fid=fopen('ka9.bin','rb');
ka9=fread(fid,'float');
fclose(fid);

fid=fopen('ka10.bin','rb');
ka10=fread(fid,'float');
fclose(fid);

P=[vir1 sat1 dad1 mum1 kas1 vir2 vir3 dad2 dad3 sat2 sat3 mum2 mum3 kas2
kas3 vir4 sat4 mum4 dad4 kas4 vir5 vir6 vir7 mum5 mum6 mum7 dad5 dad6
dad7 sat5 sat6 sat7 kas5 kas6 kas7 vir8 sat8 dad8 mum8 kas8 vir9 sat9 dad9
kas9 mum9];
C=[1 2 3 4 5 1 1 3 3 2 2 4 4 5 5 1 2 4 3 5 1 1 1 4 4 4 3 3 3 2 2 2 5 5 5 1 2 3 4 5
1 2 3 5 4];

T=ind2vec(C);
colormap(hsv)
plotvec(P,C)
alabel('p(1)','p(2)','input vectors')
S=15;
[W1,W2]=initlvq(P,S,T);
hold on
plot(W1(1,1),W1(1,2),'ow')
alabel('p(1),w(1)','p(2),w(3)','input/weight vectors')
df=100;
me=5000;
lr=0.001;
```

```
%load weight.mat
tp=[df me lr];
[W1,W2] = TRAINLVQ(W1,W2,P,T,tp)
save rec82.mat W1 W2;
P=v8;
a=simulvq(P,W1,W2)

write trained final weights to file
% W1=W1'
%fid=fopen('c:\c3xtools\W.bin','w');
% fwrite(fid,W1,'float');
% fclose(fid);
```

APPENDIX I

MEMORY ALLOCATION OF THE TARGET SYSTEM

I.1 Introduction

The memory allocations of the ADC64 target system are described in this appendix. This is specified in the CMD file, which is called during the linking process.

I.2 CMD Linker Input File

```
-c
-stack 0x1000
-heap 0x1000
rec10.obj
-i \adc64cc\lib\target
-i \adc64cc\stdio
-l stdio.lib
-l periph.lib
-l dsp.lib
-l rts30.lib                    /* runtime support */


/* SPECIFY THE SYSTEM MEMORY MAP */

{
  BL_VECS:   org = 0x900000 len = 0x40      /* Vectors in BL mode      */
```

```
  EXT0:    org = 0x001000 len = 0x400    /* External memory       */
  RAM :    org = 0x87fe00 len = 0x200    /* Ram block 0/1         */
  EXT1:    org = 0x900800 len = 0x1F600   /* orign External memory 0x7800
- 128kwords=                                0x91FFF   */
}

/* SPECIFY THE SECTIONS ALLOCATION INTO MEMORY */

SECTIONS
{
   BL_vec:    > BL_VECS      /* Interrupt Vectors        */
   .onchip:   > RAM          /* On-chip Code             */
   .text:     > EXT1         /* Code                     */
   .data      > EXT1         /* Data                     */
   .cinit:    > EXT1         /* Initialization tables    */
   .const:    > EXT1         /* Constants                */
   .stack:    > EXT1         /* System stack             */
   .bss:     > EXT1, block 0x10000      /* Variables      */
   .sysmem    > EXT1         /* Heap                     */
}
```

I.3 Memory Dump of the Output COFF File

```
FILE HEADER INFORMATION:
File magic number =c2
Number of sections = 9
Date and time stamp = (GMT) Sun Jan 20 07:21:48 2002

