

Spring 2007

Word boundary detection

Deepak Jadhav

University of New Hampshire, Durham

Follow this and additional works at: <https://scholars.unh.edu/thesis>

Recommended Citation

Jadhav, Deepak, "Word boundary detection" (2007). *Master's Theses and Capstones*. 268.
<https://scholars.unh.edu/thesis/268>

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.

WORD BOUNDARY DETECTION

BY

DEEPAK JADHAV

B.E. Babasaheb Ambedkar Marathwada University, India, 2000

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Masters of Science

In

Electrical Engineering

May 2007

UMI Number: 1443609

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1443609

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This thesis has been examined and approved.



Thesis Director, Andrew L. Kun Ph.D
Associate Professor of Electrical Engineering



W. Thomas Miller III Ph.D
Professor of Electrical Engineering



Kent Chamberlin Ph.D
Professor of Electrical Engineering

3/19/07

Date

ACKNOWLEDGEMENT

I would like to thank my advisor, Dr. Andrew L. Kun, for his constant support and guidance throughout the research and writing of this thesis.

I would like to thank my thesis committee for their valuable input.

Finally, I would like to thank my family and friends without whom this thesis would never have come to fruition.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES.....	vi
ABSTRACT	x

CHAPTER	NAME
CHAPTER 1: INTRODUCTION	1
1.1 Need For Accurate Boundary Detection:	1
1.2 Problem Definition.....	2
1.3 Goal	4
1.4 Proposed Steps	5
1.5 Thesis Organization	6
CHAPTER 2: BACKGROUND	7
2.1 Three Approaches to Word Recognition.....	7
2.2 Speech Representation.....	9
2.3 Noise.....	15
2.4 Past Research	17
CHAPTER 3: TESTING TOOL.....	21
3.1 Overview	21
3.2 Features	22
3.3 Overview of Testing Tool Software	34
CHAPTER 4: OUR PROPOSED ALGGORITHM	45
4.1 Preemphasis.....	45
4.2 Frame Blocking.....	46
4.3 Windowing.....	47
4.4 Cepstral Coefficients.....	48

4.5	Silence Template.....	50
4.6	LFCC Distance Computation	51
4.7	LFCC Magnitude Envelope Smoothing	52
4.8	Threshold Computation	55
4.9	Pulse Extraction	56
CHAPTER 5: TESTING		60
5.1	Testing Overview.....	60
5.1.1	Baseline Testing.....	60
5.1.2	Algorithm Performance Testing	61
5.2	Test Setup.....	62
5.2.1	Stage 1:.....	62
5.2.2	Stage 2:.....	64
5.2.3	Stage 3:.....	64
CHAPTER 6: RESULTS		72
6.1	Preliminary Results.....	72
6.2	Testing.....	74
6.2.1	The Time Algorithm	76
6.2.2	The LPC Algorithm	83
6.2.3	The LFCC Algorithm.....	91
6.2.4	The MFCC Algorithm.....	98
6.2.5	Comparison of Algorithm Performances.....	105
CHAPTER 7: CONCLUSIONS AND FUTURE RESEARCH		108
7.1	Conclusions.....	108
7.2	Future Research.....	110
BIBLIOGRAPHY		111
APPENDIX: IRB APPROVAL LETTER.....		113

LIST OF FIGURES

Figure 1.1	noise samples used in this thesis	3
Figure 2.1	explicit approach.....	8
Figure 2.2	implicit approach	8
Figure 2.3	hybrid approach	9
Figure 2.4	short time magnitude representation of the utterance 'zero'.....	10
Figure 2.5	frequency magnitude representation of the utterance 'zero'.....	11
Figure 2.6	ZCR representation of the utterance 'zero'	12
Figure 2.7	LP spectrum of the phoneme /ee/	13
Figure 2.8	computation of the cepstrum	14
Figure 2.9	frequency spectrum of wind noise.....	16
Figure 2.10	frequency spectrum of white noise.....	16
Figure 2.11	frequency spectrum of pink noise	17
Figure 2.12	the hybrid approach endpoint detector proposed by Lamel et al.....	18
Figure 3.1	testing tool block diagram	21
Figure 3.2	testing tool.....	22
Figure 3.3	frequency response of preemphasizer	23
Figure 3.4	selection of preemphasizer	23
Figure 3.5	suppressing background noise using preemphasis.....	24
Figure 3.6	division of a speech segment into overlapping frames	25
Figure 3.7	selection of frame length/overlap/window	25
Figure 3.8	selection of speech representation.....	26
Figure 3.9	selection of smotherer.....	27
Figure 3.10	selection of threshold.....	28
Figure 3.11	selection of frequency scale.....	29
Figure 3.12	mel scale vs. frequency scale.....	29
Figure 3.13	selection of speech file	30
Figure 3.14	selection of noise file.....	30

Figure 3.15	displaying segmented audio file	32
Figure 3.16	displaying computed boundaries	33
Figure 3.17	automated testing	33
Figure 3.18	overview of testing tool software	34
Figure 4.1	our proposed boundary detection algorithm	45
Figure 4.2	preemphasis of the noisy utterance 'zero'	46
Figure 4.3	framing of a noisy speech recording	47
Figure 4.4	windowed frames	48
Figure 4.5	LFC coefficients of first five frames	50
Figure 4.6	silence template	51
Figure 4.7	LFCC magnitude distance envelope	52
Figure 4.8	LPC and FFT spectra of a vowel phoneme	53
Figure 4.9	concatenated LFCC magnitude envelope	53
Figure 4.10	IFFT of the concatenated LFCC magnitude envelope	54
Figure 4.11	smoother output	55
Figure 4.12	pulse extraction	56
Figure 4.13	LPC smoother produces only one pulse per word.	58
Figure 5.1	baseline testing	60
Figure 5.2	overview of test setup	61
Figure 5.3	generation of test file	63
Figure 5.4	SAPI overview	64
Figure 5.5	software for recognition of WAV files	66
Figure 5.6	construction of file name array	67
Figure 5.7	grammar for recognition of digits 0 to 9	68
Figure 5.8	testing procedure	69
Figure 6.1	preemphasis vs no preemphasis	73
Figure 6.2	performance of various smootheners	74
Figure 6.3	recognition accuracy of the Time algorithm	76
Figure 6.4	an example of the failings of the Time algorithm	77
Figure 6.5	PDF of boundary errors from (Time Algorithm, -30dB to 20dB)	78

Figure 6.6	boundary error positions (Time Algorithm, -30dB to 20 dB)	78
Figure 6.7	PDF of boundary errors (Time Algorithm, 10dB to 20dB)	79
Figure 6.8	boundary error positions (Time Algorithm, 10dB to 20dB)	80
Figure 6.9	PDF of boundary errors (Time Algorithm, -19dB to 9dB)	81
Figure 6.10	boundary error positions (Time Algorithm, -19dB to 9dB)	81
Figure 6.11	PDF of boundary errors (Time Algorithm, -30dB to -20dB)	82
Figure 6.12	boundary error positions (Time Algorithm, -30dB to -20dB)	83
Figure 6.13	recognition accuracy of the LPC algorithm	84
Figure 6.14	PDF of boundary errors (LPC Algorithm, -30dB to 20dB)	85
Figure 6.15	boundary error positions (LPC Algorithm, -30dB to 20dB)	85
Figure 6.16	PDF of boundary errors from (LPC Algorithm, 5dB to 20dB)	86
Figure 6.17	boundary error positions (LPC Algorithm, 5dB to 20dB)	87
Figure 6.18	PDF of boundary errors (LPC Algorithm, -24dB to 4dB)	87
Figure 6.19	boundary error positions (LPC Algorithm, -24dB to 4dB)	88
Figure 6.20	PDF of boundary errors (LPC Algorithm, -30dB to -25dB)	89
Figure 6.21	boundary error positions (LPC Algorithm, -30dB to -25dB)	89
Figure 6.22	% of files with no boundaries detected	90
Figure 6.23	no boundaries detected	91
Figure 6.24	recognition accuracy of the LFCC algorithm	92
Figure 6.25	PDF of boundary errors (LFCC Algorithm, -30dB to 20dB)	93
Figure 6.26	boundary error positions (LFCC Algorithm, -30dB to 20dB)	93
Figure 6.27	PDF of boundary errors (LFCC Algorithm, 0dB to 20dB)	94
Figure 6.28	boundary error positions (LFCC Algorithm, 0dB to 20dB)	95
Figure 6.29	PDF of boundary errors (LFCC Algorithm, -19dB to -1dB)	95
Figure 6.30	boundary error positions (LFCC Algorithm, -19dB to -1dB)	96
Figure 6.31	PDF of boundary errors (LFCC Algorithm, -30dB to -20dB)	97
Figure 6.32	boundary error positions (LFCC Algorithm, -30dB to -20dB)	97
Figure 6.33	recognition accuracy of the MFCC algorithm	98
Figure 6.34	an example of the failings of the MFCC algorithm	99
Figure 6.35	PDF of boundary errors (MFCC Algorithm, -30dB to 20dB)	100

Figure 6.36	boundary error positions (MFCC Algorithm, -30dB to 20dB).....	100
Figure 6.37	PDF of boundary errors (MFCC Algorithm, 0dB to 20dB)	101
Figure 6.38	boundary error positions (MFCC Algorithm, 0dB to 20dB).....	102
Figure 6.39	PDF of boundary errors (MFCC Algorithm, -19dB to -1dB).....	102
Figure 6.40	boundary error positions (MFCC Algorithm, -19dB to -1dB)	103
Figure 6.41	PDF of boundary errors (MFCC Algorithm, -30dB to -20dB).....	104
Figure 6.42	boundary error positions (MFCC Algorithm, -30dB to -20dB)	104
Figure 6.43	comparison of recognition accuracy	105
Figure 6.44	comparison of boundary errors.....	107

ABSTRACT

WORD BOUNDARY DETECTION

by

Deepak Jadhav

University of New Hampshire, May, 2007

Robust word boundary detection is essential for the efficient and accurate performance of an automatic speech recognition system. Although word boundary detection can achieve high accuracy in the presence of stationary noise with high values of SNR, its implementation becomes non-trivial in the presence of non-stationary noise and low SNR values. The purpose of this thesis is to compare and contrast the accuracy and robustness of various word boundary detection techniques and to introduce modifications to better their performance.

CHAPTER 1

INTRODUCTION

Word boundary detection or endpoint detection involves the separation of speech from unwanted noise. This noise may be background noise or speaker generated artifacts.

1.1 Need For Accurate Boundary Detection:

Word boundary detection is an integral party of an Automatic Speech Recognition System (ASRS).

Accurate word boundary detection in an ASRS is important for three main reasons.

- Accurate word boundary detection lessens the computational load on further recognition stages.
- Greater accuracy in word boundary detection translates into greater accuracy in the overall speech recognition system.
- Word boundary detection techniques may incorporate word recognition techniques, in which case further recognition stages in the speech recognizer may be easier to implement.

1.2 Problem Definition

In concept, word boundary detection involves the use of certain parameters to distinguish spoken word from background noise. In some cases, the values of parameters for a speech sound and a noise sound may vary greatly, in which case boundary detection is relatively more straight forward. However, in many practical scenarios, the speech and noise sounds may be comparable to each other. In such cases, separating speech from noise becomes difficult. For example, unwanted speaker-generated artifacts such as mouth clicks or breathing may be wrongly included within word boundaries. This problem is further aggravated in the presence of higher background noise. Ideal word boundary detection involves extracting the spoken segment of a recording, regardless of the level and nature of the background noise.

The objective of the research reported in this thesis is to evaluate the performance of several word boundary detection algorithms in a mobile environment, namely in a moving vehicle. Common examples of noise in such an environment are wind noise, engine noise, tire and air conditioner noise, as well as impulse noise arising from speed bumps, potholes, etc. Different types of noise affect the boundary detection process with varying degrees.

As an example of real world noise, we have chosen wind noise for testing various boundary detection algorithms. A 2.5 sec sample recording was obtained by holding up a microphone outdoors on a windy day.

White and pink noise samples were also used to test the performance of various boundary detection algorithms under different noise conditions as a measure of algorithm robustness. White noise was obtained using Matlab's random number generator *randn*. Pink noise was downloaded from a web site [2].

These three examples of noise are shown in Figure 1.1.

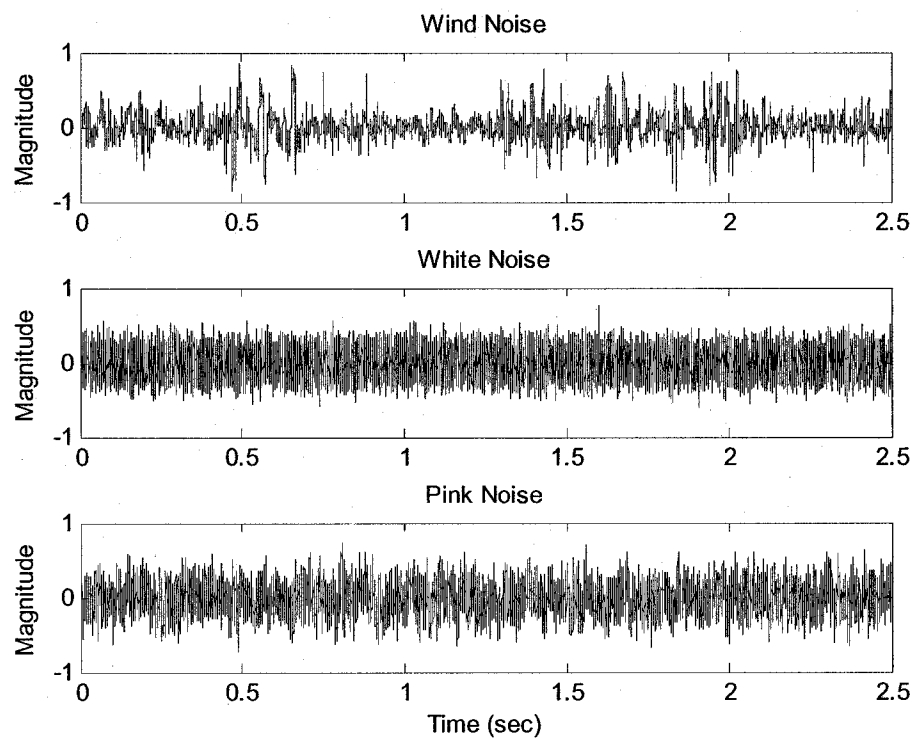


Figure 1.1 noise samples used in this thesis

Any other noise that may have been present in the recording environment or within the recording system itself was ignored.

There have been extensive publications documenting several word boundary detection techniques. Authors such as L. Rabiner [5] and M. Sambur[6] have written a great

deal on this topic. Going through such a vast reservoir of proposed algorithms, one feels the need of a testing tool which, with the click of a few buttons, allows the user to test the robustness of several different algorithms under several noise conditions, and compare their results.

Also, most of the proposed word boundary detection algorithms address signal to noise ratios of down to only 10dB. There is a need for developing word boundary detection algorithms that remain robust under signal-to- noise ratios of less than 10dB.

1.3 Goal

The work completed in this thesis achieved two goals:

The first goal was to develop a testing tool that allowed the user to test and compare the performance of several word boundary detection algorithms. A typical word boundary detection algorithm has several variables associated with it, such as the frame length of the speech segment, design of preemphasizer, type of background noise, etc. The testing tool will allow the user to vary these variables and observe their effects on the performance of these algorithms.

The second goal was to develop a word boundary detection algorithm that will remain robust under noisy conditions with low signal to noise ratios. The detector will be speaker-independent and will operate without any voice training. The algorithm will be tested in a non-real time environment i.e. pre-recorded noisy speech segments will be fed to the algorithms and the corresponding results noted.

1.4 Proposed Steps

The research goals stated in the previous section were achieved by completing three tasks:

Task 1:

We first task was the development of a testing tool in Matlab that gave the user the ability to test several word boundary detection algorithms under a wide selection of noise conditions. The user was able to introduce changes to algorithms and readily observe the resulting changes in boundary detection accuracy. This provided further insight into the factors affecting the accuracy and robustness of various word boundary detection techniques.

Task 2:

The second task was the implementation of several word boundary detection algorithms. These algorithms were tested with pre-recorded speech segments, and their accuracy and robustness under varying noise conditions were observed.

Task 3:

The third task was the development of a word boundary detector whose performance surpassed that of all the boundary detectors we examined. We conducted tests with multiple speakers and noise conditions to ensure that our algorithm demonstrated higher boundary detection accuracy.

1.5 Thesis Organization

This thesis is organized into seven chapters.

The first chapter describes the problem definition and goals of this thesis as well as the steps proposed to reach these goals.

The second chapter explores some signal characteristics of speech as well as a few types of noise. It also summarizes some relevant research conducted by several authors.

The third chapter presents our newly developed testing tool in detail. Examples describing some of its features are illustrated.

The fourth chapter explains our proposed word boundary detection algorithm.

The fifth chapter presents the testing performed to compare and contrast several word boundary detection algorithms, including the algorithm developed in Chapter 4. Also presented is an overview of the software design behind the implementation of the speech recognizer used in this thesis.

The sixth chapter presents the results of the testing performed.

The seventh chapter presents conclusions derived from tests performed and also suggests future research that may provide further insight into this topic.

CHAPTER 2

BACKGROUND

Word boundary detection is used by automatic speech recognition systems to isolate useful speech from background noise in order to extract speech patterns that further recognition stages can recognize.

For speech produced in a relatively noise-free environment, boundary detection is a simple problem. However, high levels of noise, be it background noise or noise in the transmission system, make word boundary detection difficult.

2.1 Three Approaches to Word Recognition

Rabiner and Juang [5] broadly classified endpoint detection approaches as either explicit, implicit, or hybrid depending upon the degree of interaction between the endpoint detection and word recognition stages of the automated speech recognition system (ASRS).

Explicit Approach

An explicit detection approach has the endpoint detection stage occurring prior to and independent of the recognition stage of the speech recognition system. The boundary detection stage may or may not use the same features as the word recognition stage.

This method carries the least computation load but its accuracy is the least of the three approaches[5].

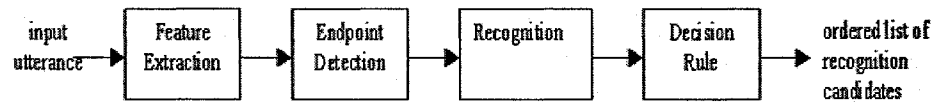


Figure 2.1 explicit approach

Implicit Approach

An implicit approach combines the endpoint detection and word recognition stages. This approach does not focus on extracting speech from noise, but it recognizes the input noisy speech segment as one of several possible noisy speech templates. It produces a list of endpoint pairs ordered according to likelihood. This method has the greatest computational load but exhibits the highest accuracy.

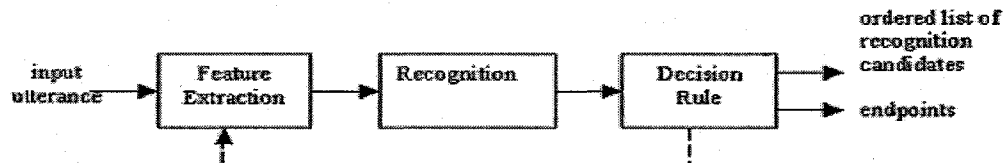


Figure 2.2 implicit approach

Hybrid Approach

In the hybrid approach, the explicit method is employed to compute several estimates of endpoints pairs, and the implicit method is used to choose a small and reasonable

set amongst them. The hybrid approach has a computational load comparable to the explicit method and accuracy levels comparable to the implicit method.

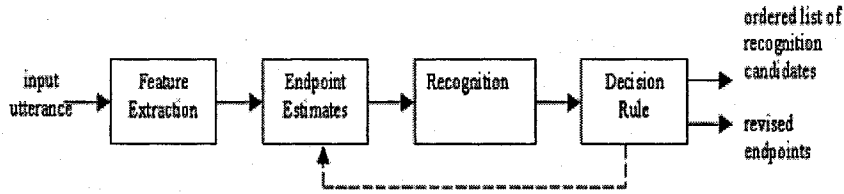


Figure 2.3 hybrid approach

In spite of its low boundary detection accuracy compared to the implicit and hybrid methods, the explicit method inherently allows the testing of the word boundary detection stage independent of the word recognition stage. Any insight obtained from testing the explicit boundary detector can be applied to the hybrid detector as well. Hence, the scope of this thesis was chosen to be limited to the explicit approach of word boundary detection.

2.2 Speech Representation

There are several characteristics of speech that separate it from noise. The ones that are most commonly employed for word boundary detection are as follows:

Short Time Magnitude

The short-time magnitude of a speech segment is computed along time. The time magnitude of vowels is substantially larger than that of consonants and also that of low-level background noise.

The short-time magnitude [6] can be defined as

$$E_n = \sum_{m=n-N+1}^n x^2(m)$$

That is, the short-time magnitude at sample n is simply the sum of squares of N samples $n-N+1$ through n .

In a typical word boundary detection algorithm using short-time magnitude, thresholds are applied to separate the high-magnitude speech from the low-magnitude background noise. Figure 2.4 shows the short-time magnitude for the utterance 'zero'.

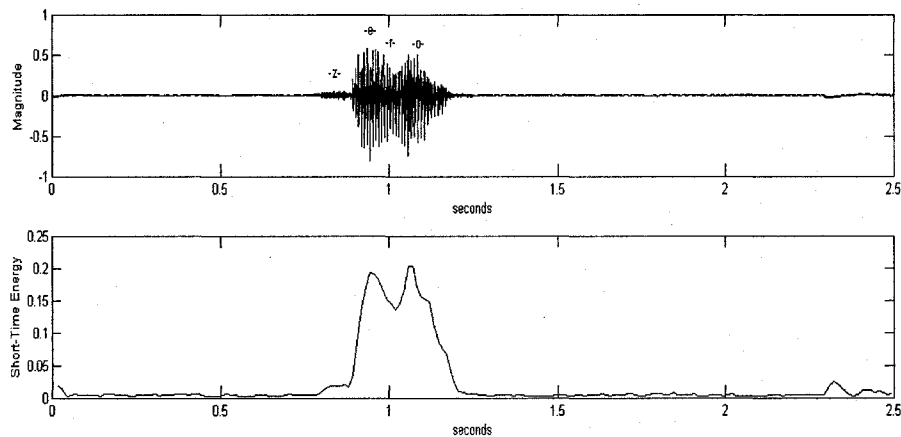


Figure 2.4 short time magnitude representation of the utterance 'zero'

Short Time Frequency Magnitude

Here, the frequency power spectrum of the speech signal is used to distinguish speech from noise. Similar to time magnitude, the frequency magnitude of speech, especially vowels, is distinctly greater than that of low-level background noise.

The frame by frame DFT [4] of a discrete sequence $x[n]$ is given by:

$$X_f(m) = \sum_{n=0}^{L-1} x(n)w(A-n)e^{-j\frac{2\pi m}{N}n}$$

where $A = fL - (f-1)O$

L = frame length

O = frame overlap

f = frame number

N = number of frequencies in the DFT output

w = window of length L

The frame by frame frequency magnitude is then computed as

$$\text{Frequency Magnitude}(f) = \sum_{m=0}^{L-1} |X(m)|$$

Figure 2.5 shows the short-time frequency representation of the utterance 'zero'.

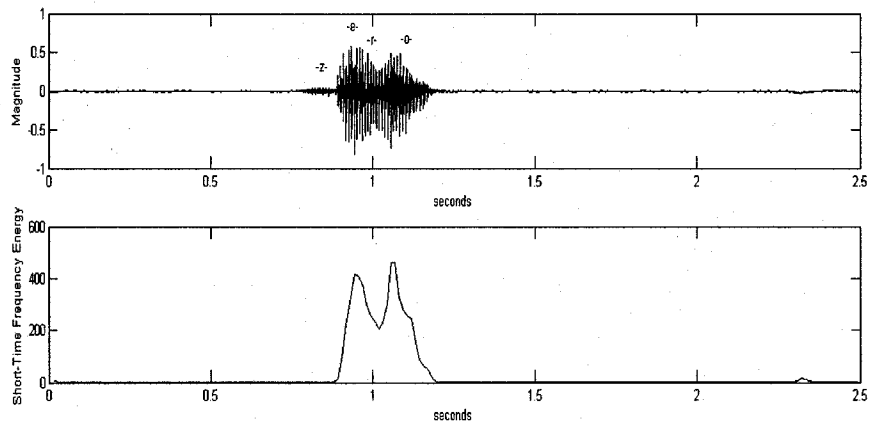


Figure 2.5 frequency magnitude representation of the utterance 'zero'

Short Time Zero Crossing Rate

A signal's zero crossing rate (ZCR) is the rate at which the signal crosses the zero axis. The zero crossing rates of fricatives tend to be greater than those of vowels. By itself, the zero crossing rate is not a very useful parameter for differentiating speech from noise. However, its use with other speech properties may prove beneficial due to its ability to distinguish weak fricatives from background noise.

The short-time average zero crossing rate [6] can be defined as

$$Z_n = \sum_{m=-n}^n |\text{sgn}[x(m)] - \text{sgn}[x(m-1)]| w(n-m)$$

where

$$\begin{aligned} \text{sgn}[x(n)] &= 1 & x(n) &\geq 0 \\ &= -1 & x(n) &< 0 \end{aligned}$$

and

$$\begin{aligned} w(n) &= \frac{1}{2N} & 0 \leq n \leq N-1 \\ &= 0 & \text{otherwise} \end{aligned}$$

Figure 2.6 shows the zero crossing rate for the utterance 'zero'. The high values of the zero crossing rate correspond to the fricative /z/ in 'zero'.

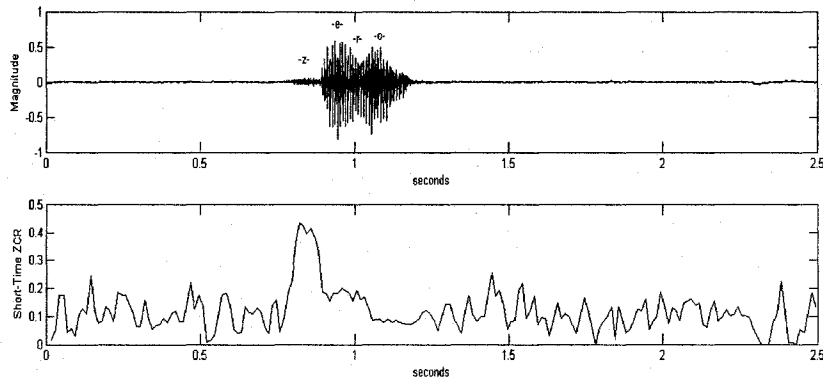


Figure 2.6 ZCR representation of the utterance 'zero'

Linear Prediction Spectrum

Linear Prediction Analysis has traditionally been used for the 'recognition' part of an automated speech recognition system. This thesis explores the possible use of Linear Prediction in word boundary detection.

Linear Prediction models the human vocal tract as an all pole system defined by its linear prediction coefficients. The coefficients are such that they most accurately predict the subsequent speech samples. This becomes useful for boundary detection as the computed coefficients give an idea about whether the produced samples are of speech or unwanted background noise.

A linear predictor with prediction coefficients a_k can be defined [6] as a system that, upon giving it an input $s(n)$, produces an output:

$$\tilde{s}(n) = \sum_{k=1}^p a_k s(n-k)$$

Figure 2.7 shows the LP spectrum of a frame (18.75ms) of the phoneme /ee/.

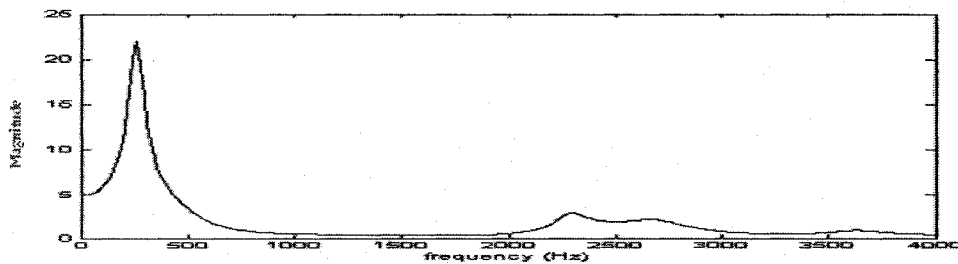


Figure 2.7 LP spectrum of the phoneme /ee/

Cepstrum

In the source filter model of speech production, the vibration of the vocal chords acts as the source and the vocal tract acts as the filter. The cepstrum [3] provides a means of taking a speech signal and separating the source signal from the filter's transfer function as shown in Figure 2.8.