Relocation information stripped from file


OPTIONAL FILE HEADER INFORMATION:
            Magic number is      108
```

Tool version number is 5.00

.text (Executable code) size: 1a02h words
.data (Initialized data) size: 0h words
.bss (Uninitialized data) size: 22ah words
Program entry point is: 90179ah
Initialized data starts at: 900800h
Text section starts at: 900800h

SECTION HEADER INFORMATION:

BL_vec starts at 00900000h, length 00000040h, contains executable code.
.onchip starts at 0087fe00h, length 00000091h, contains initialized data.
.text starts at 00900800h, length 00001a02h, contains executable code., contains initialized data.
.data starts at 00900800h, length 00000000h, contains initialized data.
.cinit starts at 00902202h, length 00000150h, contains initialized data.
.const starts at 00902352h, length 000001dah, contains initialized data.
.stack starts at 0090252ch, length 00001000h, contains uninitialized data.
.bss starts at 0090352ch, length 0000022ah, contains uninitialized data.
.sysmem starts at 00900800h, length 00000000h, a dummy section.

REFERENCES

[1] Moonasar V, "Speaker Recognition using Neural Networks", *Electronic Engineering Degree Thesis,* ML Sultan Technikon, South Africa, 1998.

[2] Furui S, "Speaker Recognition", *NTT Human Interface Laboratories*, Tokyo, Japan, 1997.

[3] Moonasar V, Venayagamoorthy GK, "Speaker Identification using a Combination of Different Parameters as Feature Inputs to an Artificial Neural Network Classifier", *IEEE Africon 99 Conference*, Cape Town, South Africa, 28 September - 2 October 1999, vol. 1, pp. 189 - 194.

[4] Moonasar V, Venayagamoorthy GK, "Artificial Neural Network based Automatic Speaker Recognition using a Hybrid Technique for Feature Extraction", *Artificial Neural Networks in Engineering Conference ANNIE 2000,* St. Louis, USA, 5-8 November, 2000, vol. 10, pp. 745-750.

[5] Moonasar V, Venayagamoorthy GK, "A Committee of Neural Networks for Automatic Speaker Recognition", *IEEE-INNS International Joint Conference on Neural Networks,* Washington DC, USA, 15-19 July, 2001, pp. 2936-2940.

[6] Moonasar V, Venayagamoorthy GK, "Optimization of Hybrid Feature Extraction Vectors for Automatic Speaker Recognition

Systems", *Journal of the School of Postgraduate Studies,* ML Sultan Technikon, Durban, February 2001, vol. 1, pp. 14-24.

[7] Moonasar V, Venayagamoorthy GK, "Automatic Speaker Recognition (ASR) Systems using Neural Networks: A Committee Approach" *Journal of the School of Postgraduate Studies,* ML Sultan Technikon, Durban, November 2001, vol. 2, pp. 10-17.

[8] Gibbon B et al (Ed.), "*EAGLES (Expert Advisory Groups on Language Engineering Standards) HANDBOOK of Standards and Resources for Spoken Language Systems*", Walter de Gruyter Publishers, Berlin & New York, 1995.

[9] Lee K, "*Automatic Speech Recognition 'The Development of the SPHINX System",* Kluwer Academic Publishers, Netherlands, 1989.

[10] Rabiner LR et. al "Comparative Performance Study of Several Pitch Detector Algorithms", *IEEE Transactions on Acoustics, Speech and Signal Processing,* October 1976, vol. 24, No. 5, pp. 399-404.

[11] Poulton AS, "*Microcomputer Speech and Synthesis*", Sigma Technical Press, Cheshire-UK, 1983.

[12] Luettin J, "*Visual Speech and Speaker Recognition",* PhD Dissertation, University of Sheffield, England, 1997.

[13] Venayagamoorthy GK, Moonasar V, Sandrasegaran K, "Voice Recognition Using Neural Networks", *Proceedings of IEEE Comsig*

'98, University of Cape Town, South Africa, 7-8 September 1998, pp. 29 –32.

[14] Hermansky H, "Perceptual Linear Predictive (PLP) Analysis of Speech", *Journal of the Acoustical Society of America,* 1990, 87(1), pp. 1738-1752.

[15] Ifeachor EC and Jervis BW, "*Digital Signal Processing, A Practical Approach*", Addison-Wesley, U.S, 1993.