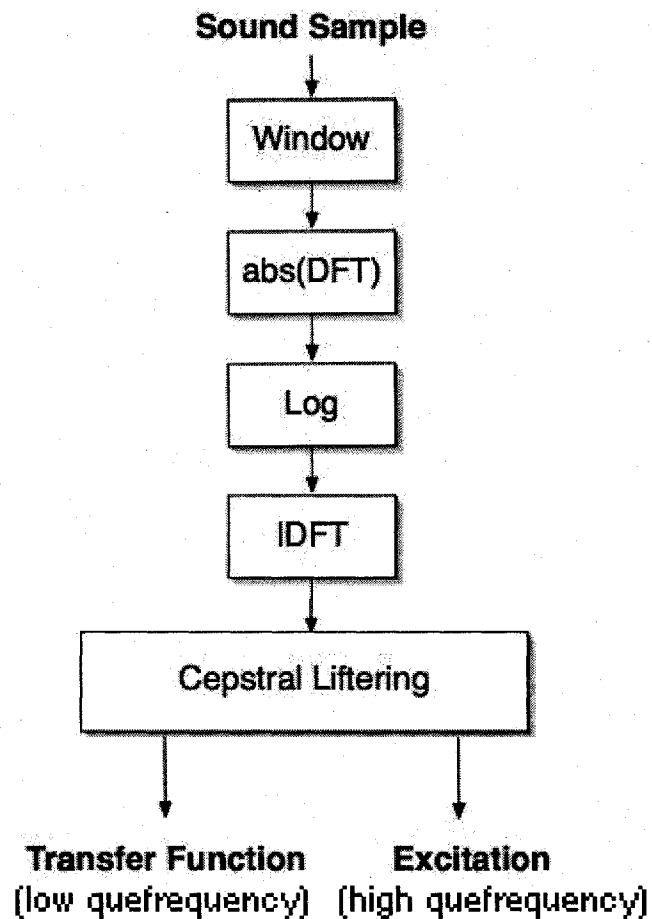


Figure 2.8 computation of the cepstrum

The real cepstrum of a discrete sequence 'x' can also be represented in pseudo code as

$$y = \text{real}(\text{ifft}(\log(\text{abs}(\text{fft}(x))))))$$

As depicted by the pseudo code, the magnitude of the Fourier Transform is taken, and its log computed. The real inverse Fourier Transform of the resultant sequence constitutes the cepstrum of the original sequence y .

Examination of the extracted source (excitation) signal provides valuable information about its nature, i.e. whether the source is of voiced speech, unvoiced speech, or unwanted noise.

2.3 Noise

As this thesis endeavors to design a robust word boundary detection algorithm that is as immune to background noise as possible, we deem it important to study the nature of several examples of noise.

The noise samples used in this thesis fall under two categories.

- real world noise
- colored noise

The real world noise are recordings of noise taken out in the field. Examples would include horn noise, wind noise and siren noise. Figure 2.9 shows the frequency spectrum of the wind noise used for testing in this thesis.

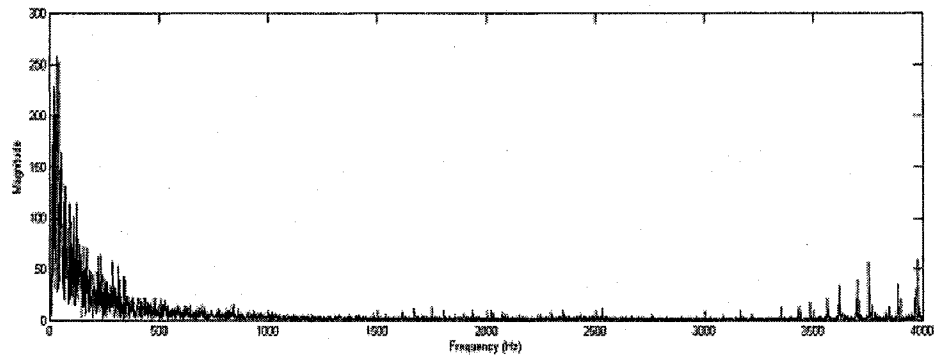


Figure 2.9 frequency spectrum of wind noise

Different colors of noise have different frequency spectra that may effect word boundary detection accuracy differently. The colors of noise used in this thesis are:

White Noise

White noise has uniform frequency magnitude at all frequencies. It typically sounds like the hiss of an untuned radio.

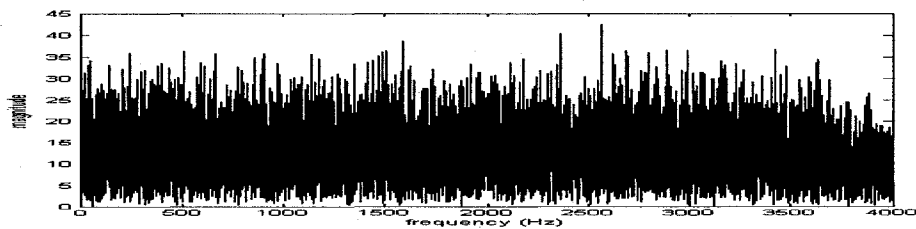


Figure 2.10 frequency spectrum of white noise

Pink Noise

The human ear perceives pink noise as having equal magnitude at all frequencies. The power density of pink noise decreases by 3dB per octave. This noise sounds like a hiss mixed with a rumble.

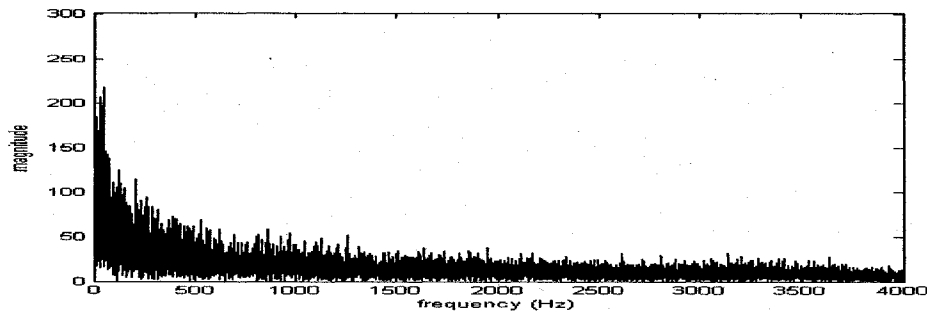


Figure 2.11 frequency spectrum of pink noise

2.4 Past Research

S. Davis and P. Mermelstein [11] compared the performance of several acoustic representations in an automatic speech recognition system based on syllabic segments. The acoustic representations they compared were Mel frequency cepstrum, the linear frequency cepstrum, the linear prediction cepstrum, reflection coefficients and cepstral coefficients derived from linear prediction coefficients. They concluded that the acoustic representation that provided the highest accuracy in word recognition was the Mel frequency cepstrum with six to ten cepstral coefficients. With these coefficients, they reported a recognition rate of 96.5% upon training of the recognition system. Other conclusions they came to were:

- Parameters derived from the short Fourier spectrum such as Mel frequency coefficients and linear prediction coefficients provide adequate representation of vowels just as linear prediction coefficients. However, the frequency derived representations are more adequate for representing consonants than the linear prediction technique.

- The Mel frequency spectrum has a significant advantage over the linear frequency spectrum.
- Cepstral parameters capture acoustic information better than their non-cepstral counterparts.
- The Itakura distance [11] is a less effective measure of spectral distance than the Euclidean distance.
- Six Mel frequency coefficients capture most of the relevant information in a speech signal, although the importance of higher order coefficients differs with different speakers.

L. Lamel et al [6] combined an adaptive level equalizer, a pulse detector, and an endpoint ordering system to form a hybrid word boundary detector as shown below.

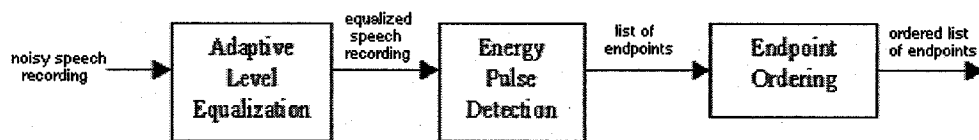


Figure 2.12 the hybrid approach endpoint detector proposed by Lamel et al

The adaptive equalizer produced an magnitude array from the recording such that the array fluctuated around zero when speech was not present, and was much larger in the presence of speech. This allows for the pulse detection stage to use absolute threshold values to detect the presence of speech. The output of the detector was a set of possible endpoints listed in order of their likelihood of being the true endpoints of the spoken utterance. This ordered list of endpoints was given to the recognizer, which found the endpoint pair representing a segment with the smallest distance from its set of patterns. If the distance obtained from one endpoint pair was sufficiently small, that

pair was taken as the true endpoint pair, otherwise the next endpoint pair was considered, and so forth. Lamel et al achieved a recognition rate of 95% with their top five endpoint pairs. They also concluded that the hybrid endpoint detector displayed a 10-15 % increase in recognition accuracy compared to a standard explicit endpoint detector.

L. Rabiner and M. Sambur [6] proposed an algorithm that used short-time magnitude to make an initial estimate of the endpoints and then used zero crossing rates to fine tune these endpoints. As the zero crossing rate was sensitive to the presence of weak fricatives, it allowed the computed boundaries to include weak fricatives which the time magnitude plot alone would normally miss. Using ten speakers uttering the numbers 0 to 9, they found that their algorithm committed no errors.

Gin-Der Wu and Chin-Teng Lin [1] proposed a word boundary detection algorithm which used the smoothed sum of time and frequency energies. This algorithm adaptively selected bands of a Mel-scale frequency bank based on the level of speech magnitude present. The algorithm reduced the recognition error rate due to incorrect boundary detection to around 20%.

M. Karnjanadecha and S. Zahorian [9] suggested varying the length of the analysis window based on its position in a recording. Shorter block lengths provided finer temporal resolution, and were thus beneficial towards the beginning and end of an utterance where the rate of spectral change was typically higher. Longer block lengths provided higher temporal smoothing. This proved useful towards the center of an utterance, which typically consisted of a slowly changing vowel region. Having introduced their varying block length concept into their word boundary detection

algorithm (using discrete cosine transform coefficients), Karnjanadecha and Zahorian's word boundary detection algorithm achieved an accuracy of 97.9%.

CHAPTER 3

TESTING TOOL

In accordance with Task 1 proposed in Chapter 2, we developed a testing tool to provide a quick and easy means of comparing and contrasting the performance of several word boundary detection algorithms. The values of several factors can be varied to observe their effects on the outcome of these algorithms.

3.1 Overview

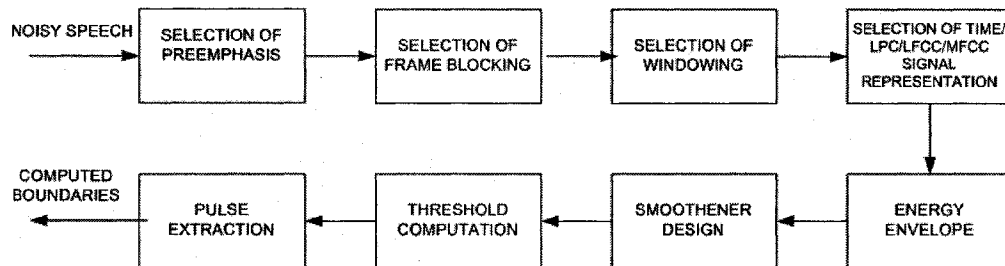


Figure 3.1 testing tool block diagram

As shown in Figure 3.1, the testing tool allows the user to select/design various parameters of a word boundary detection algorithm and observe their effects on algorithm performance. The parameters that can be changed by the user include preemphasis, frame blocking, windowing, signal representation and smoothener.

The various features provided by the testing tool are described in the following section.

3.2 Features

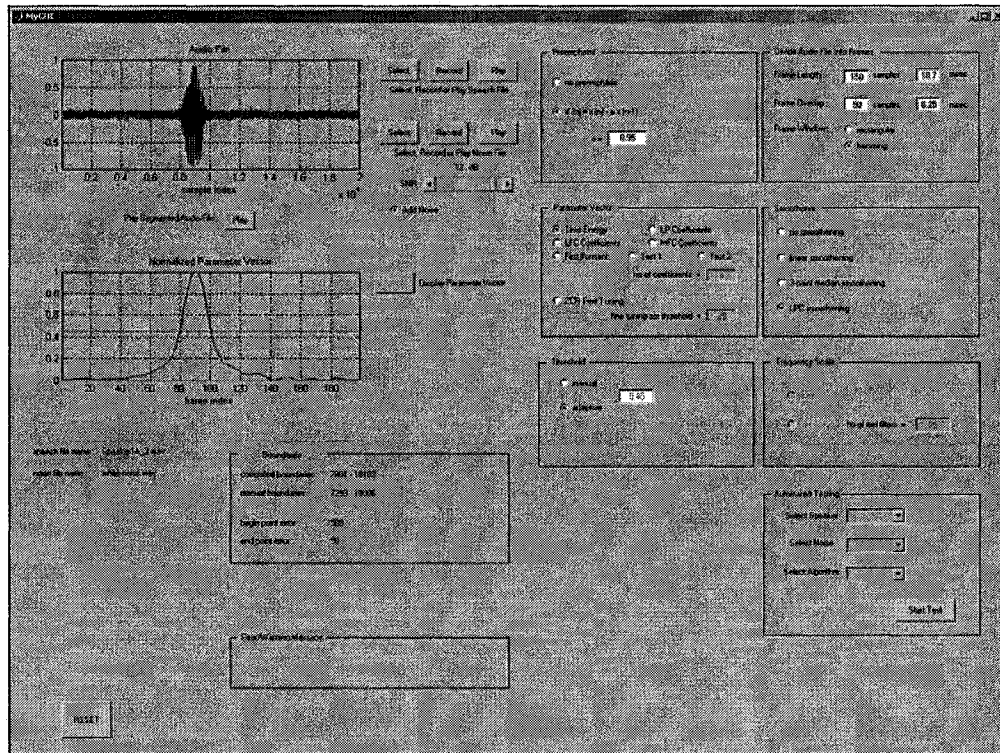


Figure 3.2 testing tool

As shown in Figure 3.2, the tool allows the user to select one of several clean speech recordings superimposed with one of several noise recordings. The user can vary the SNR of the resulting noisy speech recording from 20dB to -30dB. The resulting noisy speech signal can be applied as input to a word boundary detection algorithm that the user designs by essentially selecting several design parameters provided by the tool. The functions provided by the testing tool are described below:

Preemphasis

In a speech recording, there are certain sources of noise that have spectral properties that are very distinct from those of any speech sound we may be interested in. It may

be advantageous to remove such unwanted noise before any further processing. The preemphasizer is a high pass filter that removes low frequency noise present in the recording. The preemphasizer used by the tool can be represented by the transfer function

$$H(z) = 1 - az^{-1}$$

Figure 3.3 shows the frequency response of the preemphasizer.

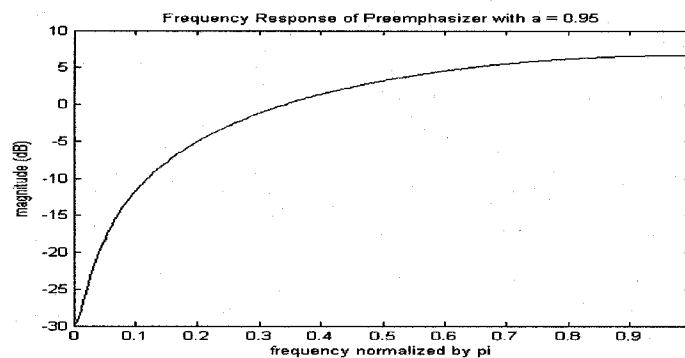


Figure 3.3 frequency response of preemphasizer

As shown in Figure 3.4, the tool allows the user to vary the value of the constant a to vary the amount of preemphasis applied to the noisy speech recording. A value of $0.95 < a < 0.97$ [3] is the value most commonly used by researchers.

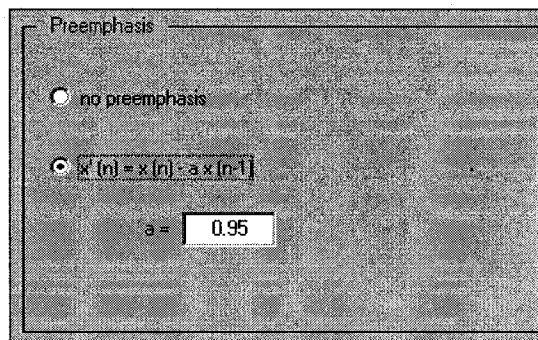


Figure 3.4 selection of preemphasizer

The benefit of using such a preemphasizer to eliminate background noise can be seen by preemphasizing a noisy recording of the utterance 'zero' as shown in Figure 3.5.

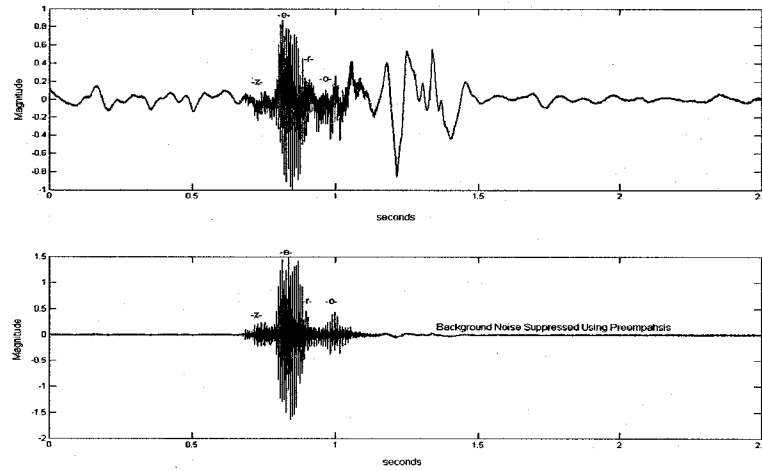


Figure 3.5 suppressing background noise using preemphasis

In spite of the benefits of using a preemphasizer, the user may refrain from using preemphasis to save on computation time.

Framing

In an automatic speech recognition system, the input speech recording is temporally divided into short speech segments, called frames, and the spectral or temporal properties of each frame are computed. Figure 3.6 demonstrates how a speech recording is divided into overlapping frames.

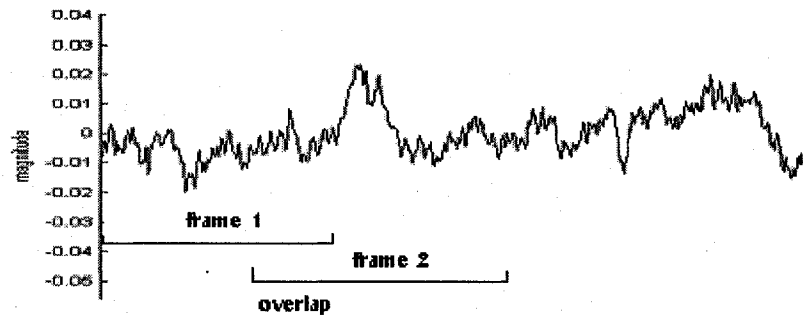


Figure 3.6 division of a speech segment into overlapping frames

As shown in Figure 3.7, the tool allows the user to choose and vary the length of each frame as well as the degree of overlap between frames.

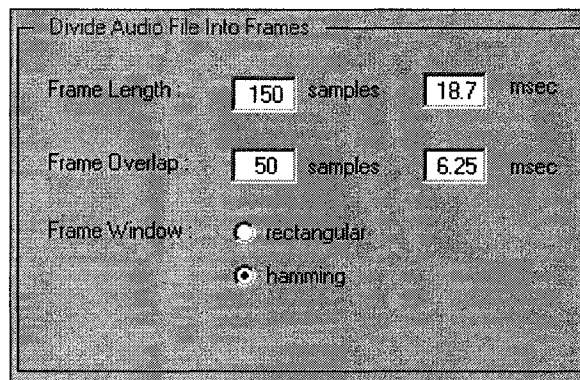


Figure 3.7 selection of frame length/overlap/window

Shorter frame lengths provide finer temporal resolution, and may prove beneficial where the rate of spectral change is high, whereas longer frame lengths provide higher temporal smoothing, which may prove useful in regions of slower spectral change. Another factor a user may consider is computational load. Smaller frame lengths with higher overlap would result in a larger number of frames per recording and hence a higher computational load on the word boundary detection algorithm.

The tool also allows the user to window each frame using a Hamming window given by

$$w(n) = 0.54 - 0.46 \cos\left(2\pi \frac{n}{N}\right) \\ 0 \leq n \leq N$$

P. Loizou and A. Spanias [10] and M. Karnjamadecha and S. Zahorian [9] are amongst many researchers that employ the Hamming window.

The effects of various combinations of frame length, overlap and windowing on an endpoint detection algorithm can be readily observed.

Speech Representation

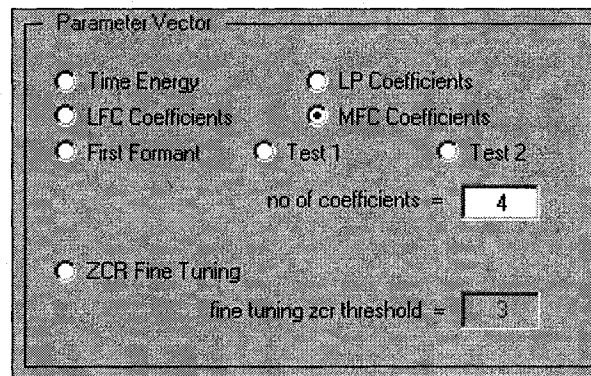


Figure 3.8 selection of speech representation

As shown in Figure 3.8, the user is able choose one of several representations of speech to use in his word boundary detection algorithm. The available choices are:

- Time Magnitude
- Frequency Magnitude

- Linear Prediction Coefficients
- Cepstral Coefficients

Details of each of the above representations have been explained in Chapter 2. The frame by frame representation of the noisy speech recording is called the magnitude envelope.

Smoothener

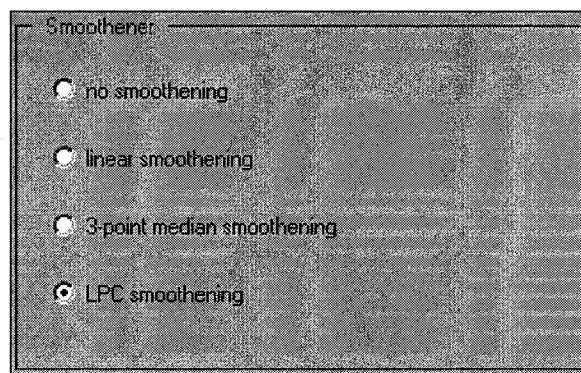


Figure 3.9 selection of smoothener

As shown in Figure 3.9, the user can choose one of three available smootheners to smoothen the magnitude envelope produced by the algorithm so that thresholds can be applied to separate speech from background noise. The choices of smootheners are:

- Linear Smoothener

This smoothener computes the average of every three consecutive samples of the magnitude envelope. Lamel et al [6] employ the use of this filter in their word boundary detection algorithm.

- Median Smoothener

This smoothener computes the median of every three consecutive samples of the magnitude envelope. Wu and Lin [1] employ a 3 point median smoothener in their algorithm.

- LPC Smoothener

This thesis introduces a new smoothener and claims that it is superior to the linear and median filters. This smoothener uses the properties of an LPC spectrum to smoothen the magnitude envelope in a manner more suitable for applying thresholds. Details of this smoothener are provided in Chapter 5.

Threshold

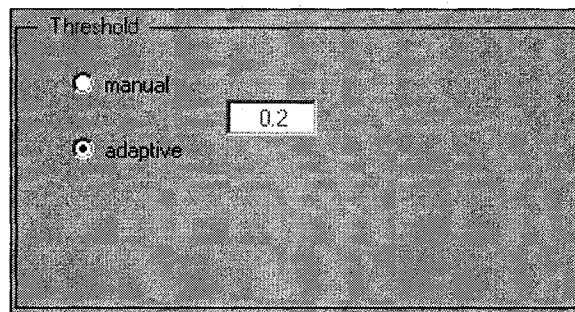


Figure 3.10 selection of threshold

As shown in Figure 3.10, the user can choose to explicitly specify a threshold level to be applied to the magnitude envelope, or he can allow the testing tool to adaptively compute the threshold for him. The manual selection allows the user to explicitly set the threshold value. The testing tool computes the threshold by evaluating the magnitude of noise in the initial frames of the test recording. This is because all test recordings are assumed to begin with background noise, which in turn is assumed to

remain constant throughout the test recording. Details of how the threshold is calculated by the testing tool are given in Section 4.8.

Frequency Scale

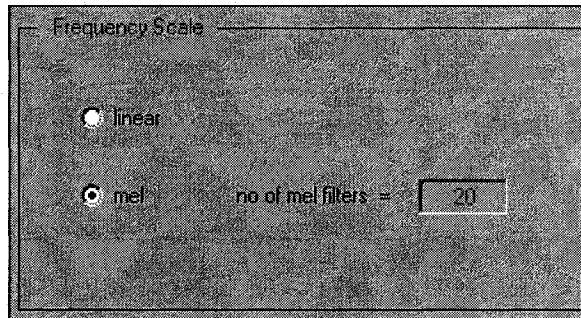


Figure 3.11 selection of frequency scale

As shown in Figure 3.11, the user can choose between using the regular frequency scale or the Mel scale to represent the noisy speech recording. The Mel scale is a scale of pitches which are perceived by the human ear to be equidistant from one another. The conversion of the linear frequency scale to the Mel scale is given in Figure 3.12.

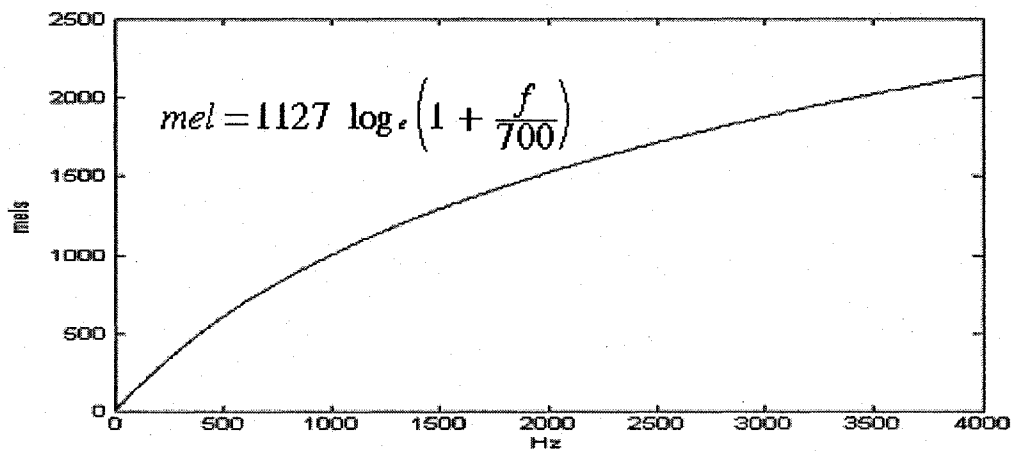


Figure 3.12 mel scale vs. frequency scale

Several researchers such as Davis and Mermelstein [11] and Wu and Lin [1] use Mel frequency cepstral coefficients in their proposed algorithms.

Speech Files

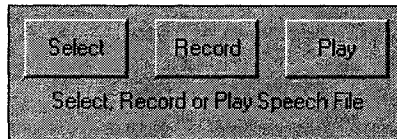


Figure 3.13 selection of speech file

As shown in Figure 3.13, the user can select from a number of prerecorded 'clean' speech files that include the utterances 'zero' to 'nine' from several different speakers. If the user chooses, he can also record his own 2.5 second (20,000 sample, 8KHz) speech file for testing.

Noise Files

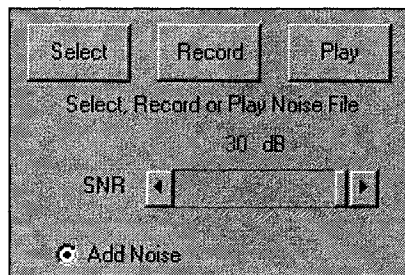


Figure 3.14 selection of noise file

As shown in Figure 3.14, the user can select from a number of prerecorded noise files to superimpose upon the selected speech files to form noisy speech files with which to test a word boundary detection algorithm. The SNR of these noisy speech samples can

also be varied from 20dB to -30dB. The user can also record his own 2.5 sec (20,000 samples at 8 KHz) noise file to superimpose upon any selected 'clean' speech file.

The SNR of the noisy speech recording is controlled by multiplying the noise file by a factor such that once it is superimposed upon the clean speech recording, the desired SNR between the manually computed boundaries is obtained.

$$factor = \sqrt{\frac{\frac{\text{speech power}}{\text{noise power}}}{\frac{SNR}{10 \cdot 10}}}$$

Segmented Audio File

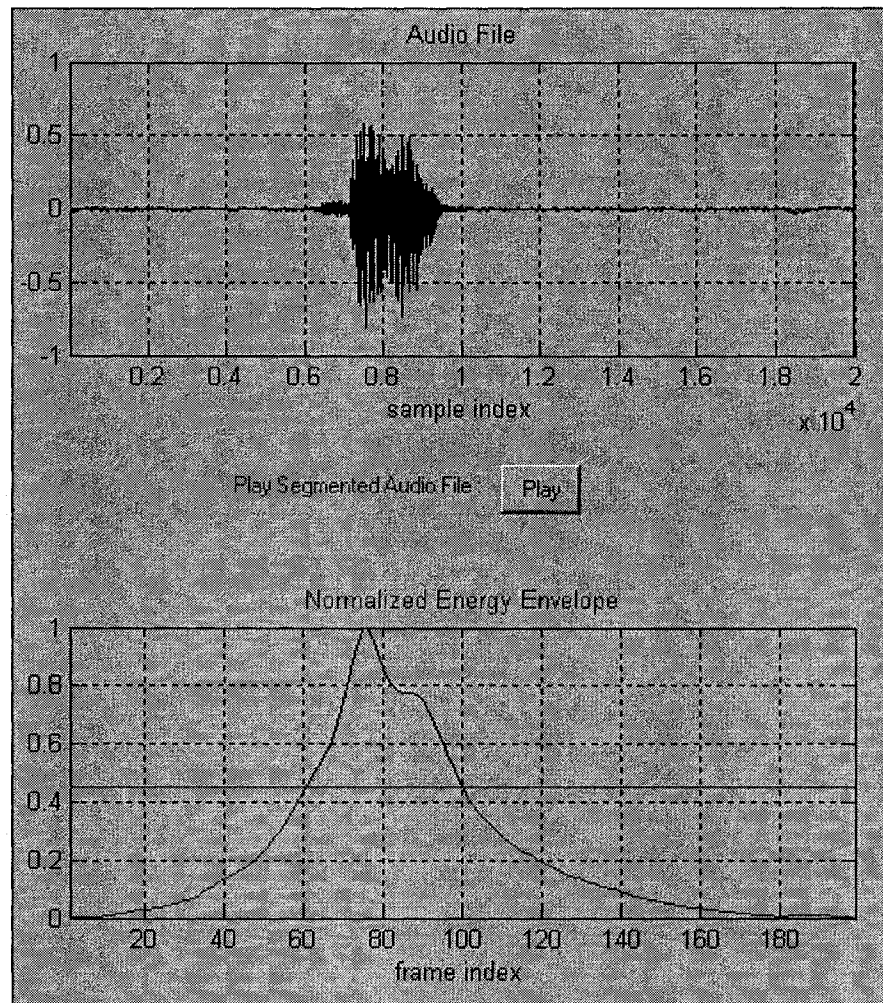
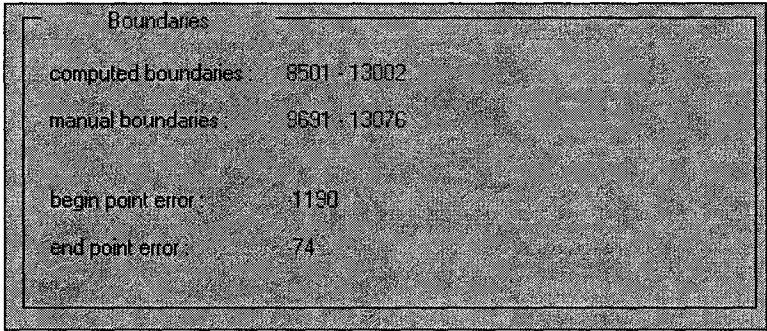


Figure 3.15 displaying segmented audio file

As shown in Figure 3.15, the tool allows the user to view the segmented noisy speech waveform where the region of speech is extracted from background noise. The normalized magnitude envelope is also displayed along with the computed or user specified threshold.

Displaying Results



Boundaries	
computed boundaries:	8501 - 13002
manual boundaries:	9691 - 13076
begin point error:	-1190
end point error:	74

Figure 3.16 displaying computed boundaries

As shown in Figure 3.16, the computed boundaries are displayed along with the manually derived boundaries. The tool also provides a measure of how close the computed boundaries are to the manually derived boundaries.

Automated Testing

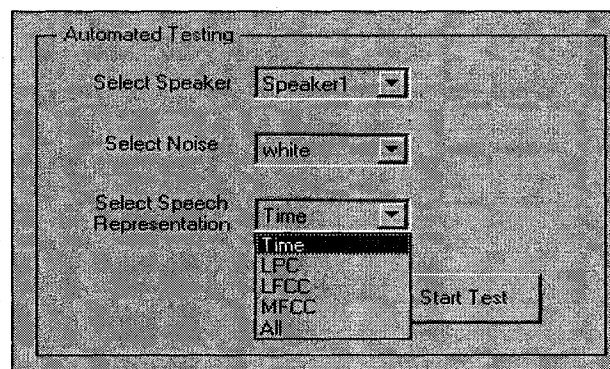


Figure 3.17 automated testing

As shown in Figure 3.17, the user can select several combinations of speakers, noise types and algorithms and run automated tests wherein the selected noise type is superimposed onto clean speech segments consisting of the selected speaker's utterances of the digits '0' to '9'. Signal to noise ratios of 20dB to -30dB are tested.

The results of this testing are the extracted segments of speech which are saved as WAV files into a separate folder.

3.3 Overview of Testing Tool Software

This section highlights the important functions that contribute to the software implementation of the testing tool. The source code for the software is included in folder named *TestingTool* on the CD located at the end of this document.

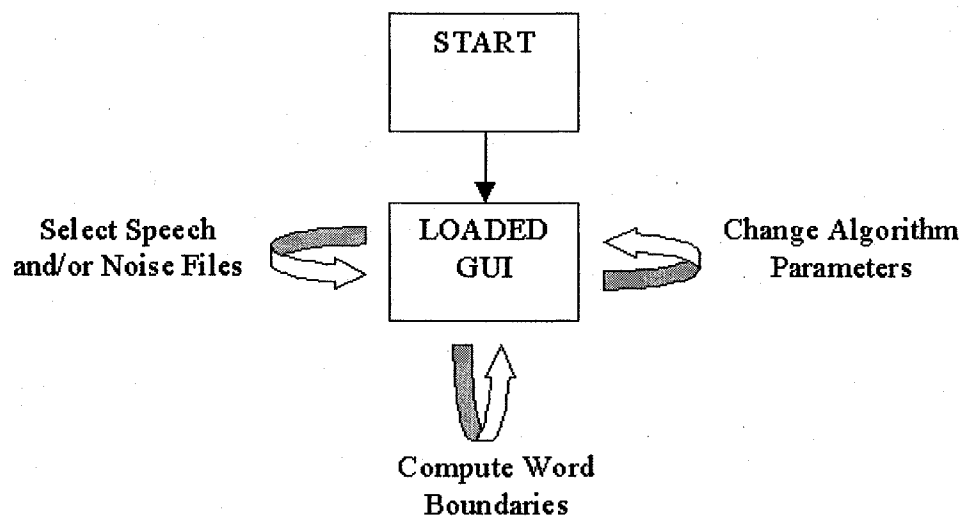


Figure 3.18 overview of testing tool software

The overview of our testing tool software is shown in Figure 3.18. Our testing tool is implemented as a graphical user interface (GUI) designed in Matlab. All of the operations of the GUI are executed by a series of 'callback' routines that are triggered upon user actions such as clicking a push button or selecting a radio button. This section gives a brief overview of the callback routines involved in the operation of the testing tool.

GetAudioFile

```
switch arg2
case 1 % Select
    AudioFileName=uigetfile('Speaker*.wav');
case 2 % Record
    xtime_clean=transpose(wavrecord(20000,sampling_rate));
    xtime=xtime_clean;
    AudioFileName=uiinputfile('temp','Save Recording');
    wavwrite(xtime,sampling_rate,AudioFileName);
case 3 % Play
    pause(0);
    wavplay(xtime,sampling_rate);
end
```

This callback allows the user to select one of several 2.5 sec speech files to be tested against his word boundary detection algorithm. The time waveform of the selected speech file is also displayed.

ChangeFrame

```
elseif strcmp(command_str,'Change_Frame')
frame_length_samples=str2num(get(h_FrameLengthSamples,'string'));
frame_length_msec=str2num(get(h_FrameLengthSeconds,'string'));
frame_overlap_samples=str2num(get(h_FrameOverlapSamples,'string'));
frame_overlap_msec=str2num(get(h_FrameOverlapSeconds,'string'));
switch arg2
case 1,
if frame_length_samples>=50 && frame_length_samples<=400
frame_length_msec=frame_length_samples/sampling_rate*1000;
set(h_FrameLengthSeconds,'string',num2str(frame_length_msec));
set(h_ErrorMessage,'string','');
else
set(h_ErrorMessage,'string','FRAME LENGTH MUST BE BETWEEN 50 AND 300 SAMPLES');
end
case 2;
if frame_length_msec>=5 && frame_length_msec<=40
frame_length_samples=floor(frame_length_msec*sampling_rate/1000);
set(h_FrameLengthSamples,'string',num2str(frame_length_samples));
set(h_ErrorMessage,'string','');
else
set(h_ErrorMessage,'string','FRAME LENGTH MUST BE BETWEEN 5 AND 40 MSEC');
end
case 3,
if frame_overlap_samples<frame_length_samples
frame_overlap_msec=frame_overlap_samples/sampling_rate*1000;
set(h_FrameOverlapSeconds,'string',num2str(frame_overlap_msec));
set(h_ErrorMessage,'string','');
else
set(h_ErrorMessage,'string','FRAME OVERLAP MUST BE LESS THAN FRAME LENGTH');
end
case 4;
if frame_overlap_msec<frame_length_msec
frame_overlap_samples=floor(frame_overlap_msec*sampling_rate/1000);
set(h_FrameOverlapSamples,'string',num2str(frame_overlap_samples));
set(h_ErrorMessage,'string','');
else
set(h_ErrorMessage,'string','FRAME OVERLAP MUST BE LESS THAN FRAME LENGTH');
end
case 5;
set(h_WindowHamming,'value',0);
if get(h_WindowRectangular,'value')==0
set(h_WindowRectangular,'value',1);
end
window=5;
case 6;
set(h_WindowRectangular,'value',0);
if get(h_WindowHamming,'value')==0
set(h_WindowHamming,'value',1);
end
window=6;
end
```

Through this callback, the user can vary the length and overlap of the frames into which the test recording is divided. He can also select/deselect a Hamming window used to window each frame.

ChangeSpeechRepresentation

```
switch arg2
case 1: % Time Energy
    set(h_freq_scale(3),'backgroundcolor',[0.8 0.8 0.8]);
    set(h_freq_scale,'enable','off','value',0);
    set(h_TimeEnergy,'value',1);
    set(h_NumCoefficients,'backgroundcolor',[0.8 0.8 0.8],'enable','off');
    vector_selection=1;
case 2: % LPC
    set(h_freq_scale(3),'backgroundcolor',[0.8 0.8 0.8]);
    set(h_freq_scale,'enable','off','value',0);
    set(h_LPC,'value',1);
    set(h_NumCoefficients,'backgroundcolor','white','enable','on');
    vector_selection=2;
case 3: % LFCC
    set(h_freq_scale(3),'backgroundcolor',[0.8 0.8 0.8]);
    set(h_freq_scale,'enable','off','value',0);
    set(h_LFCC,'value',1);
    set(h_NumCoefficients,'backgroundcolor','white','enable','on'); %No of coefficients
    vector_selection=3;
case 4: % HFCC
    set(h_freq_scale,'enable','on');
    set(h_freq_scale(3),'enable','off');
    set(h_freq_scale(2),'value',1);
    set(h_HFCC,'value',1);
    set(h_NumCoefficients,'backgroundcolor','white','enable','on');
    vector_selection=4;
case 5: % First Formant
    set(h_FrameOverlapSamples,'string',num2str(100));
    MyGUI('Change_Frame',3);
    set(h_FirstFormant,'value',1);
    set(h_NumCoefficients,'backgroundcolor',[0.8 0.8 0.8],'enable','off');
    vector_selection=5;
case 7:
    set(h_Test1,'value',1);
    set(h_NumCoefficients,'backgroundcolor',[0.8 0.8 0.8],'enable','off');
    vector_selection=7;
case 8:
    set(h_Test2,'value',1);
    set(h_NumCoefficients,'backgroundcolor',[0.8 0.8 0.8],'enable','off');
    vector_selection=6;
end
```

The user uses this callback to select which type of speech representation he wants for his word boundary detection algorithm. There are 4 different representations for the user to choose from. They are:

- Time Magnitude
- Linear Prediction Coefficients
- Linear Cepstral Coefficients
- Mel Cepstral Coefficients

ChangeMagnitudeEnvelopeFineTuning

```
if audio_file_select==1
    set(h_DisplayParameterVector,'enable','on'); % enable the 'display parameter vector' pushbutton
end
if get(h_ZCRFineTuning,'value')==0
    set(h_ZCRFineTuningThreshold,'enable','off','backgroundcolor',[0.8 0.8 0.8]);
else
    set(h_ZCRFineTuningThreshold,'enable','on','backgroundcolor','white');
end
```

This callback enables the user to select zero crossing rate based fine tuning for his boundary detection algorithm. Here the zero crossing rate around potential end points is employed to further fine tune their positions.

SmoothenMagnitudeEnvelope

```
switch arg2
    case 1;
        set(h_NoSmoothener,'value',1);
        smoothen_selection=1;
    case 2;
        set(h_LinearSmoothener,'value',1);
        smoothen_selection=2;
    case 3;
        set(h_MedianSmoothener,'value',1);
        smoothen_selection=3;
    case 4;
        set(h_LPCSmoothener,'value',1);
        smoothen_selection=4;
end
```

This callback allows the user to select which method of magnitude envelope smoothening he wishes to use. There are 3 different smootheners for the user to choose from. They are:

- Linear Smoothener
- Median Smoothener
- LPC Smoothener

DisplayMagnitudeEnvelope

```
axes(h_TestRecordingPlot); % the upper plot needs to be replotted
plot(xtime);
set(gca,'color',[0.8 0.8 0.8])
grid on;
xlim([1 length(xtime)]);
title('Audio File');
xlabel('sample index');

set(h_play_audio,'enable','on');
axes(h_ParameterVectorPlot);
plot(smoothened_vector);
set(gca,'color',[0.8 0.8 0.8])
xlim([1 no_of_frames]);
xlabel('frame index');
title('Normalized Energy Envelope','color','blue');
grid on;
set(h_DisplayParameterVector,'enable','off');
line([0 no_of_frames],[threshold1 threshold1],'color','red')
hold on;
x=temp1:temp2;
y=smoothened_vector(temp1:temp2);
plot(x,y,'color','black');
hold off;

axes(h_TestRecordingPlot);
hold on;
x=temp3:temp4;
y=xtime(temp3:temp4);
plot(x,y,'color','black');
hold off;

% Display boundaries and threshold
set(h_threshold(3),'string',num2str(threshold1));
set(h_ComputedBoundaries,'string',temp_string1);
temp_string2=[ num2str(ManualBegin) ' - ' num2str(ManualEnd) ];
set(h_ManualBoundaries,'string',temp_string2);
set(h_BeginPointError,'string',num2str(temp3-ManualBegin));
set(h_EndPointError,'string',num2str(temp4-ManualEnd));
```

This callback performs all the algorithm computation based on the user's selection of preemphasis, framing, speech representation, smoothener and threshold to compute the word boundaries of the test recordings. The callback also displays the segmented speech recording wherein the speech segment of the recording is extracted from the background noise.

ChangeThreshold

```
switch arg2
    case 1;
        set(h_threshold(1),'value',1);
    case 2;
        set(h_threshold(2),'value',1);
        set(h_threshold(3),'enable','off');
    case 3;
        set(h_threshold(1),'value',1);
        threshold=str2num(get(h_threshold(3),'string'));
end
```

The user uses this callback to either explicitly set his own threshold value or call on the testing tool to adaptively set the threshold for him. The threshold is ultimately used to extract speech from background noise of the test recording.

PlaySegmentedAudioFile

```
if arg2==1
    wavplay(xtime,sampling_rate);
else
    x_temp=[];
    [rows columns]=size(boundaries);
    if strcmp(boundaries,'No Endpoints Found')==0
        for i=1:columns
            x_temp=cat(2,x_temp,xtime_clean(boundaries(3,i):boundaries(4,i))); %if there are several p
        end
    end
    x=cat(2, zeros(1,(20000-length(x_temp))/2), x_temp, zeros(1,(20000-length(x_temp))/2) );
    if automation==0
        wavplay(x,sampling_rate);
    else
        if strcmp(boundaries,'No Endpoints Found')==1
            x=zeros(1,15000); %This makes a 30KB file.
        else
            if (boundaries(3,1)<ManualBegin-100) | (boundaries(4,columns)>ManualEnd+100) %the boundaries
                x=zeros(1,10000); %This makes a 20KB file.
            end
        end
    end
end
```

Once the testing tool has extracted the speech segment from the background noise of the test recording, the user can use this callback to listen to that extracted segment of speech.

Record

```
xtime=transpose(wavrecord(10000,str2num(get(h_edit_sampling_rate,'string'))));
MyGUI('Get_Audio_File',2);
axes(h_TestRecordingPlot);
plot(xtime);
title('Audio File');
xlabel('sample index');
grid on;
box on;
set(h_DisplayParameterVector,'enable','on');
audio_file_select=1;
```

The user employs this callback to record his own 2.5 sec speech files to be tested against his word boundary detection algorithm.

Preemphasis

```
set(h_preemphasis,'value',0);
switch arg2
    case 1
        set(h_preemphasis(1),'value',1);
        set(h_preemphasis(3),'backgroundcolor',[0.8 0.8 0.8],'enable','off');
    case 2
        set(h_preemphasis(2),'value',1);
        set(h_preemphasis(3),'backgroundcolor','white','enable','on');
    case 3
        set(h_preemphasis(2),'value',1);
end
if audio_file_select==1
    set(h_DisplayParameterVector,'enable','on');
end
```

This callback allows the user to select/deselect the preemphasizer used by the testing tool.

ChangeFrequencyScale

```
set(h_freq_scale,'value',0);
switch arg2
    case 1
        set(h_freq_scale(1),'value',1);
        set(h_freq_scale(3),'backgroundcolor',[0.8 0.8 0.8],'enable','on');
    case 2
        set(h_freq_scale(2),'value',1);
        set(h_freq_scale(3),'backgroundcolor','white','enable','off');
    case 4
        set(h_freq_scale(4),'value',1);
        set(h_freq_scale(3),'backgroundcolor',[0.8 0.8 0.8],'enable','off');
end
if audio_file_select==1
    set(h_DisplayParameterVector,'enable','on');
end
```

This callback allows the user to select either a linear frequency scale or a Mel frequency scale to use for the speech representation used in the boundary detection algorithm. In the case of the Mel frequency scale being selected, the user can also specify the number of filters used in the Mel frequency bank

ChangeNoCoefficients

```
set(h_DisplayParameterVector,'enable','on');
```

If the user chooses to use Linear Prediction coefficients or Cepstral coefficients as his speech representation, he can use this callback to specify the number of coefficients to be employed.

GetNoiseFile

```
set(h_noise_file_name_text,'visible','on');
set(h_noise_file_name,'visible','on','string',noise_file_name);
noise_file_select=1;
if audio_file_select==1
    xtime=wavread(AudioFileName)';
    xtime=xtime+noise(1:length(xtime));
    axes(h_TestRecordingPlot);
    plot(xtime);
    grid on;
    axis on;
    title('Noisy Speech');
    xlabel('sample index');
    set(gca,'color',[0.8 0.8 0.8]);
    set(h_DisplayParameterVector,'enable','on');
else
    axes(h_TestRecordingPlot);
    plot(noise);
    grid on;
    axis on;
    title('Noise Recording');
    xlabel('sample index');
    set(gca,'color',[0.8 0.8 0.8]);
end
end
```

The user can select one of several noise files to superimpose onto his selected speech file to form a noisy speech file to test against his word boundary detection algorithm.

SNR

```
SNR=get(h_snr,'value');
set(h_noise(2),'string',num2str(SNR));
noise=wavread(noise_file_name)';
xtime=wavread(AudioFileName)';
noise=get_noise(noise(1:length(xtime)),SNR,xtime);
xtime=wavread(AudioFileName)'+noise;
if automation==0
    axes(h_TestRecordingPlot);
    plot(xtime);
    grid on;
    axis on;
    title('Noisy Speech');
    set(gca,'color',[0.8 0.8 0.8]);
    set(h_DisplayParameterVector,'enable','on');
end
end
```

This callback allows the user to vary the signal to noise ratio (SNR) of a noisy speech record by superimposing a selected noise file onto a selected speech file. The SNR can be varied from 20dB to -30dB.

Automation

```
for snrNum = 20:-1:-30
    snr = num2str(snrNum);
    output_file = cat( 2 , snr , 'dB_', NoiseType , '_', ParamVector , '_', person , '_', letter , '_', number);
    set(h_snr,'value',str2num(snr));
    MyGUI('SNR',1);
    MyGUI('Display_Parameter_Vector');
    MyGUI('Play_Segmented_Audio_File',2);
```

This callback allows the user to perform automated tests on several combinations of speech files, noise files and boundary detection algorithms.

CHAPTER 4

OUR PROPOSED ALGORITHM

This chapter presents our proposed word boundary detection algorithm and explains the motivation behind several of our chosen design parameters.

Figure 4.1 displays a block diagram of our proposed algorithm.

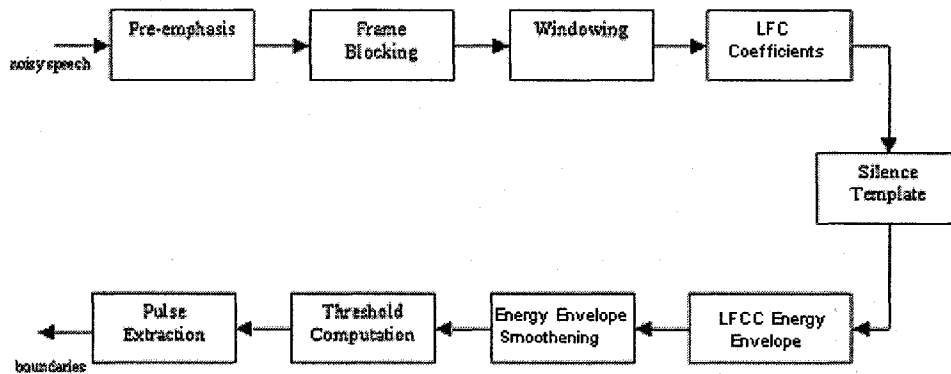


Figure 4.1 our proposed boundary detection algorithm

4.1 Preemphasis

The noisy speech signal is first filtered using a first order high pass filter. The prephasizer is defined by:

$$H(z) = 1 - az^{-1}$$

This eliminates unwanted low frequency background noise. The constant a is typically chosen from within $0.9 < a < 1$ [3].

Figure 4.2 shows the preemphasis of a noisy utterance of the word 'zero'

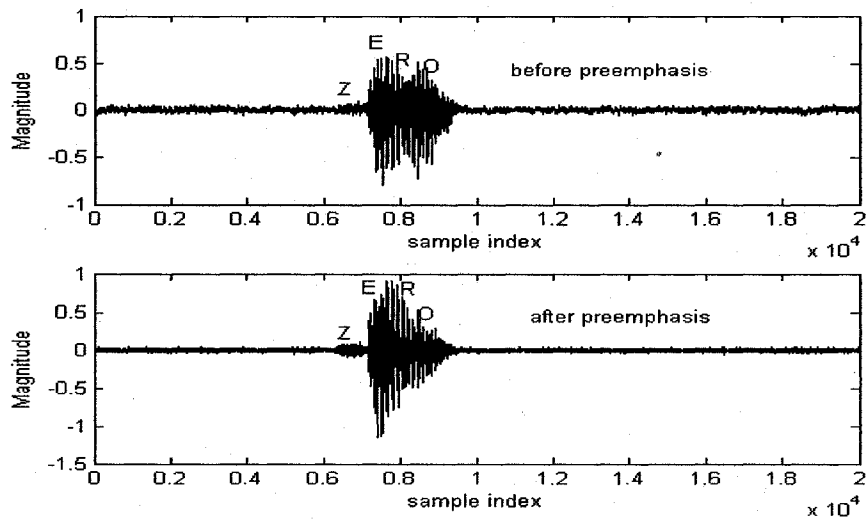


Figure 4.2 preemphasis of the noisy utterance 'zero'

4.2 Frame Blocking

The preemphasized noisy speech segment is then divided into overlapping frames. The length of each frame is 150 samples and the overlap is 50 samples. For a sampling rate of 8000 Hz, these values correspond to approximately 19ms and 6ms respectively. Greater values of overlap result in higher temporal granularity at the cost of computational load. The values of frame length and frame overlap used were chosen to provide a good balance between the two.

The first five frames of the noisy utterance 'zero' are shown in Figure 4.3.

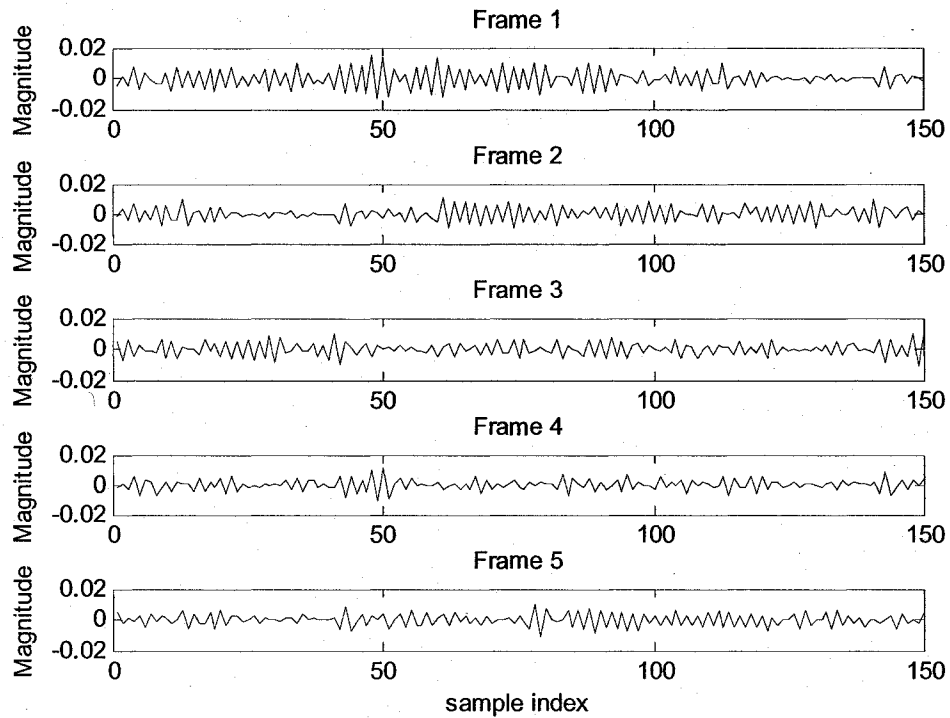


Figure 4.3 framing of a noisy speech recording

4.3 Windowing

Each frame is multiplied by a Hamming window given by

$$w(k+1) = 0.54 - \cos\left(2\pi \frac{k}{n-1}\right)$$

where $k = 0, \dots, n-1$

A Hamming window was chosen as it was the most common window used by previous researchers [9].