[16] Watanabe H and Katagiri S, *"HMM Speech Recognizer based on Discriminitive Metric Design*", IEEE, 1997, pp. 3237-3240.

[17] Morris LR, "Fast Speech Spectrogram Reduction and Display on Minicomputer/Graphics Processors," *IEEE Transactions on Acoustics, Speech, & Signal Processing, June 1975, USA, vol. ASSP-23, no.3, pp.297-300.*

[18] Chengalvarayan R and Deng L, "Use of Generalized Dynamic Feature Parameters for Speech Recognition", *IEEE Transactions On Speech And Audio Processing*, 1997, vol. 5, No. 3, pp. 232-242.

[19] Cohen MA and Grossberg S, "Parallel Auditory Filtering by Sustained and Transient Channels Separates Coarticulated Vowels and Consonants", *IEEE Transactions On Speech And Audio Processing*, 1997, vol. 5, No. 4, pp. 301-317.

[20] Fallside F (ed.), "*Computer Speech Processing",* Prentice/Hall International, UK, 1985.

[21] Pawate BI, "*Implementation of an HMM-based, Speaker-independent, speech recognition system on the TMS320C2x and TMS320C5x'*, Speech and Image Understanding Laboratory, Computer Sciences Centre, Texas instruments, Incorporated, 1996.

[22] Riis SK, "Hidden Markov Models and Neural Networks for Speech Recognition", PhD Thesis, Department of Mathematical Modelling, Technical University of Denmark, 1998.

[23] Zurada JM, "*Introduction to Artificial Neural Systems*", West Publishing, St. Paul, USA, 1992.

[24] Kohonen T, "*Self-Organizing Maps*", *2nd Edition,* Springer-Verlag, *Berlin*, 1997.

[25] Demuth H, and Beale M, "*Matlab Neural Network Toolbox - User's guide*", Natick, USA, 1994.

[26] Kohonen T, "*Self-Organization and Associative Memory*", 2nd Edition, Springer-Verlag, Berlin, 1987.

[27] "The *DARPA Neural Network Study*", AFCEA International Press, 1988, pp. 60.

[28] Haykin S, "*Neural Networks: A Comprehensive Foundation*", Macmillan, NY, 1994.

[29] Nigrin A, "*Neural Networks for Pattern Recognition*", The MIT Press, Cambridge, 1993.

[30] Zurada JM, "*Introduction To Artificial Neural Systems*", PWS Publishing Company, Boston, 1992, pp. xv.

[31] *Cool Edit User's Manual*, Syntrillium Software Corporation, Phoenix, USA, 1995.

[32] Krauss TP et. al, "*Signal Processing Toolbox*", The MathWorks Inc.,1995.

[33] McIlraith AL, Card HC, "Birdsong Recognition using Backpropagation and Multivariate Statistics", *IEEE Transactions On Signal Processing*, 1997, vol. 45, No. 11, pp. 1053-1587.

[34] Kohonen T, et. al, "*The Learning Vector Quantization Program Package*", Laboratory of Computer and Information Science, Helsinki University of Technology, Rakentajanaukio 2 C, SF-02150 Espoo, Finland, 1995.

[35] "*ADC64/cADC64 Hardware Manual*", Innovative Integration, Westlake Village, USA, 1996.

[36] "*ADC64/cADC64 Development Package Software Manual*", Innovative Integration, Westlake Village, USA, 1997.

[37] Barclay KA, "*ANSI C, Problem Solving and Programming*", Prentice Hall International, UK, 1990.

[38] *"TMS320C3x/C4x Optimizing C Compiler User's Guide"*, Digital Signal Processing Solutions, Texas Instruments, Custom Printing Company, Missouri, USA, 1997.

[39] Kreyszig E, "*Advanced Engineering Mathematics*", seventh edition, John Wiley & Sons, Singapore, 1993.

[40] Spiegel MR, "*Theory and Problems of Advanced Mathematics for Engineers and Scientists*", Schaum's Outline Series, McGraw-Hill Book Co., Singapore, 1983.

[41] Liu LC, et al, "*Layered Neural Nets Applied in the Recognition of Voiceless Unaspirated Stops*", IEEE Proceedings, 1991, vol. 138, No.2, pp. 69-75.

[42] Venayagamoorthy GK, "*Implementation of a Continually Online Trained Artificial Neural Network Controller for a Turbogenerator*", Masters Thesis, University of Natal, Durban, 1998.

[43] Cole RA (ed) et al, "*Survey of the State of the Art in Human Language Technology*", National Science Foundation and European Commission, 1996.