Figure 4.4 displays the first four frames of the noisy utterance 'zero' after they are windowed with a Hamming window.

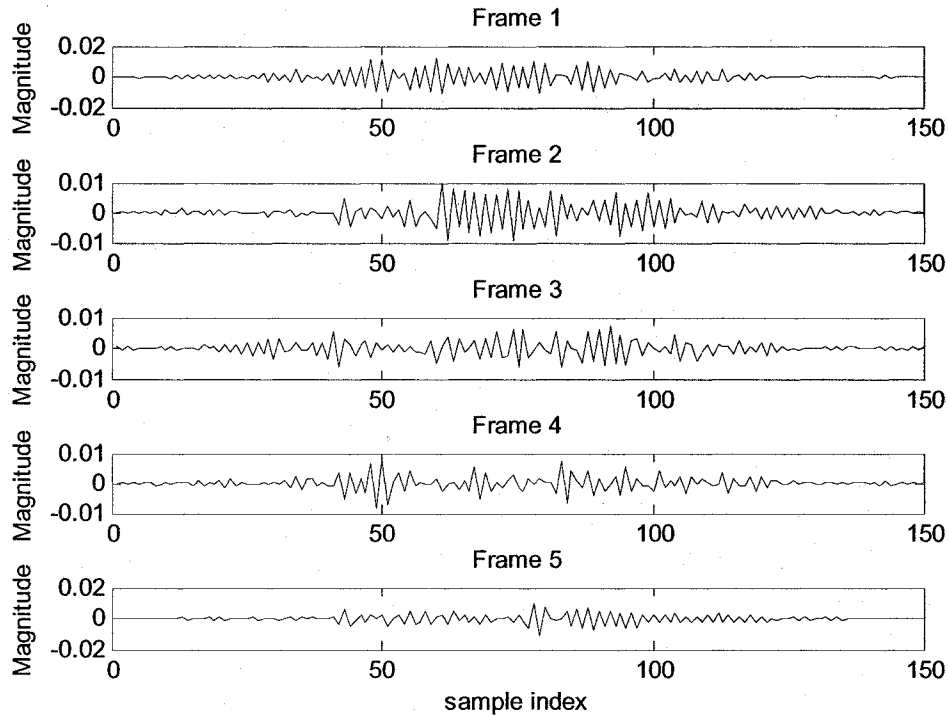


Figure 4.4 windowed frames

4.4 Cepstral Coefficients

Several researchers such as Davis and Mermelstein [11] and Wu and Lin [1] have published results proving the superiority of Cepstral Coefficients for use in word boundary detection. Our own tests corroborate these findings. Hence we have chosen Linear Frequency Cepstral (LFC) coefficients in our algorithm to distinguish speech from background noise. Four linear frequency cepstral (LFC) coefficients are computed for every frame. The number of coefficients was limited to four as it was

observed that a greater number of coefficients did not improve algorithm performance. Each frame of 20 elements is converted into a 4 element frame. The LFC coefficients are computed from each frame using the pseudo code.

```
LFC coefficients= real ( ifft ( log ( abs ( fft ( frame ) ) ) ) );
```

As depicted by the pseudo code, the magnitude of the Fourier Transform is taken, and its log computed. The real inverse Fourier Transform of the resultant sequence constitutes the real cepstrum of the original sequence.

The first 4 cepstral coefficients of each frame are used.

Figure 4.5 shows the first 4 coefficients of the first five frames of the noisy speech recording 'zero'.

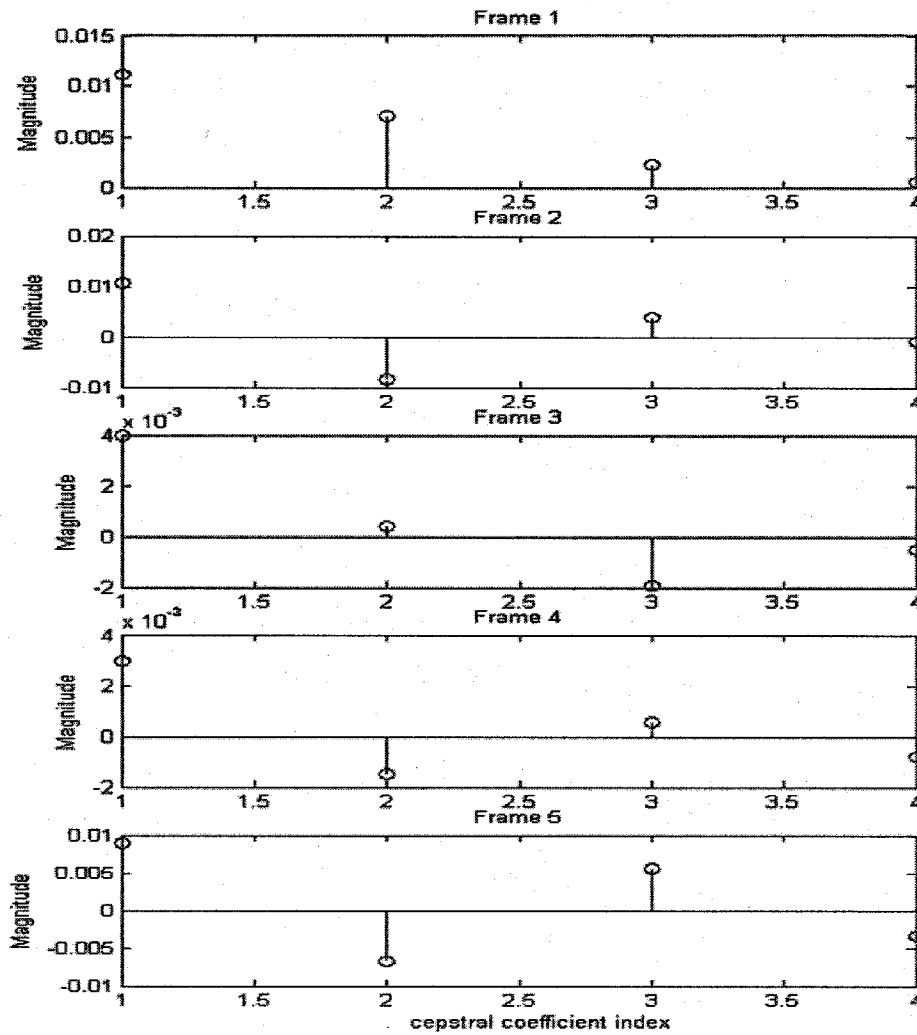


Figure 4.5 LFC coefficients of first five frames

4.5 Silence Template

The first LFCC frame is chosen as the silence template to which all further LFCC frames are compared. The first frame is chosen because it is assumed that all test recordings begin with silence or background noise. Thus the silence template defines the background noise.

Figure 4.6 shows the silence coefficient template for the noisy utterance 'zero'.

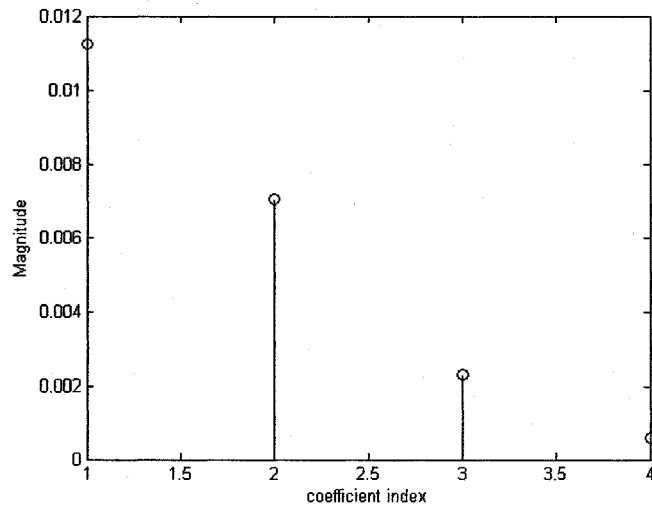


Figure 4.6 silence template

4.6 LFCC Distance Computation

The Euclidean distance between the LFC coefficient set of each frame and the silence template is computed to form the LFCC magnitude envelope. Thus, the number of elements in the magnitude envelope is equal to the number of frames of the recording.

The Euclidean distance between two 4-element vectors is given by

$$\text{Euclidean Dist} = \sqrt{\sum_{i=1}^4 (x_i - s_i)^2}$$

where x is the coefficient set for a frame and s is the silence template. The Euclidean distance was chosen over the Itakura distance as it was found to be more successful in

indicating phonetic differences between spectra [11]. Figure 4.7 shows the LFCC magnitude envelope for the noisy utterance of 'zero'.

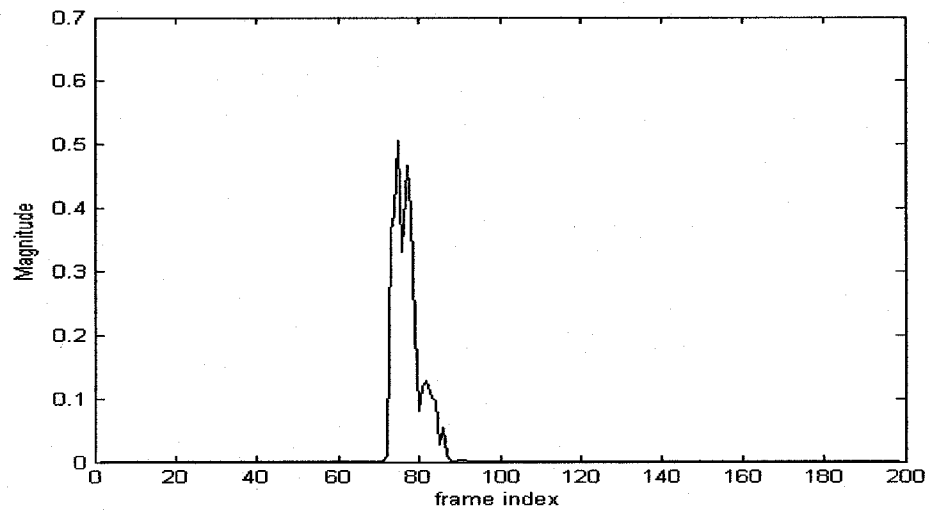


Figure 4.7 LFCC magnitude distance envelope

4.7 LFCC Magnitude Envelope Smoothing

An ideal smoothener takes a magnitude envelope, amplifies the peaks arising from speech, and flattens all other peaks. Our proposed linear prediction smoothener meets these two requirements due to its ability to follow the contour of a frequency spectrum as shown in Figure 4.8.

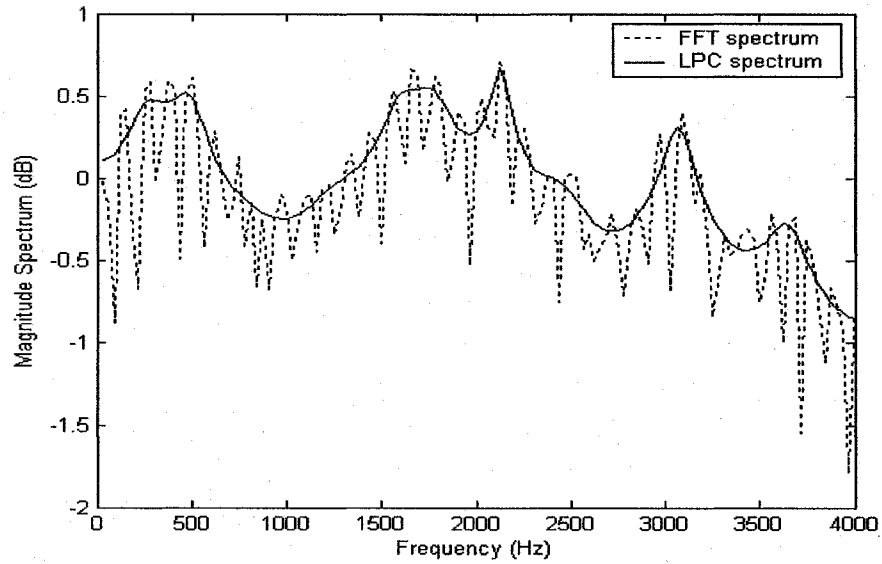


Figure 4.8 LPC and FFT spectra of a vowel phoneme

Our LP smoothener is applied to the LFCC magnitude envelope in three steps.

Step 1

The mirror image of the LFCC distance envelope is concatenated to its original form as shown in Figure 4.9.

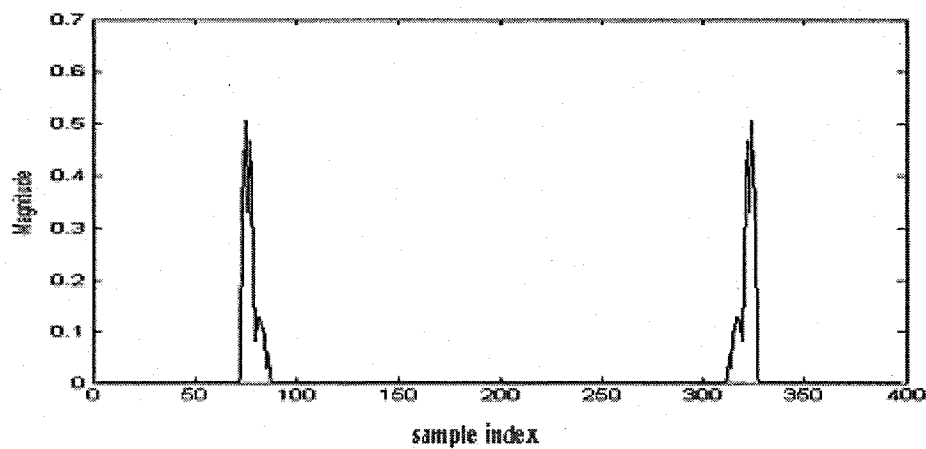


Figure 4.9 concatenated LFCC magnitude envelope

If we treat this concatenated distance envelope as the frequency spectrum of a discrete sequence, we can eventually obtain the smooth LPC spectrum of that sequence.

Step 2

The inverse FFT of the concatenated magnitude envelope is computed as shown in Figure 4.10.

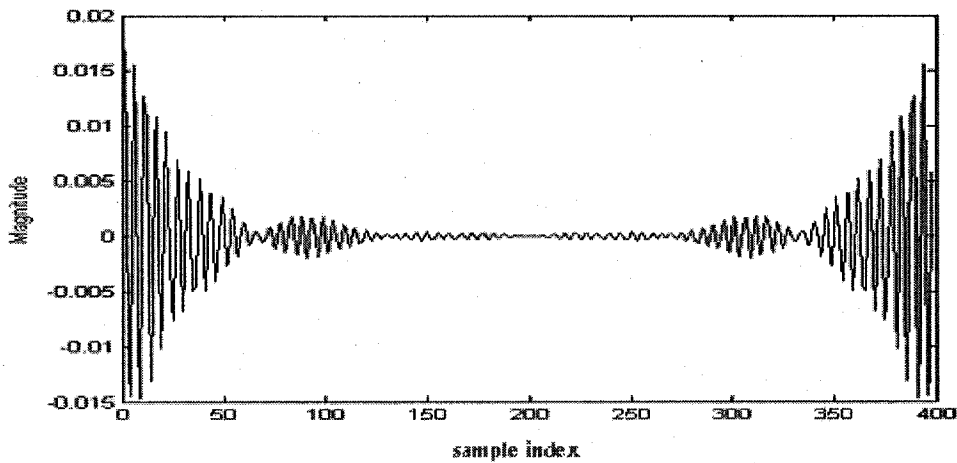


Figure 4.10 IFFT of the concatenated LFCC magnitude envelope

Step 3

The 12th order linear prediction spectrum of the resulting sequence is then computed and then normalized by its maximum value to produce the final output of the smoothener as shown in Figure 4.11.

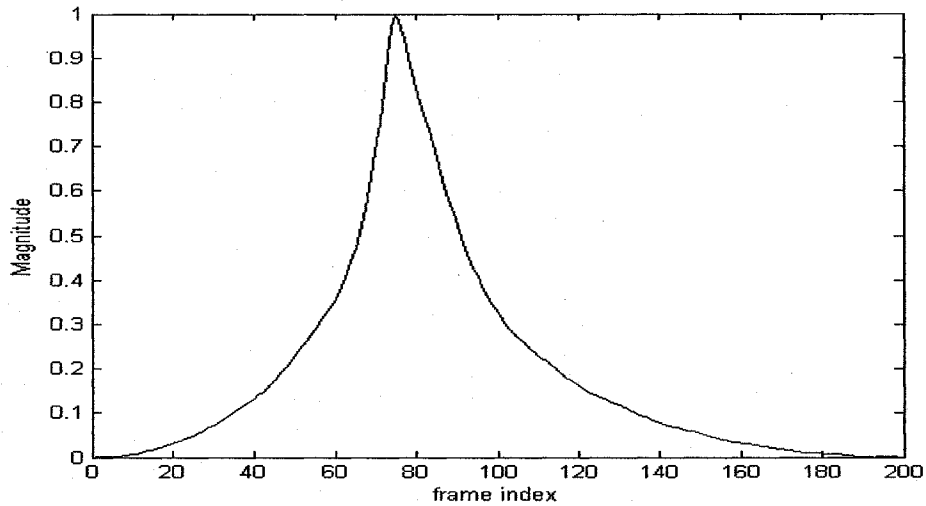


Figure 4.11 smoothener output

This version of the original magnitude envelope emphasizes the peak caused by speech over the other peaks, providing a much smoother magnitude envelope to apply thresholds against.

4.8 Threshold Computation

The threshold is computed from the first 20 values of the smoothened, normalized LFCC magnitude envelope 's' using the following equation:

$$threshold = 0.3 + \sum_{i=1}^{20} s(i)$$

The initial values of the magnitude envelope were chosen as each recording was assumed to begin with background noise which in turn was assumed to be constant throughout the test recording. The first 20 values corresponded to approximately

25ms of the test recording and were assumed to adequately represent the level of background noise. Thus, the threshold represented the magnitude of background noise. This threshold was deduced empirically such that the algorithm displayed the highest accuracy possible.

4.9 Pulse Extraction

The computed threshold is applied to the smoothed magnitude envelope to extract one or more boundary pairs. Only boundary pairs that are at least 5 frames (93.75 ms) wide and whose peaks are at least 0.1 units above the threshold are considered to be of speech. These values were empirically deduced. The boundaries thus obtained are the beginning and ending frame boundaries. The first sample of the beginning frame and the last sample of the ending frame are the final output of our word boundary detection algorithm. Figure 4.12 shows how the computed threshold is applied to extract a speech pulse.

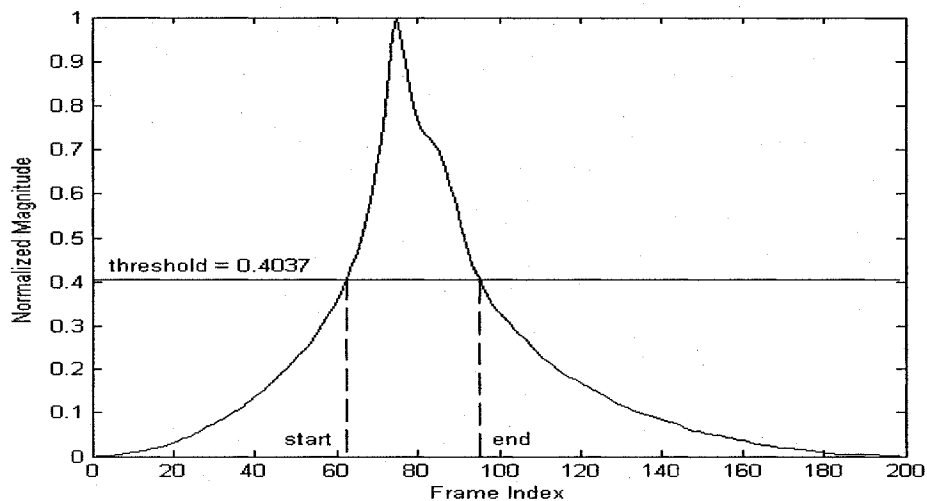


Figure 4.12 pulse extraction

An advantage of using the LPC smoothener over other smootheners is that the LPC smoothener tends to produce an magnitude envelope with only one peak per word (assuming the recorded speaker is speaking at a normal speed) as seen in Figure 4.13. In many cases, this precludes the added algorithm complexity of accommodating multiple magnitude pulses, wherein the algorithm has to decide whether the distinct peaks are of the same word or separate words. In the case of the LPC smoothener, each distinct pulse tends to be of a single word.

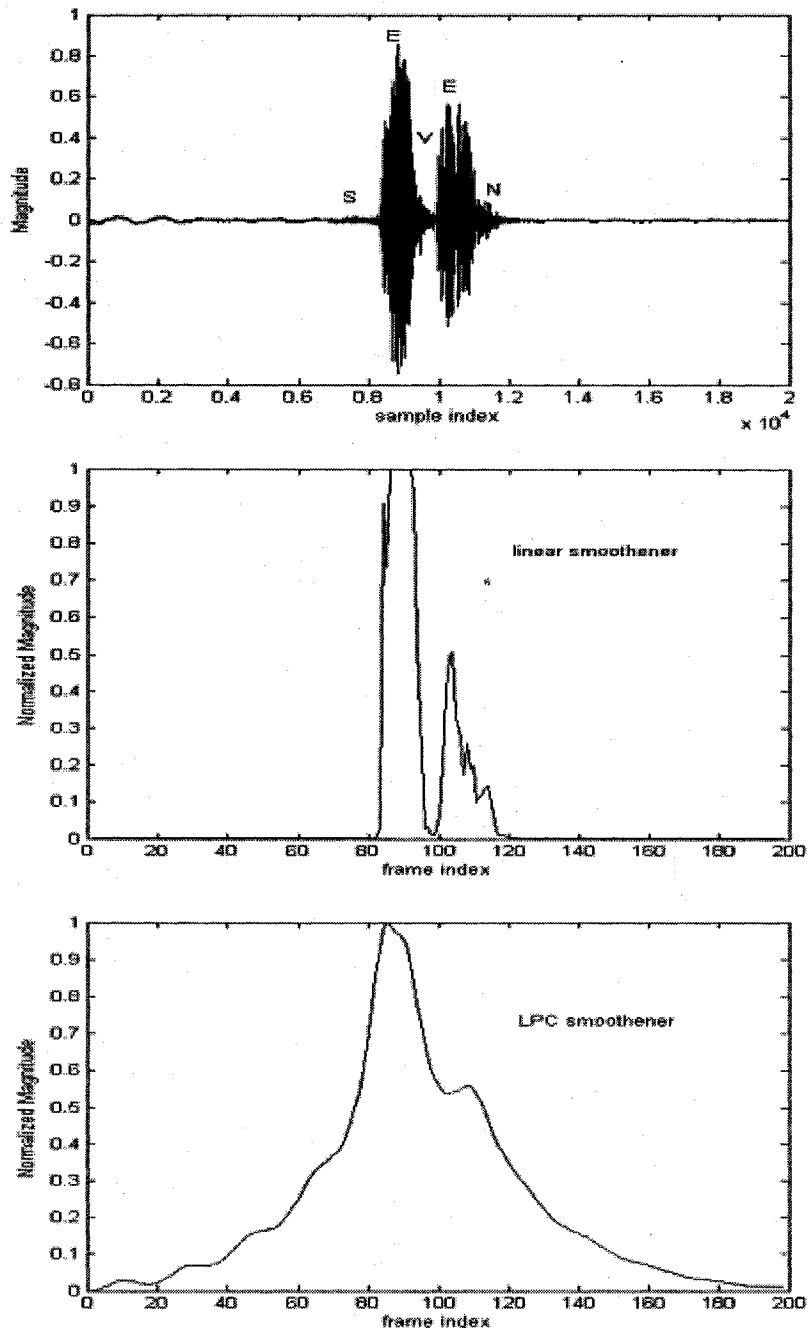


Figure 4.13 LPC smoother produces only one pulse per word.

But in cases where the smoothed distance envelope does contain more than one peak, the algorithm initially computes a boundary pair for each peak. However, if the end boundary of a boundary pair is less than 5 frames from the beginning boundary of

the following boundary pair, then the algorithm combines both boundary pairs into one. Such cases arise in the event of multiple words per recording. However, these cases are not within the scope of this thesis, as all the test recordings are of single words.

CHAPTER 5

TESTING

This chapter describes the testing performed to compare the performance of several word boundary detection algorithms.

5.1 Testing Overview

5.1.1 Baseline Testing

All 90 of the noise-free test recordings were randomly fed to the SAPI based word recognizer 10 times each and the accuracy of the recognizer was noted, as shown in Figure 5.1.

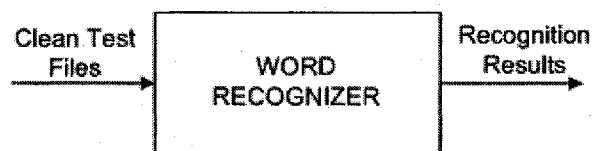


Figure 5.1 baseline testing

The recognizer was able to recognize all of the clean test files with 100% accuracy. Given this result, we assume that any recognition error made by the recognizer during the course of testing a boundary detection algorithm can be attributed to the algorithm and not the word recognizer.

5.1.2 Algorithm Performance Testing

Testing the various word boundary detection algorithms involved testing these algorithms against prerecorded noisy speech files and attempting to measure the accuracy of the computed boundaries using a word recognizer as shown in Figure 5.2.

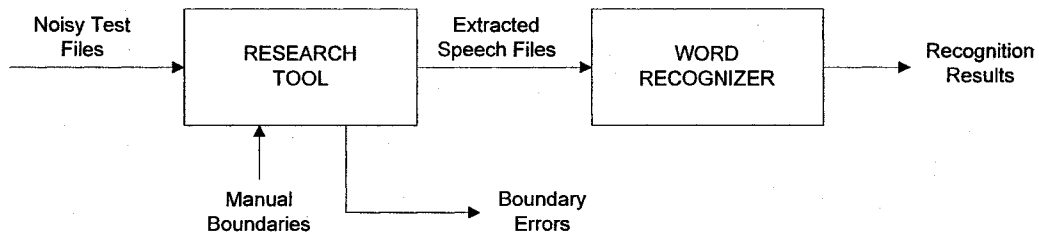


Figure 5.2 overview of test setup

The testing tool (as described in the Chapter 3) and the word recognition system were the main tools in this setup. The testing tool computed word boundaries of noisy speech segments and these boundaries were applied to the corresponding clean speech segments (the speech segments before superimposing noise). The word recognition system in turn attempted to recognize these speech segments. If the recognition system recognized the segmented clean speech correctly, then it was assumed that the boundaries were correctly computed. However, if the recognition system failed to correctly recognize the segmented clean speech, then the computed boundaries were deemed to be incorrect. A log was maintained depicting all the WAV files tested along with their recognition results (recognized or not recognized).

With this test setup, a correct recognition occurred if a boundary detector erroneously computed the boundaries to be the first and last samples of the test recording. This would incorrectly suggest high performance of the boundary detector. Hence a second

log was maintained depicting magnitudes of boundary errors committed by the boundary detection algorithms.

Details of the test setup are given in the following sections.

5.2 Test Setup

The work involved in our testing was divided into three stages.

5.2.1 Stage 1:

Several noise-free speech segments were recorded from 3 different speakers. The set of utterances consisted of the digits 0 to 9 . The 3 different speakers are described below:

	GENDER	AGE
SPEAKER1 :	Male	29
SPEAKER2 :	Male	30
SPEAKER3 :	Female	32

These 'clean' utterances were saved as WAV files with a sampling rate of 8 KHz and a file length of 20,000 samples (2.5 sec). Several noise files were obtained either by recording them or by downloading them from web sites. These sample noise files were also of a sampling rate of 8Kz and 20,000 samples long. These noise files

include wind noise, fan noise as well as several colors of noise. During the course of testing, these noise files were amplified and superimposed onto the 'clean' speech files to produce 'noisy' speech files. The degree of amplification was determined by the level of Signal to Noise Ratio (SNR) we desired. We employed 51 different SNR values from 20dB to -30dB in 1dB intervals.

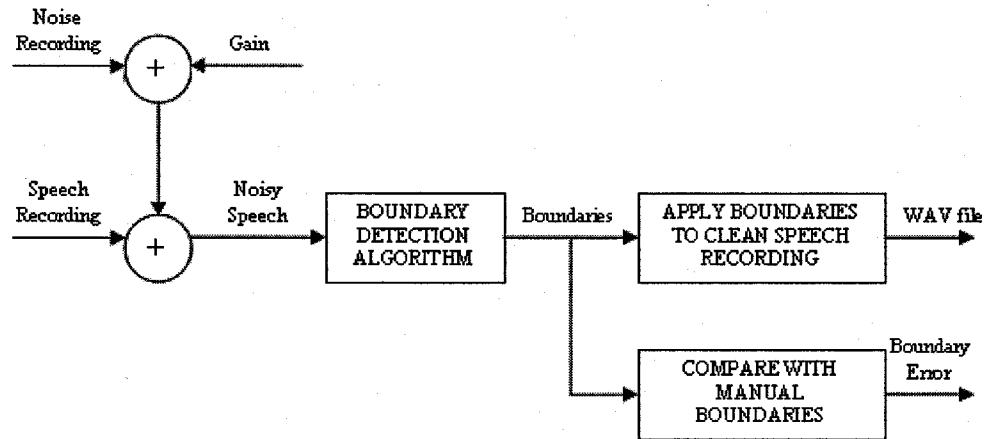


Figure 5.3 generation of test file

As shown in Figure 5.3, each WAV file thus created was fed as input to 4 different word boundary detection algorithms. Each algorithm extracted from each input file a pair of sample numbers as proposed endpoints of the noisy utterance. These computed endpoints were compared with manually deduced endpoints and the resulting boundary errors were logged into an Excel file. Furthermore, the contents of the original 'clean' utterance present between the above computed boundaries were saved into a separate WAV files. These WAV files were the final outputs of this stage. This complete stage was implemented in Matlab. The number of WAV files produced by 3 speakers uttering 10 numbers 3 times each, with 3 different types of superimposed

noise to form noisy speech recordings with 51 different SNR values, applied to 4 different algorithms = $3*10*3*3*51*4 = 55,080$ files.

Details of the Matlab software used to create these files are provided in Section 3.3

5.2.2 Stage 2:

The WAV files generated above were given to the second stage which employed a Microsoft SAPI (Speech Application Programming Interface) application. SAPI provides an interface for applications to use the Microsoft Speech Engine as shown in Figure 5.4.

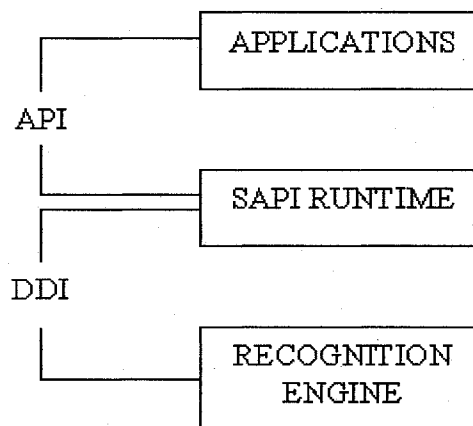


Figure 5.4 SAPI_overview

5.2.3 Stage 3:

The third stage involved recognition of all 55,080 files generated in stage 1. The speech recognition application attempted to recognize all of these WAV files in random order. The recognizer was developed in VC++ and employs SAPI to achieve

word recognition. Recognition results were logged into a six-dimensional matrix. The six dimensions were for speaker, utterance, attempt at utterance, SNR value, noise type, and algorithm. Once recognition attempts were made on all the files the matrix was exported to an Excel sheet, from which graphs were plotted displaying recognition accuracy against SNR values. It were these graphs along with the boundary errors logged in Stage 1 that provided insight into the performance of our word boundary detection algorithms under various levels of background noise.

Figure 5.5 presents an overview of the software developed to implement the recognizer. The source code for the software is included in folder named *Recognizer* on the CD located at the end of this document.

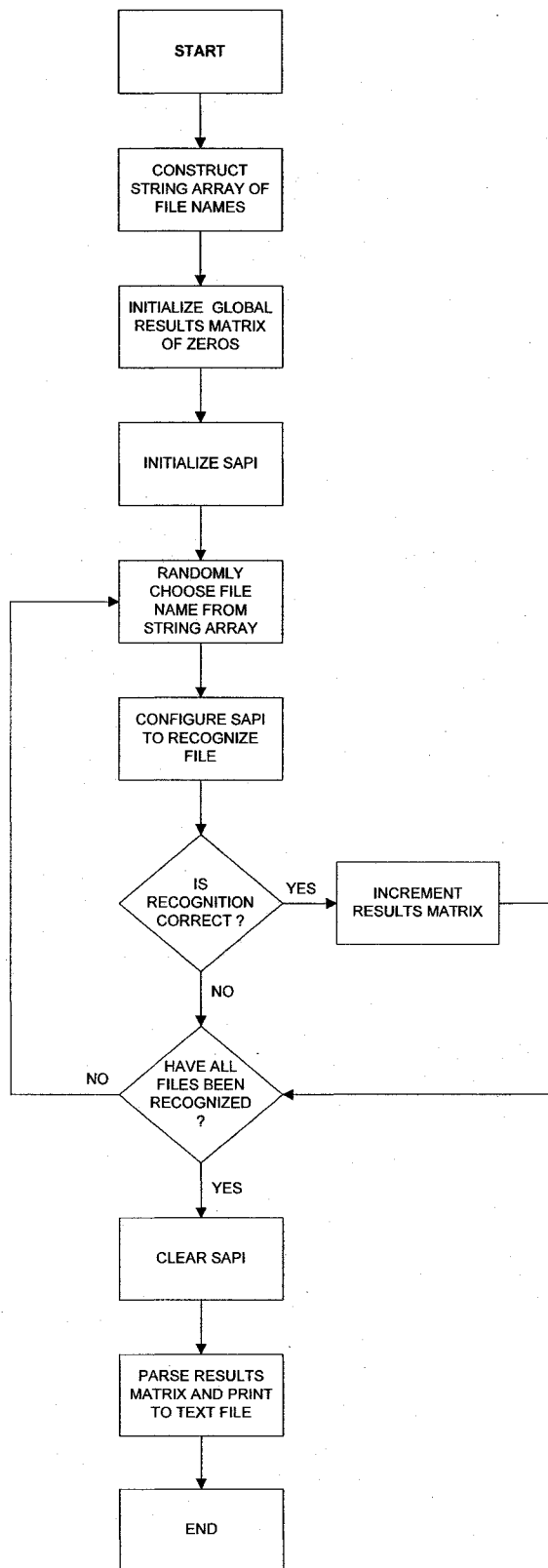


Figure 5.5 software for recognition of WAV files

Construct String Array of File Names

```
void CreateCell ()
{
    int fileindex=0;
    for (int isnr=0; isnr<MAXSNR; isnr++)
    {
        for (int inoise=0 ; inoise<MAXNOISETYPE ; inoise++)
        {
            for (int ialgorithm=0 ; ialgorithm<MAXALGORITHMS ; ialgorithm++)
            {
                for (int ispeaker=0; ispeaker<MAXSPEAKERS; ispeaker++)
                {
                    for (int iletter=0; iletter<MAXATTEMPTS; iletter++)
                    {
                        for (int iutterance=0 ; iutterance<MAXUTTERANCE ; iutterance++)
                        {
                            wchar_t sum[MAXFILENAMELENGTH];
                            wchar_t *sum1, *sum2, *sum3, *sum4, *sum5, *sum6;
                            sum1=snr[isnr];
                            sum2=Noise[inoise];
                            sum3=Algorithm[ialgorithm];
                            sum4=Speaker[ispeaker];
                            sum5=Letter[iletter];
                            sum6=Utterance[iutterance];
                            wcscpy(sum,sum1);
                            wcscat(sum,sum2);
                            wcscat(sum,sum3);
                            wcscat(sum,sum4);
                            wcscat(sum,sum5);
                            wcscat(sum,sum6);
                            for(int i=0;i<MAXFILENAMELENGTH;i++)
                            {
                                Files[fileindex][i] = sum[i];
                            }
                            fileindex++;
                            *sum = NULL;
                            sum1 = NULL;
                            sum2 = NULL;
                            sum3 = NULL;
                            sum4 = NULL;
                            sum5 = NULL;
                            sum6 = NULL;
                        }
                    }
                }
            }
        }
    }
}
```

Figure 5.6 construction of file name array

The snippet of code displayed in Figure 5.6 displays how a global string array is constructed to contain the names of all the WAV files we want to be recognized.

Initialize Global Results Matrix

A global multidimensional matrix is initialized with zeros. This 'Results' matrix is used to store recognition results. Each element of the 'Results' matrix corresponds to one file from the global string array. Every correct recognition of a file increments its corresponding matrix element by one.

Initialize SAPI

```
CComPtr<ISpStream> cpInputStream;  
CComPtr<ISpRecognizer> cpRecognizer;  
CComPtr<ISpRecoContext> cpRecoContext;  
CComPtr<ISpRecoGrammar> cpRecoGrammar;
```

Since SAPI is a COM (Component Object Model) based application, COM must be initialized to activate SAPI. The main interface SAPI provides for speech recognition is the *IspRecoContext* COM interface. It is from this interface that SAPI receives requests for speech recognition. The application must set up one of two available speech engines using SAPI's *IspRecoContext* interface. The engine used for our testing was the *InProc* speech recognition engine. The application also has to specify a grammar using SAPI's *IspRecoGrammar* interface. This grammar essentially dictates the type of utterances to be recognized, and is defined in an XML format. The XML file defining the grammar used in our testing limits recognition to one of ten possibilities, namely the digits 0 to 9. Figure 5.7 shows the XML file defining the grammar used in our testing.

```
<GRAMMAR LANGID="409">  
  <DEFINE>  
    <ID NAME="DIGIT" VAL="3"/>  
  </DEFINE>  
  <RULE NAME="DIGIT" TOPLEVEL="ACTIVE">  
    <L PROPNAME="DIGIT">  
      <P VALSTR="Zero">Zero</P>  
        <P VALSTR="One">One</P>  
        <P VALSTR="Two">Two</P>  
        <P VALSTR="Three">Three</P>  
        <P VALSTR="Four">Four</P>  
        <P VALSTR="Five">Five</P>  
        <P VALSTR="Six">Six</P>  
        <P VALSTR="Seven">Seven</P>  
        <P VALSTR="Eight">Eight</P>  
        <P VALSTR="Nine">Nine</P>  
    </L>  
  </RULE>  
</GRAMMAR>
```

Figure 5.7 grammar for recognition of digits 0 to 9

Randomly Choose File Names From String Array

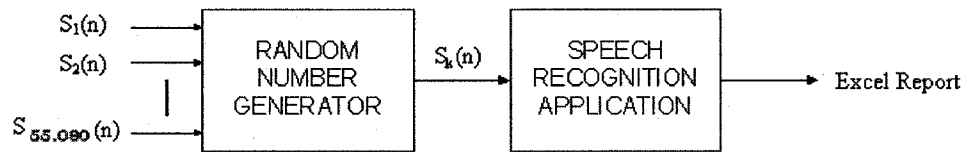


Figure 5.8 testing procedure

Figure 5.8 displays how WAV files are randomly chosen to be recognized by the recognizer to prevent any 'learning' by the recognizer.

Configure SAPI To Recognize WAV File

```
hr = SPBindToFile( file_name, SPFM_OPEN_READONLY, &cpInputStream, NULL, NULL);
hr=cpRecognizer.CoCreateInstance(CLSID_SpInprocRecognizer);
hr=cpRecognizer->SetInput(cpInputStream, FALSE);
hr=cpRecognizer->CreateRecoContext(&cpRecoContext);
hr=cpRecoContext->CreateGrammar(3, &cpRecoGrammar);
hr=cpRecoGrammar->LoadCmdFromFile(L"temp.xml", SPLO_STATIC);
hr=cpRecoContext->SetInterest(SPEI(SPEI_RECOGNITION), SPEI(SPEI_RECOGNITION));
hr=cpRecoGrammar->SetGrammarState(SPGS_ENABLED);
hr=cpRecoGrammar->SetRuleIdState(0, SPRS_ACTIVE);

BOOL fRecognition = FALSE;

while ( (!fRecognition) && S_OK == cpRecoContext->WaitForNotifyEvent(2700))
{
    CSpEvent spEvent;
    while (!fRecognition && S_OK == spEvent.GetFrom(cpRecoContext))
    {
        switch (spEvent.eEventId)
        {
            case SPEI_RECOGNITION:
                static const WCHAR wszUnrecognized[]=L"<Unrecognized>";
                dstrText.Append(L"");
                fRecognition=TRUE;
                break;
        }
    }

    if(wcsstr(file_name, dstrText) != NULL)
    {
        result=1;
    }
    spEvent.Clear();
}
```

The application has to set up events used by SAPI to communicate with the application and also specify which events it wants to be notified of. Examples of event notifications provided by SAPI are *SPEI_SOUND_START*, *1SPEI_SOUND_END*, *SPEI_SR_END_STREAM* and *SPEI_RECOGNITION*.

A SAPI stream object is set up with each WAV file to be recognized. An in-process speech recognition engine is configured and each WAV file is bound to the engine. A recognition context is configured to store the required SAPI events.

Increment Results Matrix

```
if(resultRecognized ==1)
{
    AddToMatrix(file_name);
}
```

Whenever a WAV file is correctly recognized, its corresponding element in the global 'Results' matrix is incremented by one. Incrementing does not occur if the recognizer makes an incorrect recognition. Thus, upon completion of testing, the value of each element denotes the total number of successful recognitions of its corresponding test WAV file. As the maximum possible value of each matrix element (corresponding to all successful recognitions) is known, the value of each element denotes the recognition accuracy of the tested algorithm for that particular test WAV file.

Parse Results Matrix And Print To Text File

```
void PrintText(){
FILE * pFile;
pFile = fopen("Results.txt" "w");
int ResultSet[MAXUTTERANCE][MAXSNR] = {0.0}; //For Utterance vs SNR
for (int ialgoritha=0 ; ialgoritha<MAXALGORITHMS ; ialgoritha++)
{
wchar_t algoName[30];
wscpy(algoName,L"Algoritha: ");
wscat(algoName,Algoritha[ialgoritha]);
for (int inoise=0 ; inoise<MAXNOISETYPE ; inoise++)
{
wchar_t noiseName[30];
wscpy(noiseName,L"Noise: ");
wscat(noiseName,Noise[inoise]);
for (int ispeaker=0; ispeaker<MAXSPEAKERS; ispeaker++)
{
wchar_t speakerName[30];
wscpy(speakerName,L"Speaker: ");
wscat(speakerName,Speaker[ispeaker]);
for (int iletter=0; iletter<MAXATTEMPTS; iletter++)
{
for (int isnr=0; isnr<MAXSNR; isnr++)
{
for (int utterance=0 ; utterance<MAXUTTERANCE ; utterance++)
{
ResultSet[utterance][isnr] += Results[utterance][isnr][ispeaker][iletter][ialgoritha][inoise];
}
}
}
}
}
}
//Print To File
if (pFile!=NULL)
{
fwrite(algoName,1,30,pFile);
fprintf(pFile,"\n");
fwrite(noiseName,1,30,pFile);
fprintf(pFile,"\n");
fwrite(speakerName,1,30,pFile);
fprintf(pFile,"\n");
for(int sLine=MAXSNR-1;sLine>=0;sLine--){
fprintf(pFile,"\t");
fwrite(snr[sLine],1,7,pFile);
}
fprintf(pFile,"\n");
for(int si=0;si<MAXUTTERANCE;si++){
fwrite(Utterance[si],1,7,pFile);
for(int sj=MAXSNR-1;sj>=0;sj--){
fprintf(pFile,"\t%d",ResultSet[si][sj]);
ResultSet[si][sj]=0;
}
fprintf(pFile,"\n");
}
fprintf(pFile,"\n\n\n");
}
}
}
fclose(pFile);
}
```

After the recognizer attempts to recognize all the files present in the global string array, the 'Results' matrix is parsed and the results are printed to a text file.

CHAPTER 6

RESULTS

This chapter presents the results of testing performed using the testing tool and SAPI driven recognizer configured in this thesis. We also present a comparison of the performances of the algorithms tested in this thesis.

6.1 Preliminary Results

Our testing tool allows the user to select or deselect the use of a preemphasizer and to select one of 3 available smootheners. All of the algorithms tested in this thesis have used the preemphasizer and an LPC smoothener provided by our testing tool.

Preemphasis

We employed preemphasis in all of our testing to remove low frequency background noise. Figure 6.1 shows the recognition accuracy obtained from testing a Mel Frequency Cepstral Coefficient (MFCC) algorithm with several noisy speech files. Testing was performed once without using preemphasis and once using preemphasis. The results shows the benefit of using preemphasis.

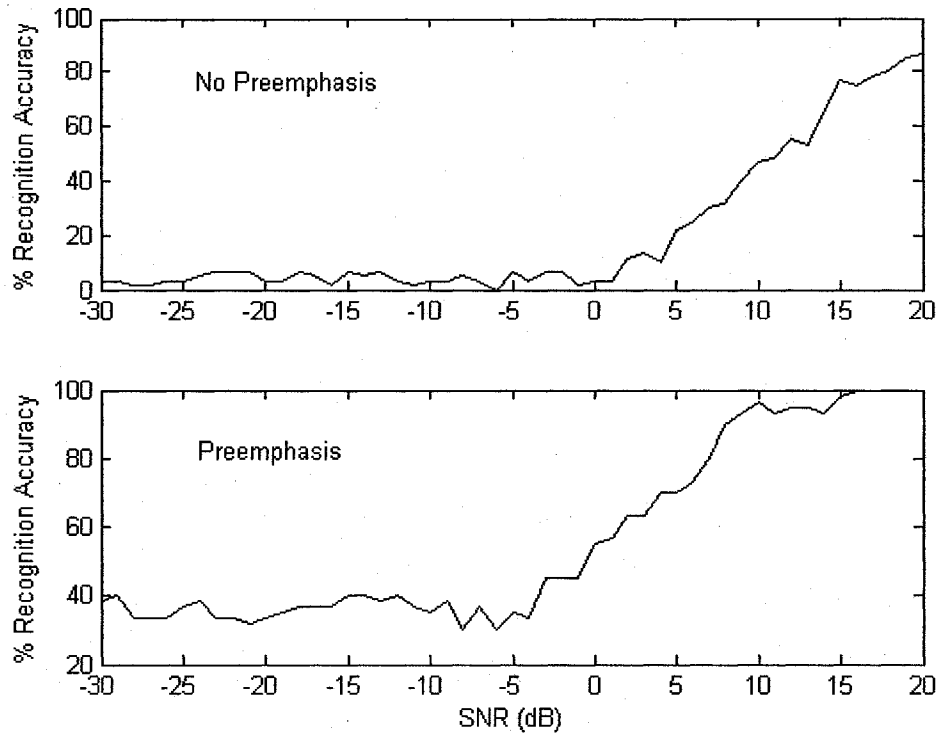


Figure 6.1 preemphasis vs no preemphasis

LPC Smoothing

In this thesis, we introduced an LPC smoother used to smoothen magnitude envelopes to make it easier to apply thresholds to extract speech. We used this LPC smoother throughout the testing performed for this thesis. Figure 6.2 displays the results of testing the 'Time Magnitude' algorithm with various noisy speech files using several smootheners. Compared to the other smootheners, the LPC smoother allowed for the highest recognition accuracy.

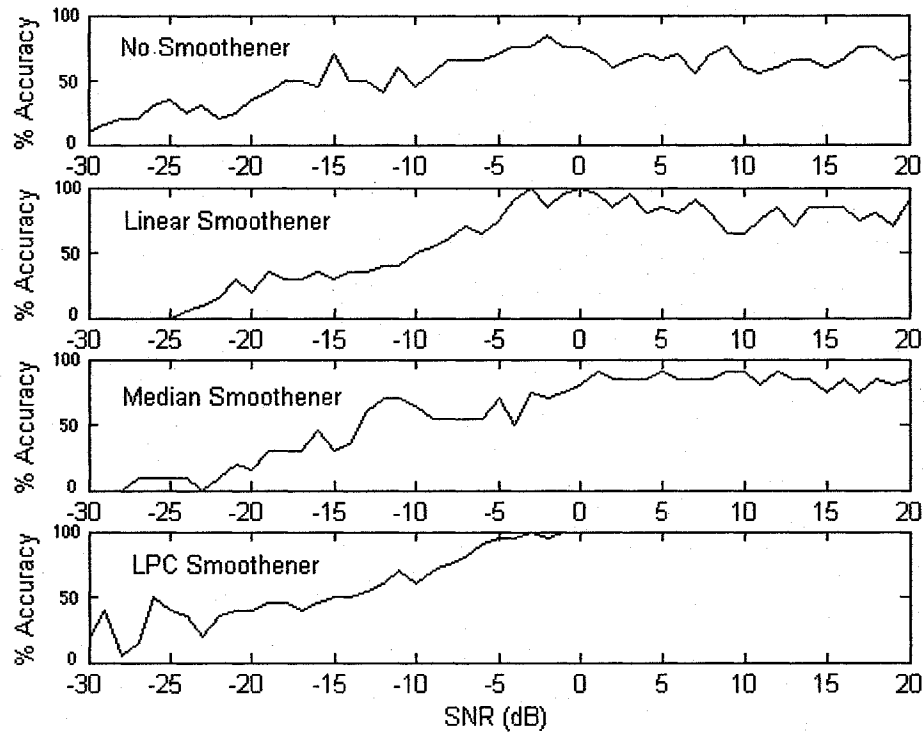


Figure 6.2 performance of various smootheners

6.2 Testing

Our testing tool provides a means of comparing and contrasting the performance of various word boundary detection algorithms. We have tested 4 well know algorithms using 3 different speakers and 3 different types of noise. The performances of these algorithms are compared using two measures: recognition accuracy and boundary errors.

Recognition Accuracy

The recognition accuracy is a measure of how well the SAPI driven recognizer is able to recognize clean speech segments with boundaries computed by a word boundary detection algorithm.

Boundary Error

The boundary error is a measure of how close the computed boundaries are to the manually deduced boundaries. The boundary error is measured as:

$$\text{Boundary Error} = \text{Manually Deduced Boundary} - \text{Computed Boundary}$$

Boundary errors are positive when the computed boundaries lie to the left of the manually deduced boundaries and are negative when the computed boundaries lie to the right of the manually deduced boundaries. When the computed begin boundary error is negative or the computed end boundary error is positive, then that computed boundary is said to lie within its manually deduced counterpart, and part of the speech signal is lost. However, if the computed begin boundary error is positive or the computed end boundary error is negative, then that computed boundary is said to lie outside its corresponding manually deduced counterpart and no speech signal is lost.

In the following sections, we present the results obtained from testing four algorithms for their recognition accuracies and boundary errors. All four algorithms employed the same 1st order preemphasizer, frame length of 150 samples and frame overlap of 50 samples, a Hamming window, an LPC smoothener and a threshold calculated from the smoothened magnitude envelope $s[i]$ by:

$$threshold = 0.3 + \sum_{i=1}^{20} s(i)$$

6.2.1 The Time Algorithm

Recognition Accuracy

Figure 6.3 displays the recognition accuracy of the Time algorithm against SNR values from -30dB to 20dB. The algorithm displayed an accuracy of 98% at 20dB and deteriorated at lower SNR values.

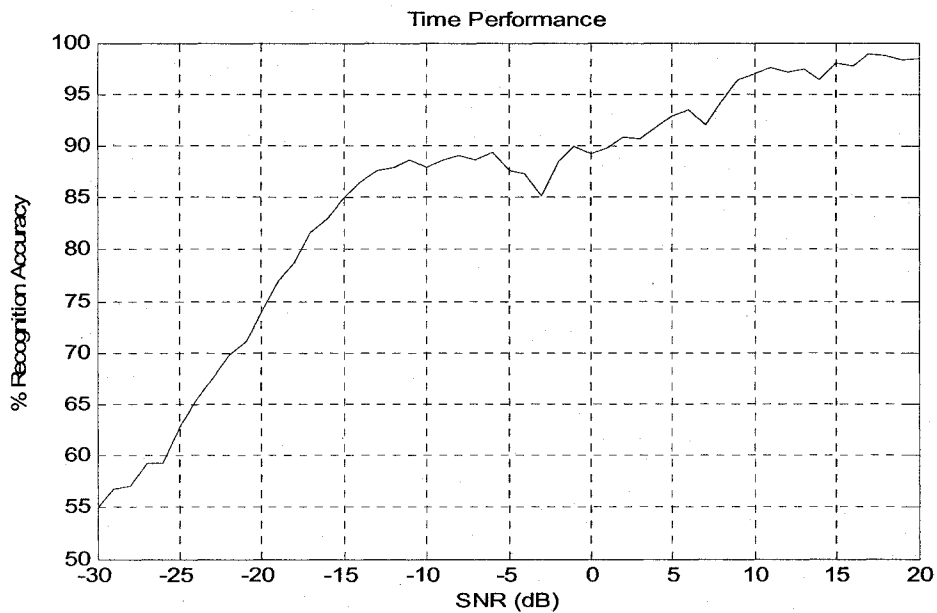


Figure 6.3 recognition accuracy of the Time algorithm

The performance of the Time algorithm was less than perfect even at SNR values as high as 20dB. Further investigation revealed that this was mainly due to the algorithm's difficulty in accurately computing boundaries for the utterance 'six'. An

example of this limitation is shown in Figure 6.4, wherein the algorithm failed to accurately compute the end boundary of the utterance 'six' at an SNR of 20dB.

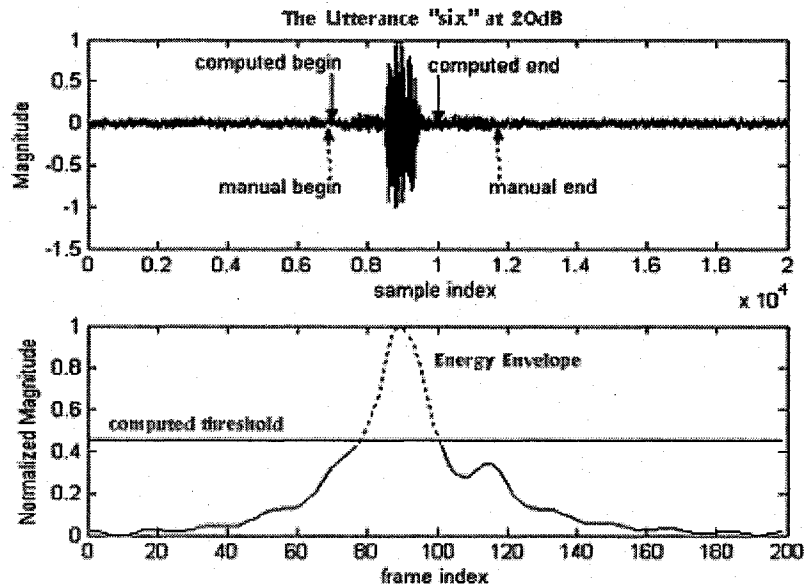


Figure 6.4 an example of the failings of the Time algorithm

Boundary Errors

The probability distribution functions (PDFs) of boundary errors and the distributions of the positions of the boundary errors relative to their manually deduced counterparts are shown in Figure 6.5 to Figure 6.12. These distributions are shown for several ranges of SNR values to provide insight into the performance of the Time algorithm under these ranges.

The PDFs of begin and end boundary errors of the Time algorithm under SNR values from -30dB to 20dB are shown in Figure 6.5. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown.

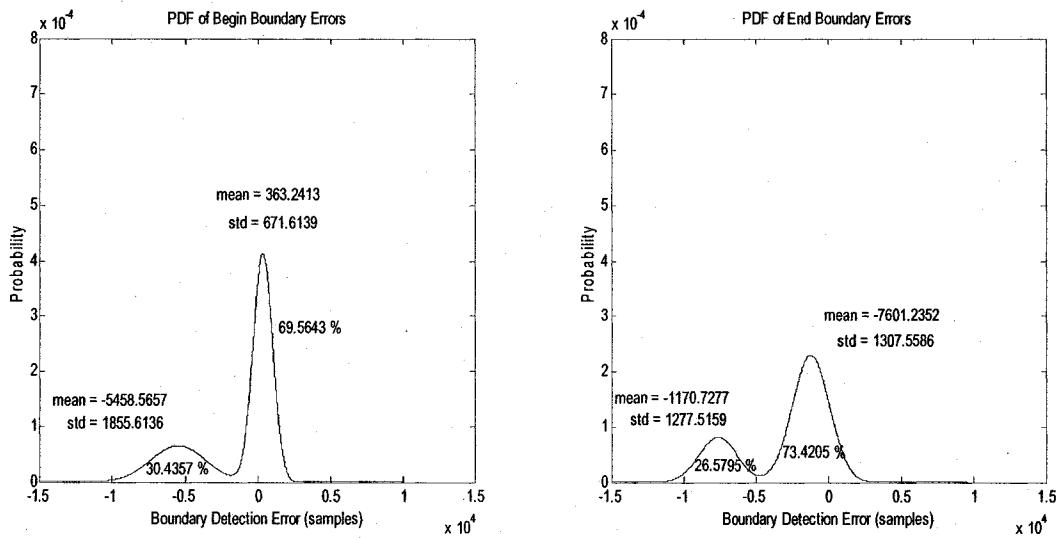


Figure 6.5 PDF of boundary errors from (Time Algorithm, -30dB to 20dB)

Figure 6.6 shows the distribution of boundary error positions for SNR values from -30dB to 20dB. It can be seen that roughly 52% of computed begin boundaries and 90% of computed end boundaries lay within their manually deduced counterparts.

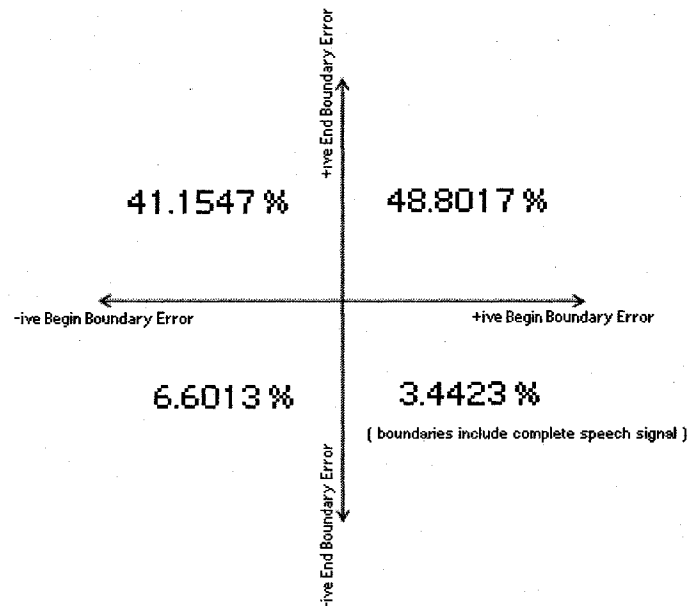


Figure 6.6 boundary error positions (Time Algorithm, -30dB to 20 dB)

The PDFs of begin and end boundary errors made by the Time algorithm under SNR values from 10dB to 20dB are shown in Figure 6.7. The distributions of begin and end boundaries were found to be bimodal and trimodal respectively with means and standard deviations as shown. The recognition accuracy of the Time algorithm against this range of SNR values was greater than 94%. This high performance is reflected in the small magnitudes of the means and standard deviations as well as the large sizes of the primary modes relative to the other modes of the distributions.

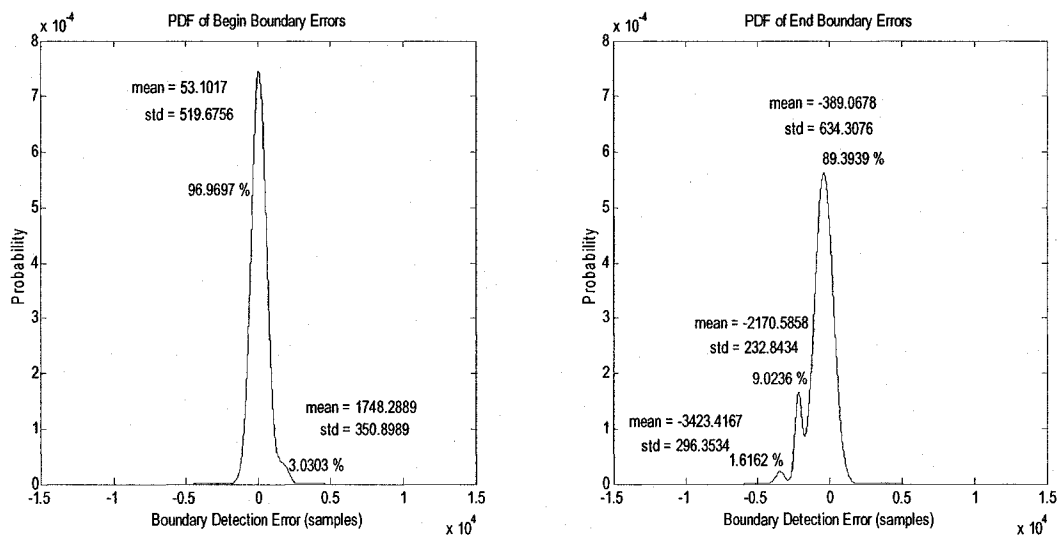


Figure 6.7 PDF of boundary errors (Time Algorithm, 10dB to 20dB)

Figure 6.8 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from 10dB to 20dB. It can be seen that 54% of computed begin boundaries and 74% of computed end boundaries lay within their corresponding manual boundaries. In spite of these seemingly unfavorable results (favorable results would have computed boundaries lying outside their manually deduced counterparts), the algorithm displayed high recognition accuracy. This would suggest that even if the computed boundaries lie within the

manually deduced boundaries, recognition accuracy is not harmed as long as the magnitudes of boundary errors are small.

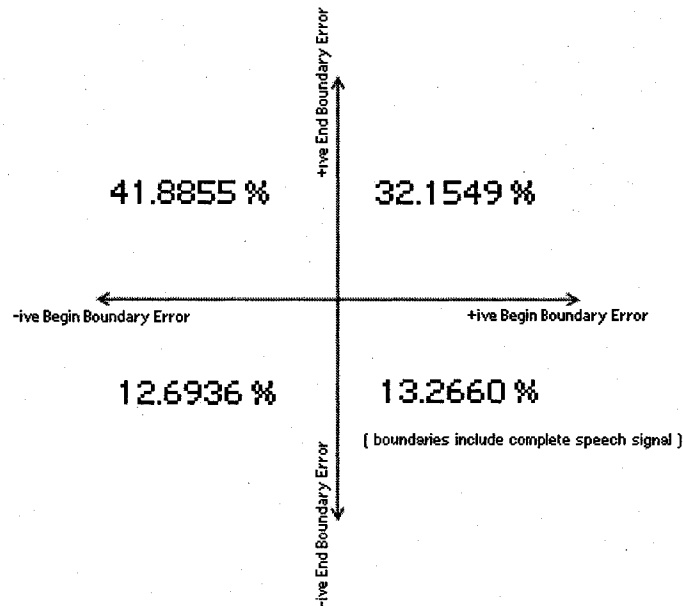


Figure 6.8 boundary error positions (Time Algorithm, 10dB to 20dB)

The PDFs of begin and end boundary errors made by the Time algorithm under SNR values from -19dB to 9dB are shown in Figure 6.9. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the Time algorithm against this range of SNR values was found to be between 75% and 97%.

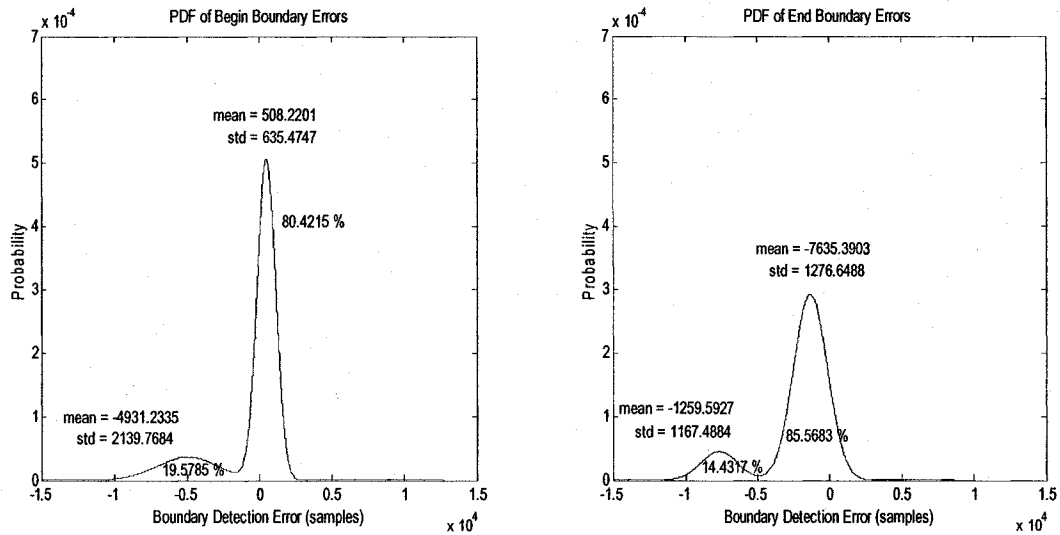


Figure 6.9 PDF of boundary errors (Time Algorithm, -19dB to 9dB)

Figure 6.10 shows the distribution of boundary error positions for SNR values ranging from -19dB to 9dB. It can be seen that 62% of computed begin boundaries and 92% of computed end boundaries lay within their corresponding manually deduced boundaries.

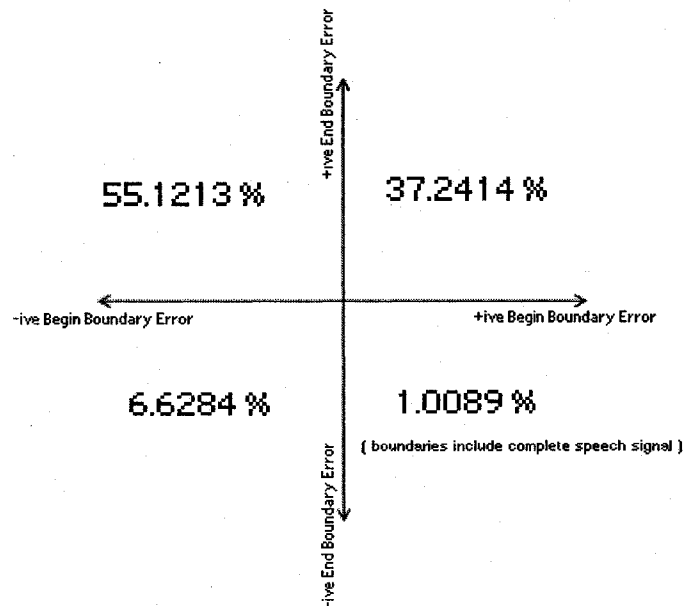


Figure 6.10 boundary error positions (Time Algorithm, -19dB to 9dB)

The PDFs of begin and end boundary errors made by the Time algorithm under SNR values from -30dB to -20dB are shown in Figure 6.11. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the Time algorithm against this range of SNR values was less than 75%. This is reflected in the large magnitudes of the means and standard deviations of the primary modes of the distributions.

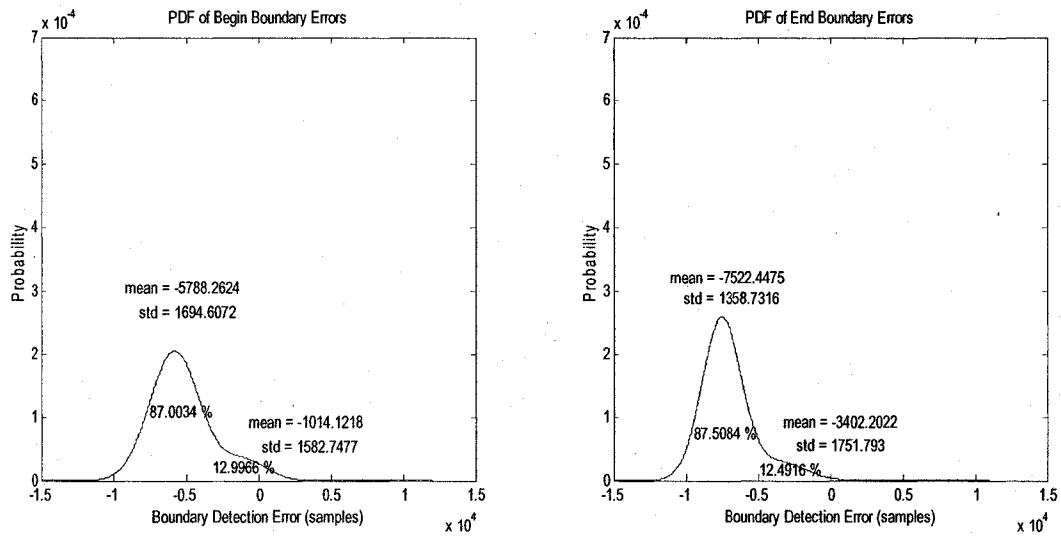


Figure 6.11 PDF of boundary errors (Time Algorithm, -30dB to -20dB)

Figure 6.12 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from -30dB to -20dB. It can be seen that 4% of computed begin boundaries and 99% of computed end boundaries lay within their corresponding manual boundaries.

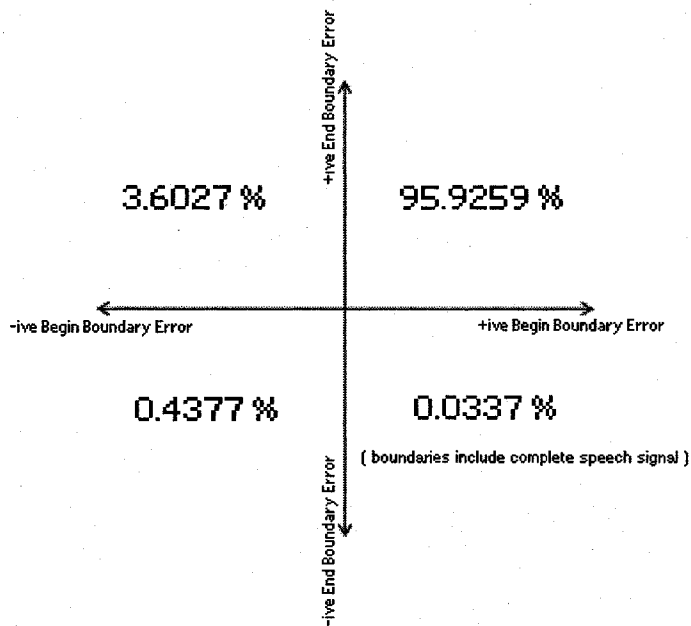


Figure 6.12 boundary error positions (Time Algorithm, -30dB to -20dB)

6.2.2 The LPC Algorithm

Recognition Accuracy

Figure 6.13 displays the recognition accuracy of the LPC algorithm against SNR values from -30dB to 20dB. The LPC word boundary detection algorithm displayed an accuracy of 98% at an SNR of 20dB and its performance deteriorated gradually below SNR values of 5dB.

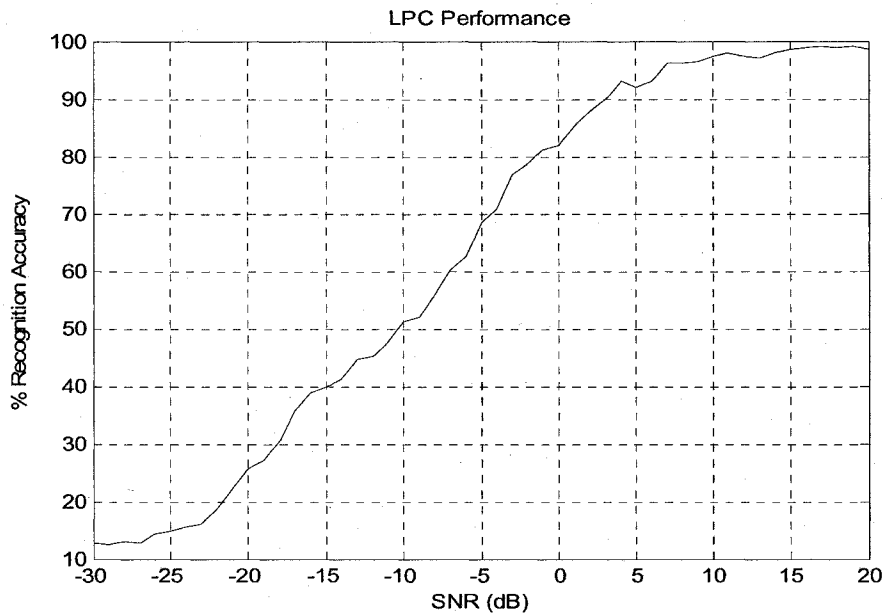


Figure 6.13 recognition accuracy of the LPC algorithm

Boundary Errors

The probability distribution functions (PDFs) of boundary errors and the distribution of the positions of the boundary errors made by the LPC algorithm are shown in Figure 6.14 to Figure 6.21. These distributions are shown for several ranges of SNR values to provide insight into the performance of the LPC algorithm under these ranges.

The PDFs of begin and end boundary errors made by the LPC algorithm under SNR values from -30dB to 20dB are shown in Figure 6.14. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown.

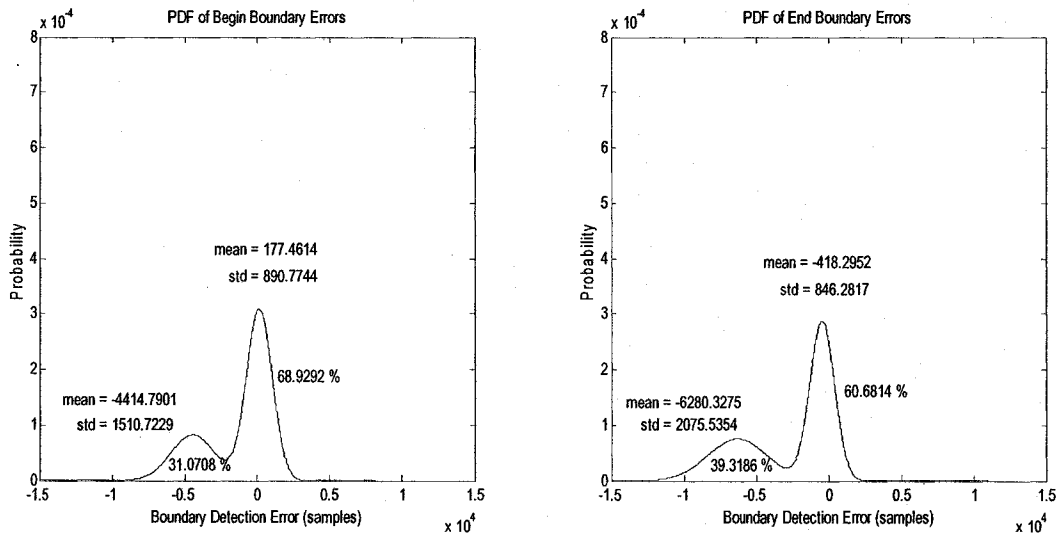


Figure 6.14 PDF of boundary errors (LPC Algorithm, -30dB to 20dB)

Figure 6.15 shows the distribution of boundary error positions for the SNR values ranging from -30dB to 20dB. It can be seen that roughly 64% of computed begin boundaries and 20% of computed end boundaries lay within their manually deduced counterparts.

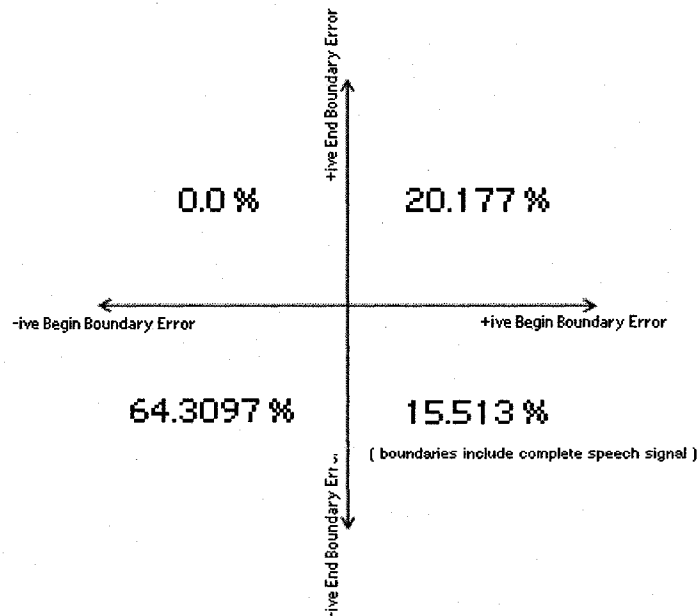


Figure 6.15 boundary error positions (LPC Algorithm, -30dB to 20dB)

The PDFs of begin and end boundary errors made by the LPC algorithm under SNR values from 5dB to 20dB are shown in Figure 6.16. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the LPC algorithm against this range of SNR values was greater than 93%. This is reflected in the small magnitudes of the means and standard deviations as well as the large sizes of the primary modes relative to the other modes of the distributions.

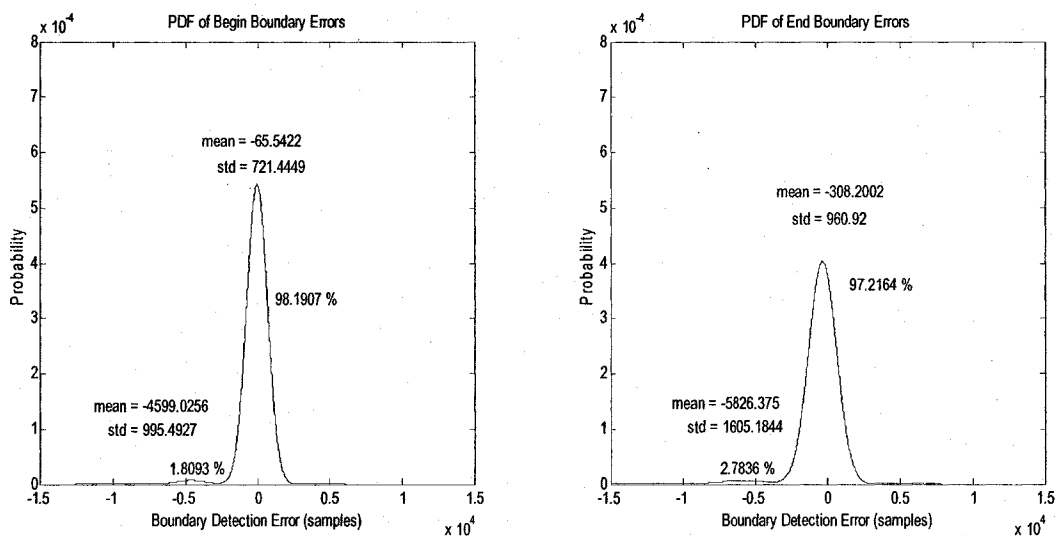


Figure 6.16 PDF of boundary errors from (LPC Algorithm, 5dB to 20dB)

Figure 6.17 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from 5dB to 20dB. It can be seen that 62% of computed begin boundaries and 41% of computed end boundaries lay within their manually deduced counterparts

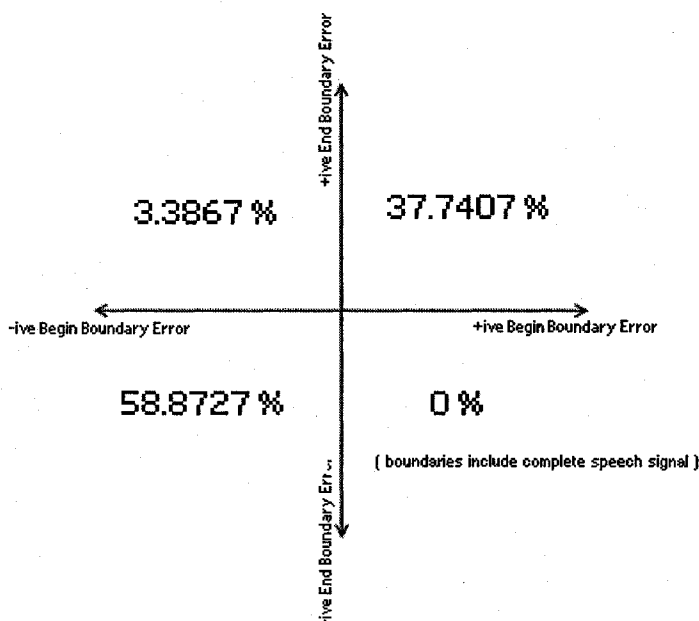


Figure 6.17 boundary error positions (LPC Algorithm, 5dB to 20dB)

The PDFs of begin and end boundary errors made by the LPC algorithm under SNR values from -24dB to 4dB are shown in Figure 6.18. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the LPC algorithm against this range of SNR values was found to be between 15% and 92%.

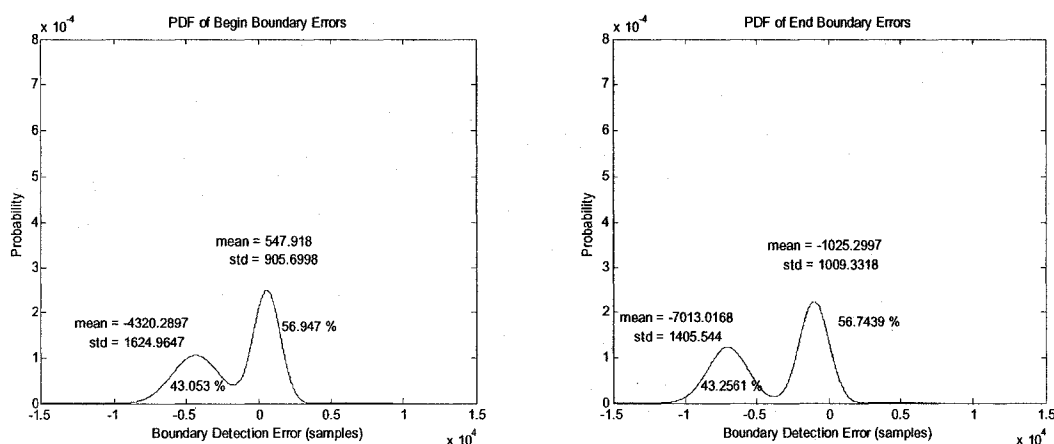


Figure 6.18 PDF of boundary errors (LPC Algorithm, -24dB to 4dB)

Figure 6.19 shows the distribution of boundary error positions for SNR values ranging from -24dB to 4dB. It can be seen that 40% of computed begin boundaries and 92% of computed end boundaries lay within their corresponding manually deduced boundaries

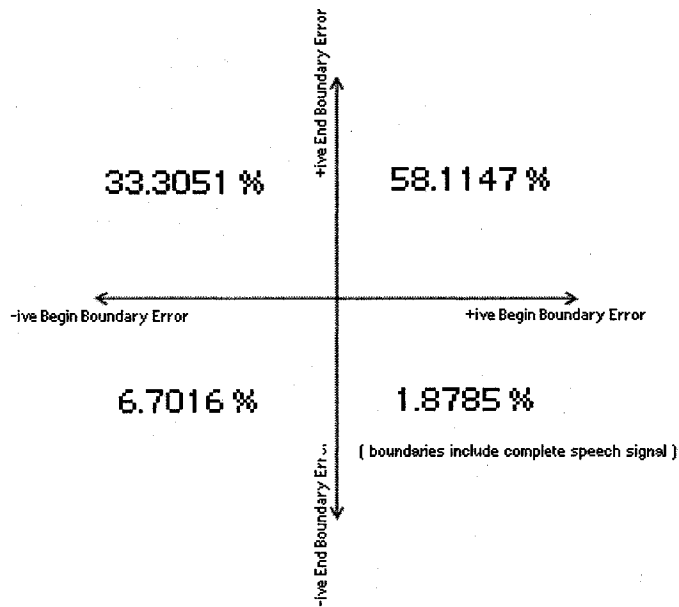


Figure 6.19 boundary error positions (LPC Algorithm, -24dB to 4dB)

The PDFs of begin and end boundary errors made by the LPC algorithm under SNR values from -30dB to -25dB are shown in Figure 6.20. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the LPC algorithm against this range of SNR values was less than 15%. This is reflected in the large magnitudes of the means and standard deviations of the primary modes of the distributions.

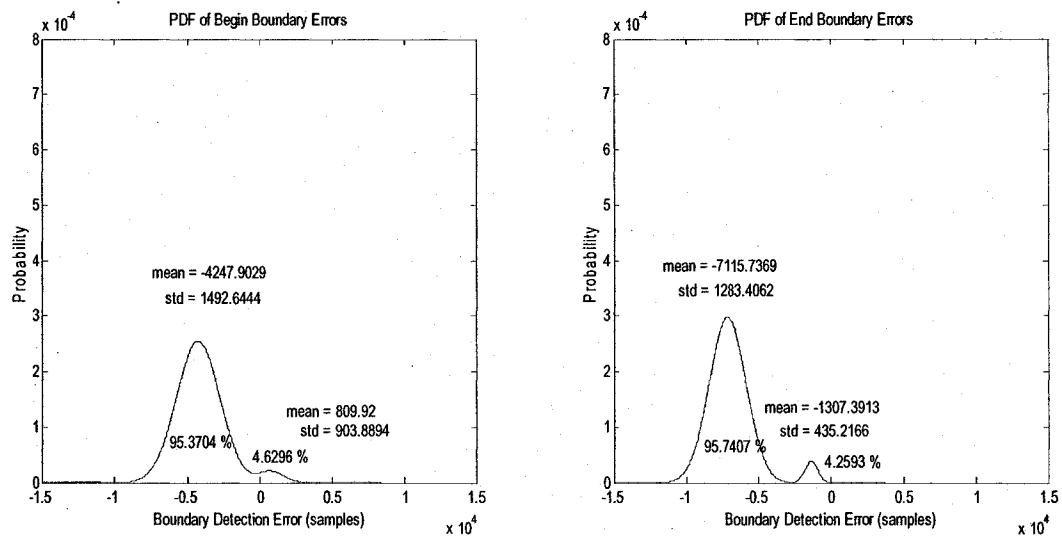


Figure 6.20 PDF of boundary errors (LPC Algorithm, -30dB to -25dB)

Figure 6.21 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from -30dB to -25dB. It can be seen that 4% of computed begin boundaries and 100% of computed end boundaries lay within their corresponding manual boundaries.

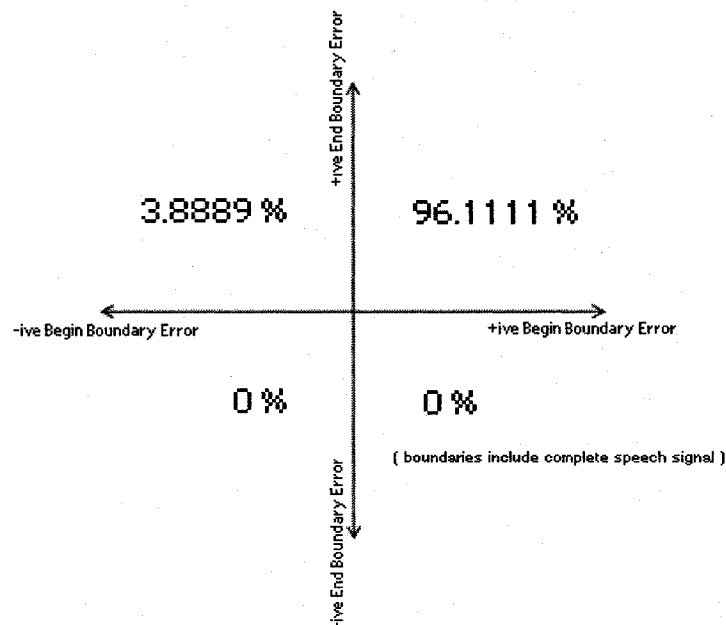


Figure 6.21 boundary error positions (LPC Algorithm, -30dB to -25dB)

The LPC algorithm failed to compute any boundaries for numerous test files, most of which contained white noise at very low SNR values. Figure 6.22 indicates the percentage of all test files that the LPC algorithm failed to compute boundaries for.

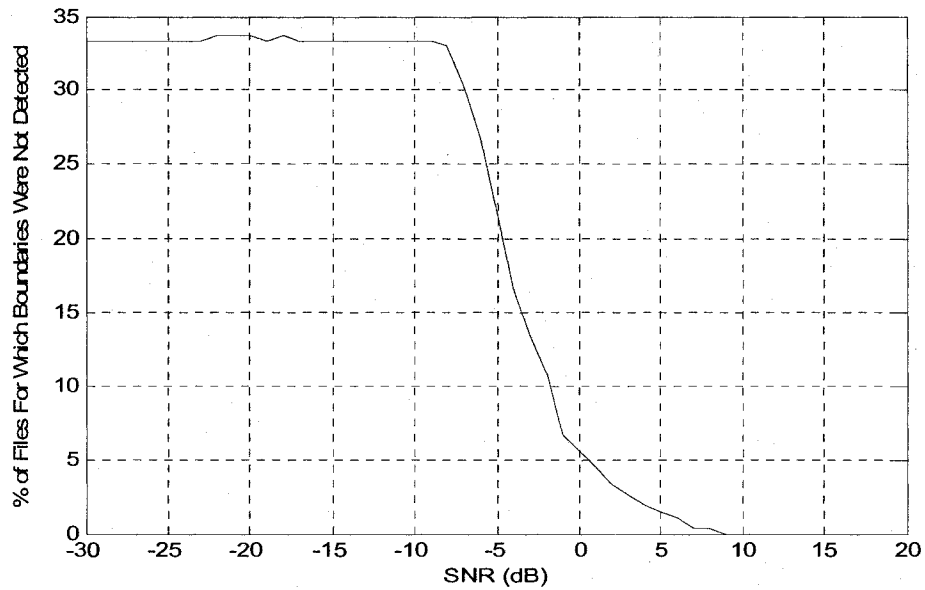


Figure 6.22 % of files with no boundaries detected

Figure 6.23 shows an example where the LPC algorithm failed to compute boundaries for one of the test files.

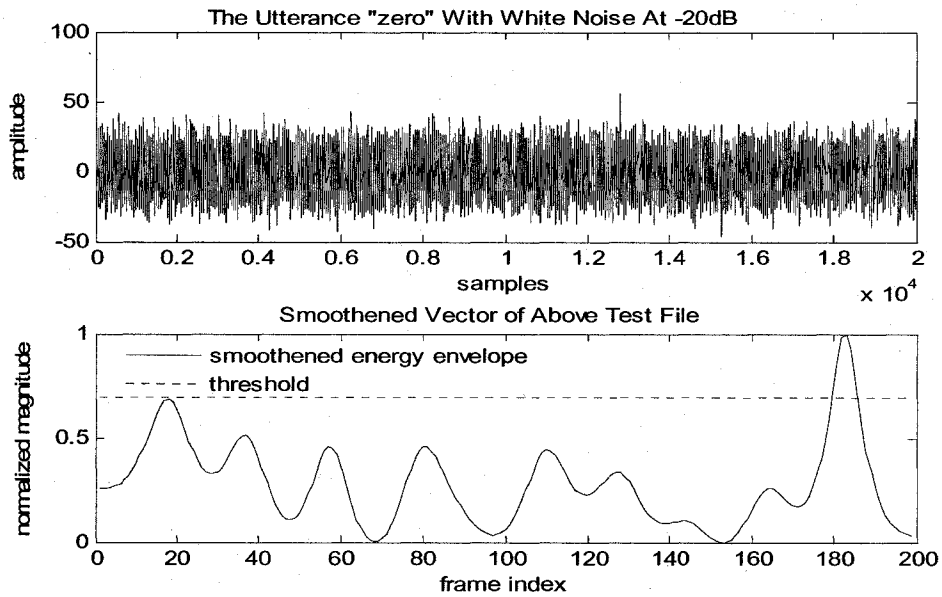


Figure 6.23 no boundaries detected

6.2.3 The LFCC Algorithm

Recognition Accuracy

Figure 6.24 displays the recognition accuracy of the LFCC algorithm against SNR values from -30dB to 20dB. The LFCC word boundary detection algorithm displayed an accuracy of about 100% at 20dB and an accuracy of around 95% at 0dB.

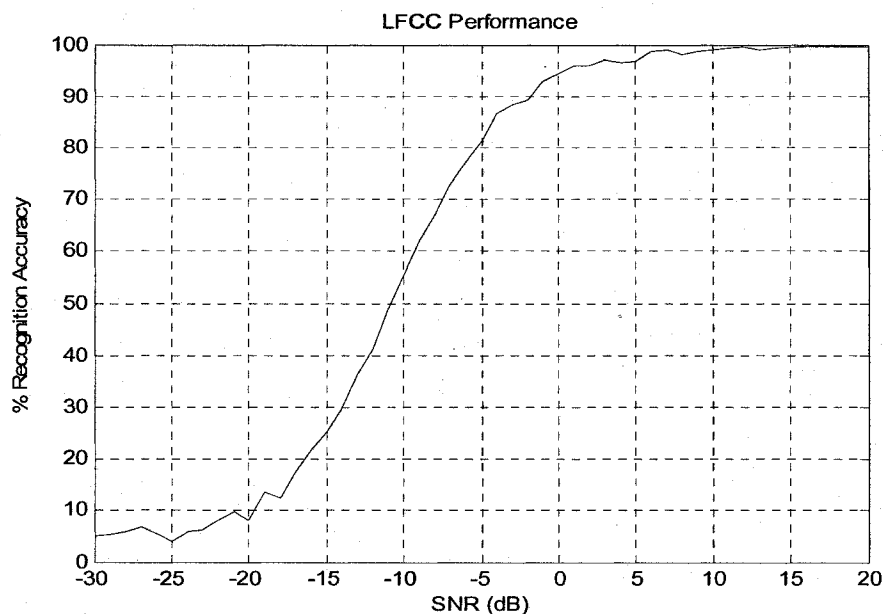


Figure 6.24 recognition accuracy of the LFCC algorithm

Boundary Errors

The probability distribution functions (PDFs) of boundary errors and the distributions of the positions of the boundary errors made by the LFCC algorithm are shown in Figure 6.25 to Figure 6.32. These distributions are shown for several ranges of SNR values to provide insight into the performance of the LFCC algorithm under these ranges.

The PDFs of begin and end boundary errors made by the LFCC algorithm under SNR values from -30dB to 20dB are shown in Figure 6.25. The distributions of begin and end boundaries were found to be trimodal with means and standard deviations as shown.

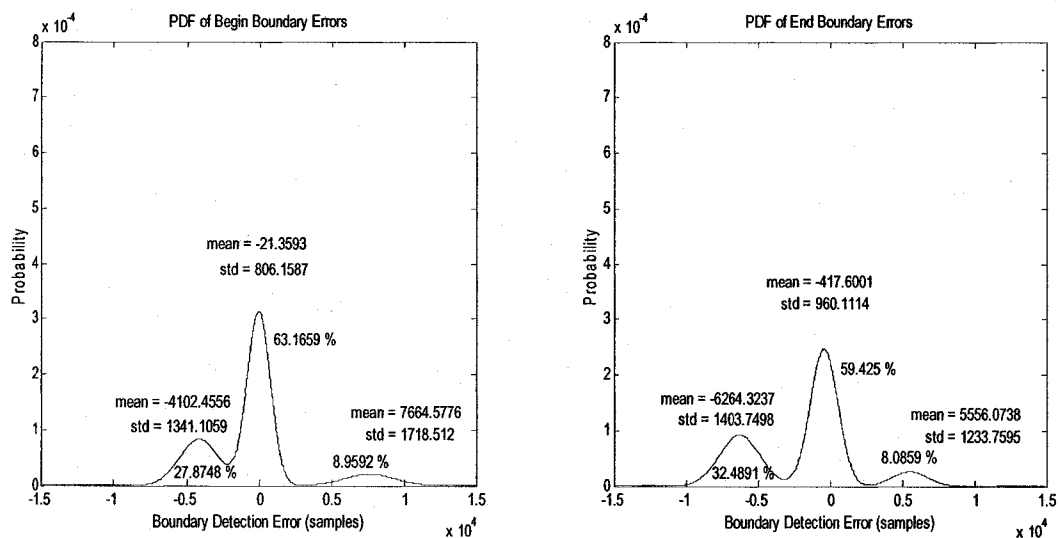


Figure 6.25 PDF of boundary errors (LFCC Algorithm, -30dB to 20dB)

Figure 6.26 shows the distribution of boundary error positions for SNR values ranging from -30dB to 20dB. It can be seen that roughly 38% of computed begin boundaries and 70% of computed end boundaries lay within their manually deduced counterparts.

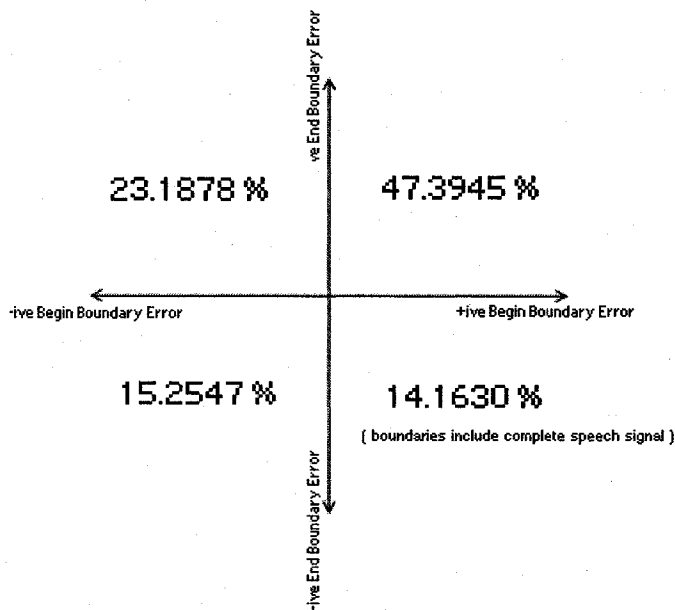


Figure 6.26 boundary error positions (LFCC Algorithm, -30dB to 20dB)

The PDFs of begin and end boundary errors made by the LFCC algorithm under SNR values from 0dB to 20dB are shown in Figure 6.27. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown. The recognition accuracy of the LFCC algorithm within this SNR range was greater than 95%. This high performance is reflected in the small magnitudes of means and standard deviations of the primary modes as well as the large sizes of the primary modes relative to the other modes.

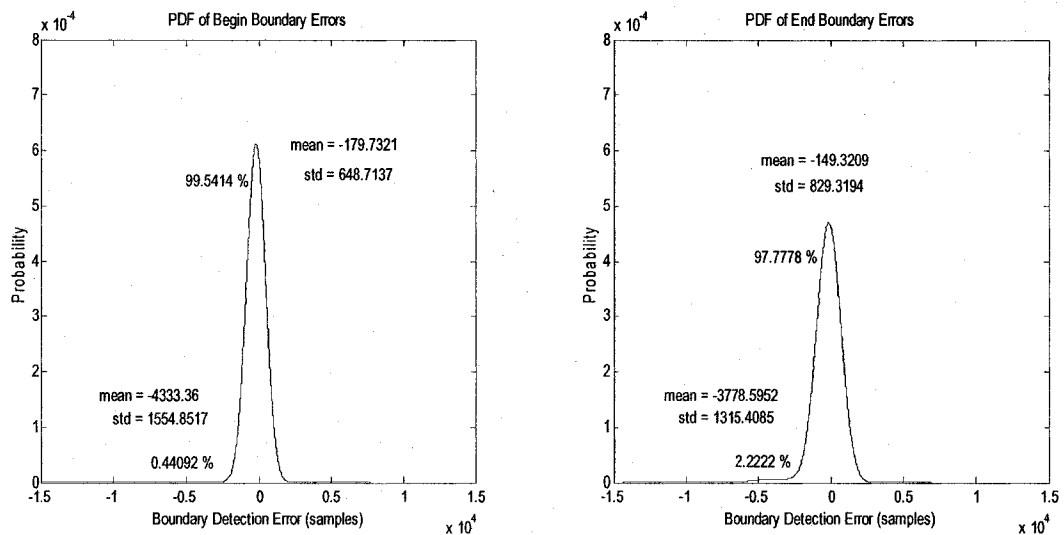


Figure 6.27 PDF of boundary errors (LFCC Algorithm, 0dB to 20dB)

Figure 6.28 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from 0dB to 20dB. In spite of the algorithms high recognition accuracy in this range SNR range, it can be seen that 37% of computed begin boundaries and 54% of computed end boundaries lay within their manually deduced counterparts.

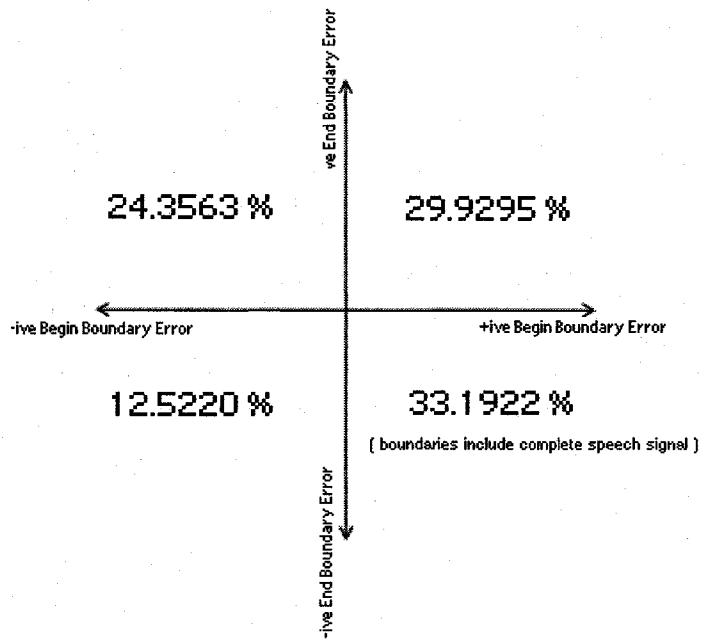


Figure 6.28 boundary error positions (LFCC Algorithm, 0dB to 20dB)

The PDFs of begin and end boundary errors made by the LFCC algorithm under SNR values from -19dB to -1dB are shown in Figure 6.29. The distributions of begin and end boundaries were found to be trimodal with means and standard deviations as shown. The recognition accuracy of the LFCC algorithm against this range of SNR values was found to be between 9% and 95%.

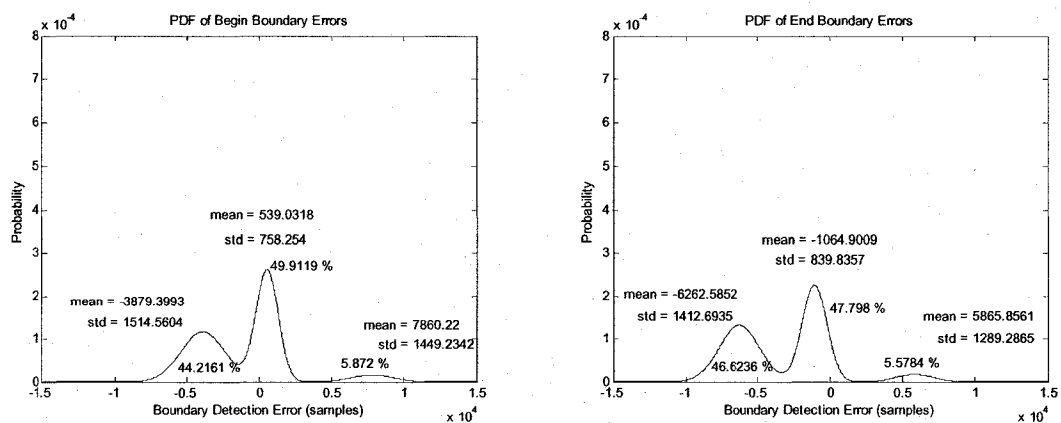


Figure 6.29 PDF of boundary errors (LFCC Algorithm, -19dB to -1dB)

Figure 6.30 shows the distribution of boundary error positions for SNR values from -24dB to 4dB. It can be seen that 53% of computed begin boundaries and 90% of computed end boundaries lay within their corresponding manually deduced boundaries.

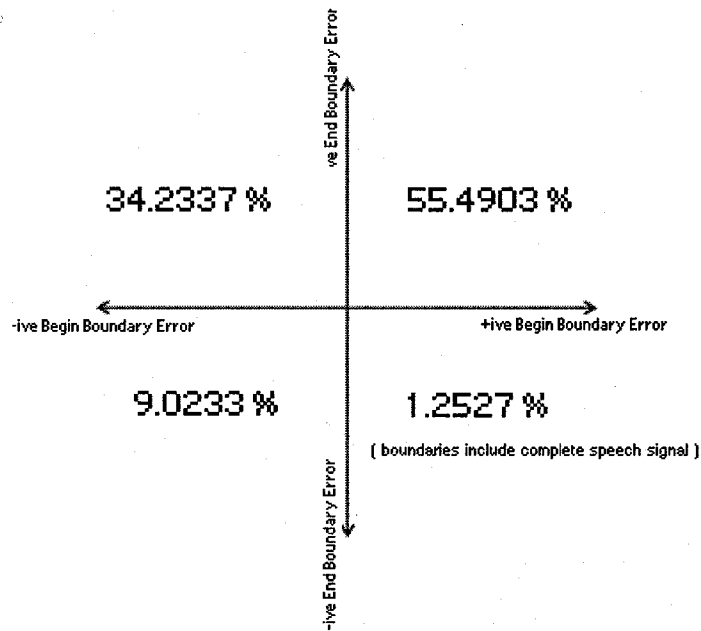


Figure 6.30 boundary error positions (LFCC Algorithm, -19dB to -1dB)

The PDFs of begin and end boundary errors made by the LFCC algorithm under SNR values from -30dB to -20dB are shown in Figure 6.31. The distributions of begin and end boundaries were found to be bimodal with means and standard deviations as shown.

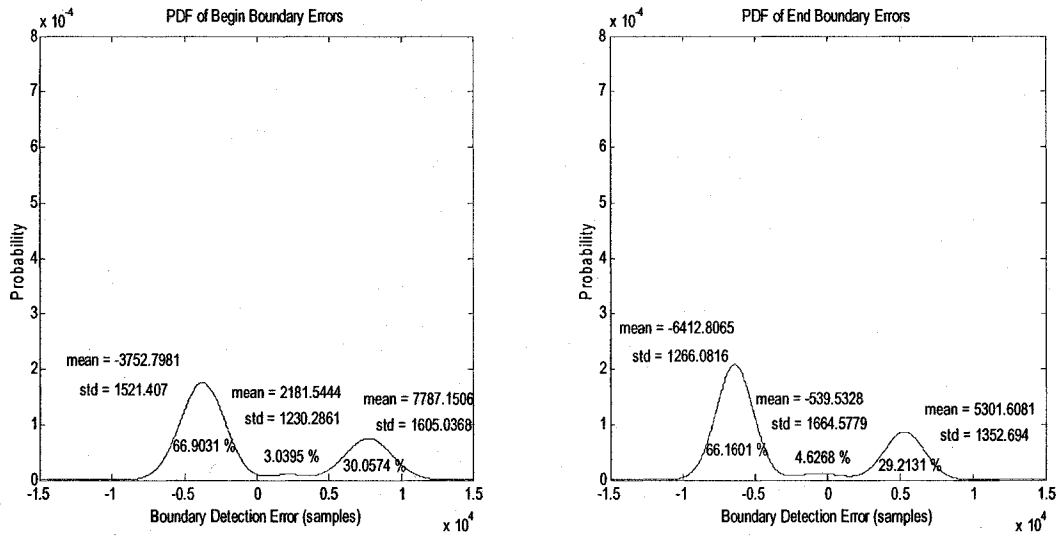


Figure 6.31 PDF of boundary errors (LFCC Algorithm, -30dB to -20dB)

Figure 6.32 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from -30dB to -20dB. It can be seen that 33% of computed begin boundaries and 68% of computed end boundaries lay within their corresponding manually deduced counterparts.

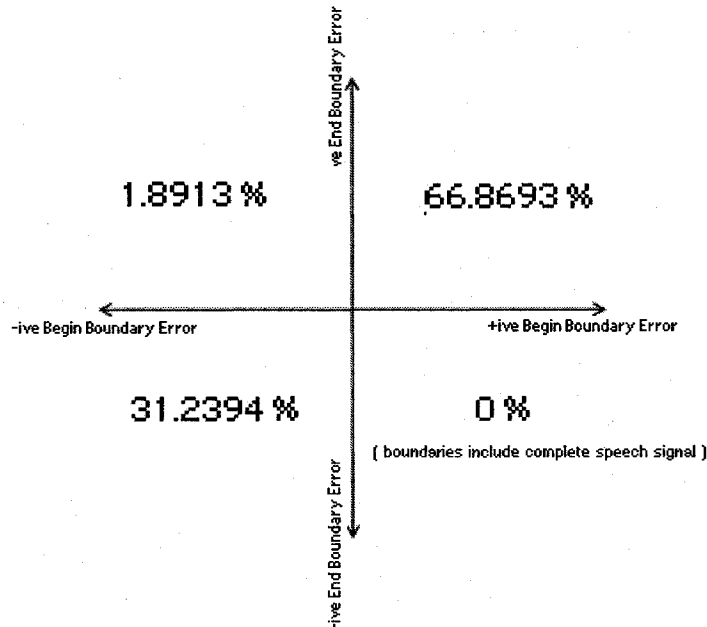


Figure 6.32 boundary error positions (LFCC Algorithm, -30dB to -20dB)

6.2.4 The MFCC Algorithm

Recognition Accuracy

Figure 6.33 displays the recognition accuracy of the MFCC algorithm against SNR values from -30dB to 20dB. The MFCC word boundary detection algorithm displayed an accuracy of 95% at 20dB and displayed rapid deterioration in performance at lower SNR values.

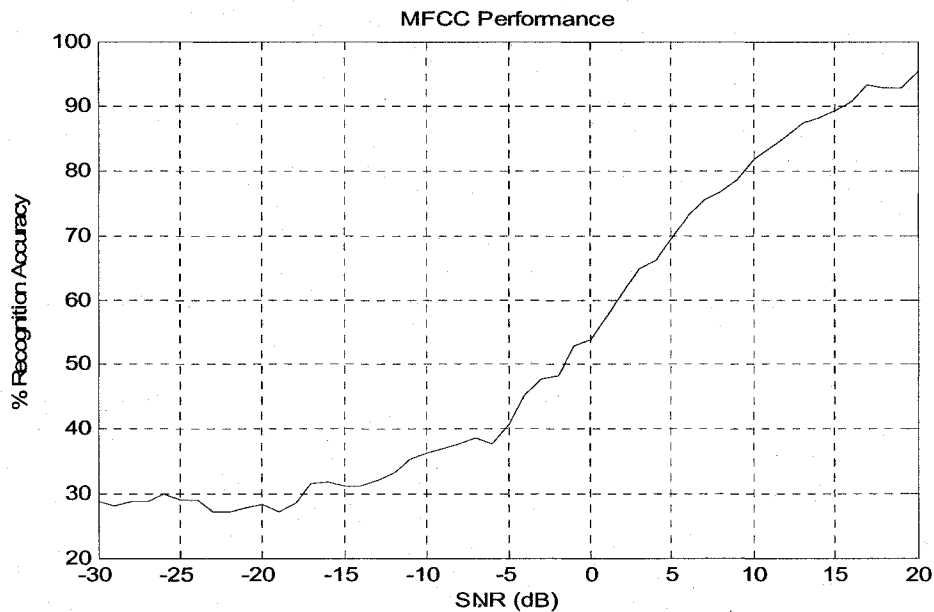


Figure 6.33 recognition accuracy of the MFCC algorithm

Like the Time algorithm, the performance of the MFCC algorithm was less than perfect even at SNR values as high as 20dB. This was also due to the algorithm's difficulty in accurately computing boundaries for the utterance 'six'. An example of this limitation is shown in Figure 6.34, wherein the algorithm failed to accurately compute the end boundary of the utterance 'six' at an SNR of 20dB.

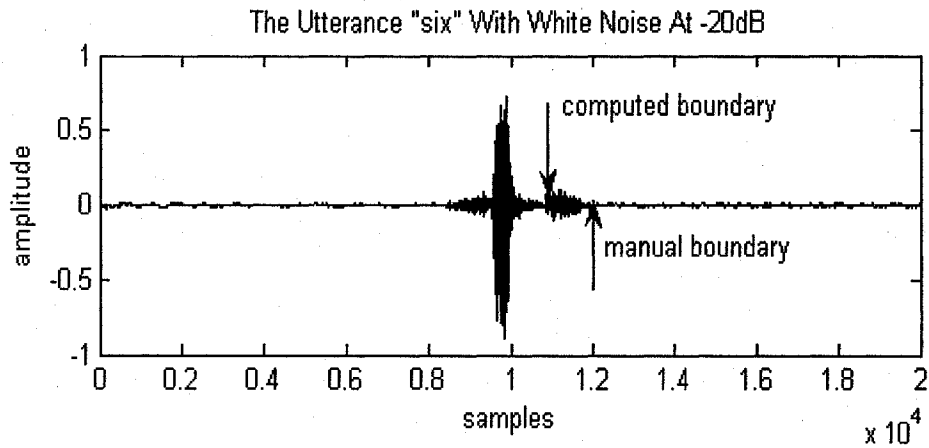


Figure 6.34 an example of the failings of the MFCC algorithm

Boundary Errors

The probability distribution functions (PDFs) of boundary errors and the distributions of the positions of the boundary errors made by the MFCC algorithm are shown in Figure 6.35 to Figure 6.42. These distributions are shown for several ranges of SNR values to provide insight into the performance of the MFCC algorithm under these ranges.

The PDFs of begin and end boundary errors made by the MFCC algorithm under SNR values from -30dB to 20dB are shown in Figure 6.35. The distributions of begin and end boundaries were found to be unimodal with means and standard deviations as shown.

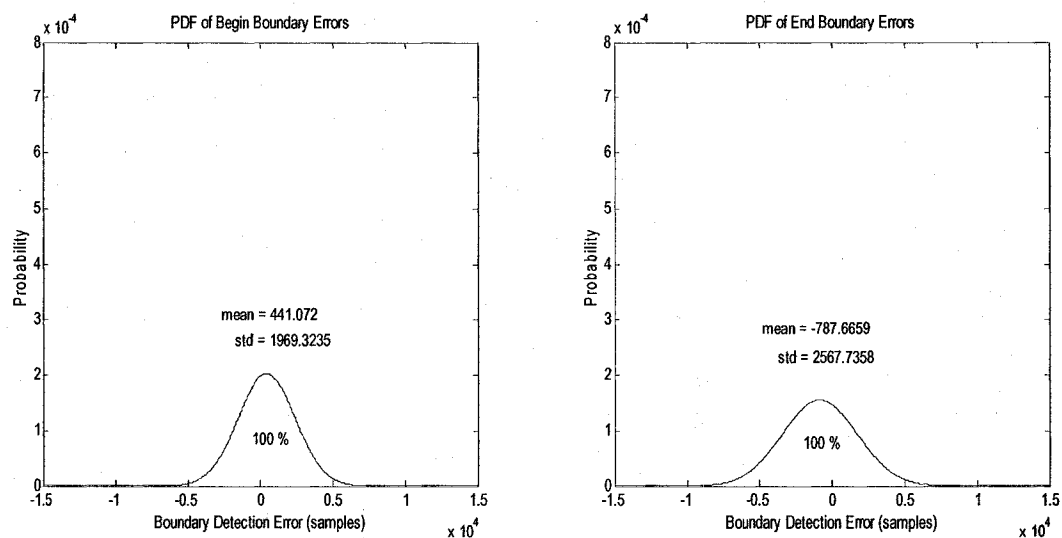


Figure 6.35 PDF of boundary errors (MFCC Algorithm, -30dB to 20dB)

Figure 6.36 shows the distribution of boundary error positions for SNR values ranging from -30dB to 20dB. It can be seen that roughly 57% of computed begin boundaries and 64% of computed end boundaries lay within their manually deduced counterparts.

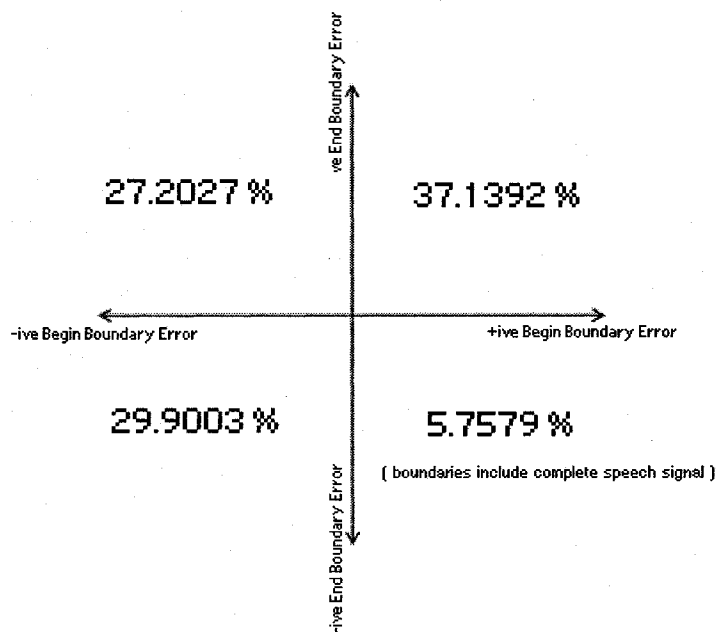


Figure 6.36 boundary error positions (MFCC Algorithm, -30dB to 20dB)

The PDFs of begin and end boundary errors made by the MFCC algorithm under SNR values from 0dB to 20dB are shown in Figure 6.37. The distributions of begin and end boundaries were found to be bimodal and trimodal respectively with means and standard deviations as shown.

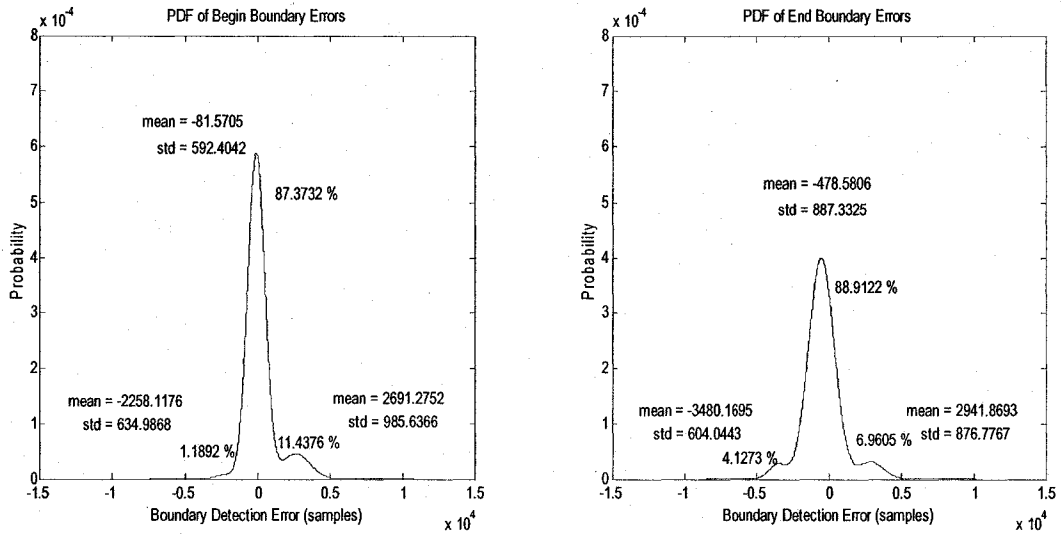


Figure 6.37 PDF of boundary errors (MFCC Algorithm, 0dB to 20dB)

Figure 6.38 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from 10dB to 20dB. It can be seen that 49% of computed begin boundaries and 66% of computed end boundaries lay within their manually deduced counterparts.

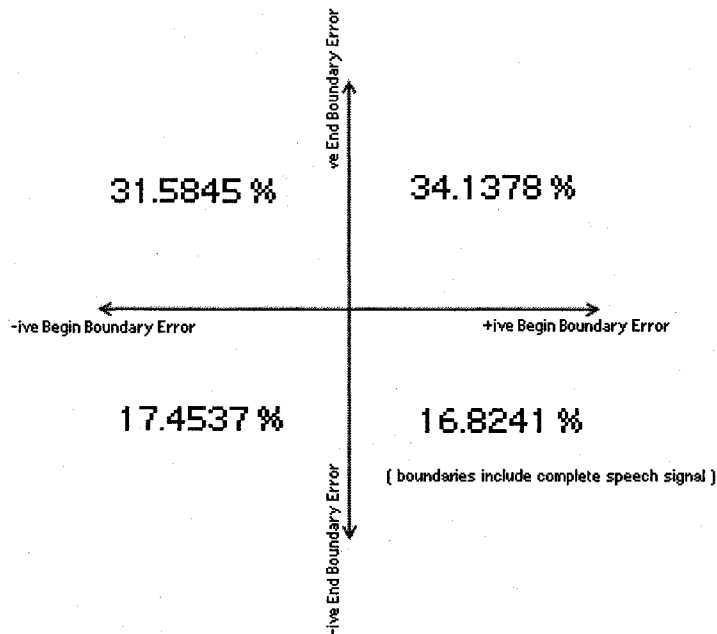


Figure 6.38 boundary error positions (MFCC Algorithm, 0dB to 20dB)

The PDFs of begin and end boundary errors made by the MFCC algorithm under SNR values from -19dB to 9dB are shown in Figure 6.39. The distributions of begin and end boundaries were found to be unimodal with means and standard deviations as shown. The recognition accuracy of the MFCC algorithm against this range of SNR values was found to be between 8% and 81%.

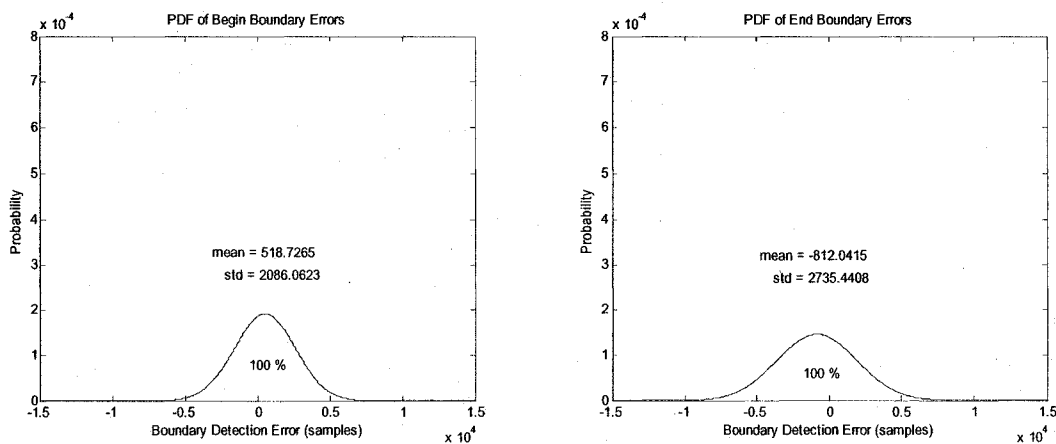


Figure 6.39 PDF of boundary errors (MFCC Algorithm, -19dB to -1dB)

Figure 6.40 shows the distribution of boundary error positions for SNR values ranging from -19dB to 9dB. It can be seen that 60% of computed begin boundaries and 63% of computed end boundaries lay within their corresponding manually deduced boundaries.

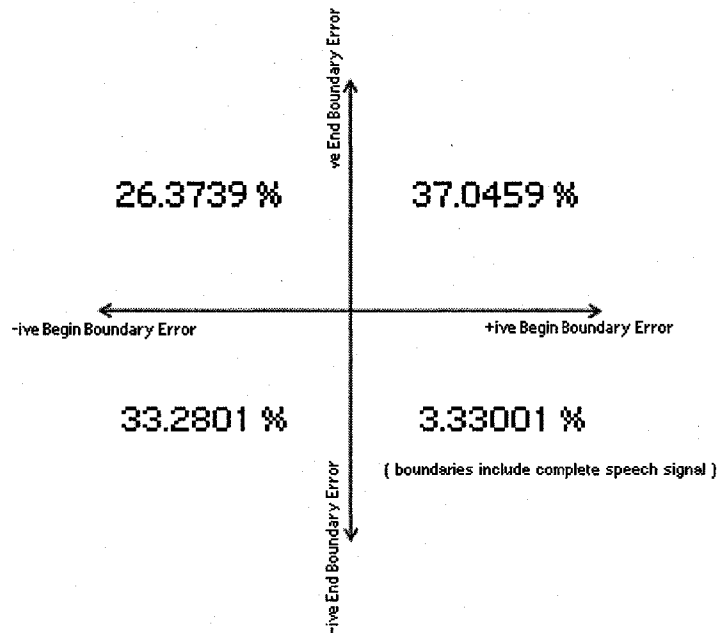


Figure 6.40 boundary error positions (MFCC Algorithm, -19dB to -1dB)

The PDFs of begin and end boundary errors made by the MFCC algorithm under SNR values from -30dB to -20dB are shown in Figure 6.41. The distributions of begin and end boundaries were found to be unimodal and bimodal respectively with means and standard deviations as shown.

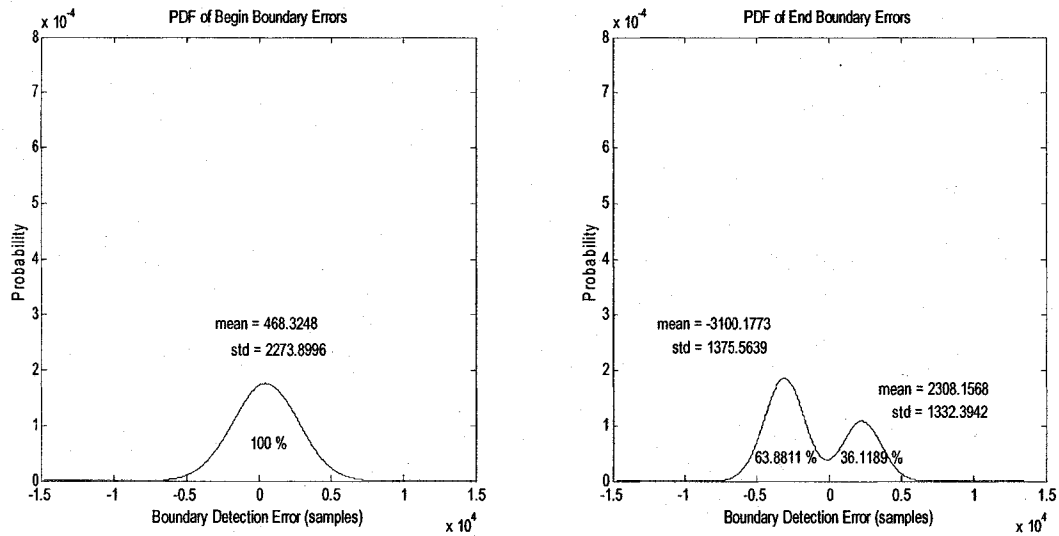


Figure 6.41 PDF of boundary errors (MFCC Algorithm, -30dB to -20dB)

Figure 6.42 shows the distribution of computed boundary positions relative to their manually deduced counterparts for SNR values from -30dB to -20dB. It can be seen that 58% of computed begin boundaries and 65% of computed end boundaries lay within their corresponding manually deduced boundaries.

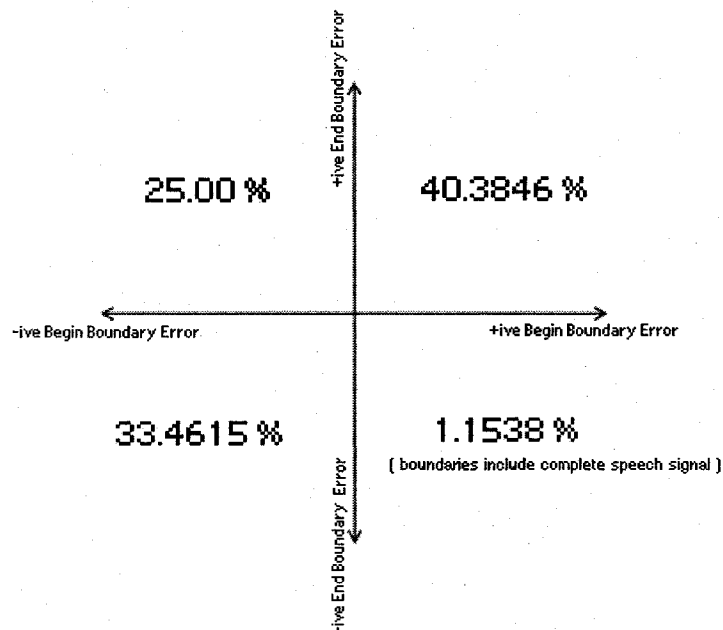


Figure 6.42 boundary error positions (MFCC Algorithm, -30dB to -20dB)

6.2.5 Comparison of Algorithm Performances

In this thesis, we have used recognition accuracy and boundary error magnitude as the two measures of word boundary detection algorithm performance. This section presents the comparison of the performances of the four tested algorithms.

Recognition Accuracy:

A good boundary detection algorithm displays high recognition accuracy even at low values of SNR. Figure 6.43 displays the comparison of recognition accuracies exhibited by the four algorithms tested in this thesis.

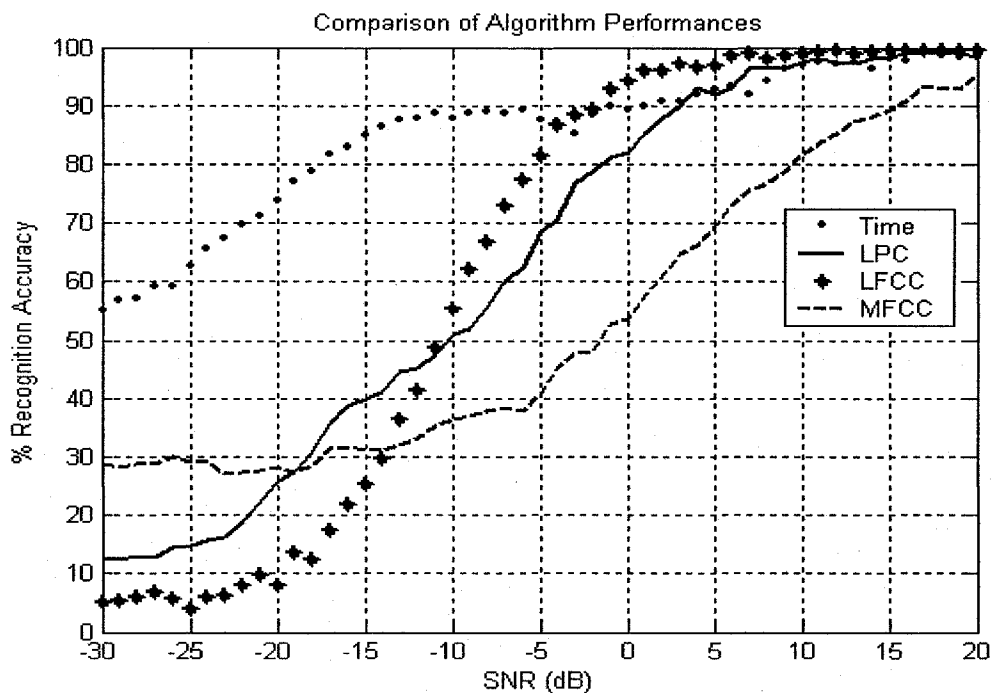


Figure 6.43 comparison of recognition accuracy

- For SNR values above 0dB, the LFCC algorithm displayed the highest recognition accuracy, and the MFCC algorithm displayed the lowest accuracy. The Time and

LPC algorithms displayed similar recognition accuracies slightly lower than that of the LFCC algorithm.

- For SNR values below 0dB, the Time algorithm displayed the highest recognition accuracy of the four algorithms. However, its recognition accuracy was not high enough to be used in any practical speech recognition system under noise conditions below 0dB.
- Although the LFCC algorithm displayed the highest recognition accuracy under SNR values above 0dB, it also displayed the most rapid degradation of recognition accuracy for SNR values less than 0dB.

Boundary Errors

The performance of a good boundary detection algorithm results in boundary errors with very small magnitudes of mean and standard deviation. Figure 6.44 displays the comparison of the means and standard deviations of the boundary errors committed by the four algorithms under noise conditions from 0dB to 20dB.

	BEGIN BOUNDARY ERRORS		END BOUNDARY ERRORS	
	MEAN	STD	MEAN	STD
TIME	239.36 (29.92ms)	613.48 (76.69ms)	-772.30 (-96.54ms)	878.99 (109.87ms)
LPC	-103.52 (-12.94ms)	1051.50 (131.44ms)	-626.46 (-78.31ms)	1466.1 (183.26ms)
LFCC	-198.29 (-24.79ms)	710.79 (88.49ms)	-229.97 (-28.75ms)	998.44 (124.81ms)
MFCC	418.80 (52.35ms)	1482.10 (185.26ms)	-356.54 (-44.56ms)	1884.90 (235.61ms)

Figure 6.44 comparison of boundary errors

Negative boundary errors indicate computed boundaries that lie to the right and positive boundary errors indicate computed boundaries that lie to the left of the manually deduced boundaries. This means that loss of speech information occurs in the case of negative begin boundary errors as well as positive end boundary errors, which typically results in degraded recognition accuracy.

The LFCC algorithm displayed the lowest magnitudes of mean and standard deviation of boundary errors.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

This chapter presents the conclusions obtained from testing the performance of 4 different word boundary detection algorithms and also suggests areas of future research.

7.1 Conclusions

The characteristics of speech that we explored were

- Time Magnitude
- Frequency Magnitude
- Linear Prediction Coefficients
- Linear Cepstral Coefficients
- Mel Cepstral Coefficients

It was observed that these characteristics displayed magnitudes that differed for speech and background noise.

The first task we had set out to complete was to develop a testing tool that allowed the user to implement several boundary detection algorithms and vary their parameters to observe their effects on boundary detection accuracy. Chapter 3 explained in detail

the functionality of the testing tool developed in this thesis. This tool proved very useful in observing the effect different factors had on word boundary detection performance.

Our second task was to develop a word boundary detection algorithm which displayed high boundary detection accuracy. The algorithm we proposed employed Linear Frequency Cepstral Coefficients

Our final task was to perform extensive testing to compare and contrast our proposed boundary detection algorithm with other algorithms. We found that our algorithm displayed higher accuracy than the other tested algorithms and was robust under SNR values as low as 0dB. Other conclusions made in the course of the testing were:

- Preemphasis resulted in higher word boundary detection accuracy.
- The LPC smoothener introduced in this thesis facilitated higher performance of word boundary detection algorithms.
- Our proposed algorithm employing Linear Frequency Cepstral Coefficients displayed the highest word boundary detection performance with a recognition accuracy of above 96% at SNR values above 1dB.
- The recognition accuracy of word boundary detection algorithms does not degrade when computed boundaries fall within their manually deduced counterparts as long as the magnitudes of boundary errors are small (<700 samples).

7.2 Future Research

This section proposes several ideas for future research in the field of word boundary detection

- This thesis was limited to testing boundary detection algorithms with pre-recorded noisy test recordings. Real time testing of boundary detection algorithms can be performed in which an utterance is recorded and the boundaries computed in real time.
- We examined algorithms that employ a given type of speech representation to compute boundaries and it was observed that an algorithms performance differed under different types of noise. Algorithms may be developed wherein the first few frames of background noise are observed and the algorithm's speech representation chosen accordingly.
- The testing of algorithms in this thesis was performed with a 'command and control' grammar which was limited to the digits '0' to '9'. Further testing can be performed using other 'command and control' grammars or a context-free 'dictation' grammar.
- Boundary detection algorithms can be developed that will remain robust even in the presence of non-stationary background noise. This may involve the use of a pulse extraction threshold whose value varies throughout the length of the test recording.
- The scope of this thesis was limited to 'explicit' word boundary detection algorithms. Similar testing can be performed for 'implicit' and 'hybrid' boundary detection algorithms.

BIBLIOGRAPHY

1. *Gin-Der Wu, Chin-Teng Lin (IEEE Transactions on Acoustics, Speech and Signal Processing Vol 8, No 5, September 2000): "Word Boundary Detection with Mel-Scale Frequency Bank in a Noisy Environment"*
2. http://en.wikipedia.org/wiki/Colors_of_noise
3. <http://cnx.org/content/m12469/latest/>
4. http://en.wikipedia.org/wiki/Short-time_Fourier_transform
5. *Lawrence Rabiner, Bing-Hwang Juang (Prentice-Hall Signal Processing Series) : "Fundamentals of Speech Recognition"*
6. *Lawrence Rabiner, M.R.Sambur (The Journal of Acoustical Society of America, 1974) : "An Algorithm for Determining the Endpoints for Isolated Utterances".*
7. *Lawrence Rabiner, R. Schafer : (Prentice-Hall Signal Processing Series) : "Digital Processing of Speech Signals"*
8. *Lori F. Lamel, Aaron E. Rosenberg and Jay G. Wilpon (IEEE Transactions on Acoustics, Speech and Signal Processing, August 1981): "An Improved Endpoint Detector for Isolated Word Recognition"*

9. *Montri Karnjamadecha and Stephen A. Zahorian (IEEE Transactions on Speech and Audio Processing Vol 9, No 6, September 2001) : "Signal Modeling for High-Performance Robust Isolated Word Recognition"*

10. *Philipos C. Loizou and Andreas Spanias (IEEE Transactions on Speech and Audio Processing Vol 4, No 6, November 1996) : "High Performance Alphabet Recognition"*

11. *Steven B. Davis, Paul Mermelstein (IEEE Transactions on Acoustics, Speech and Signal Processing Vol ASSP-28, No 4, August 1980): "Comparisons of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences"*

APPENDIX

IRB APPROVAL LETTER



UNIVERSITY of NEW HAMPSHIRE

June 21, 2006

Andrew Kun
Electrical and Computer Engineering
Kingsbury Hall
Durham, NH 03824

IRB #: 2980

Study: Speech Sample Collection for Speech Recognition Engine Comparison and Development

Review Level: Expedited

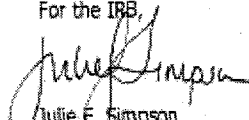
Approval Expiration Date: 06/24/2007

The Institutional Review Board for the Protection of Human Subjects in Research (IRB) has reviewed and approved your request for time extension for this study. Approval for this study expires on the date indicated above. At the end of the approval period you will be asked to submit a report with regard to the involvement of human subjects. If your study is still active, you may apply for extension of IRB approval through this office.

Researchers who conduct studies involving human subjects have responsibilities as outlined in the document, *Responsibilities of Directors of Research Studies Involving Human Subjects*. This document is available at <http://www.unh.edu/osr/compliance/IRB.html> or from me.

If you have questions or concerns about your study or this approval, please feel free to contact me at 603-862-2003 or Julie.simpson@unh.edu. Please refer to the IRB # above in all correspondence related to this study. The IRB wishes you success with your research.

For the IRB,


Julie F. Simpson
Manager

cc: File
Brett Vindiguerra

Research Conduct and Compliance Services, Office of Sponsored Research, Service Building,
51 College Road, Durham, NH 03824-3585 * Fax: 603-862-3564