DYADIC WAVELET FEATURES FOR ISOLATED WORD, SPEAKER DEPENDENT SPEECH RECOGNITION

THESIS

Stephen Ainge
Flight Lieutenant, RAAF

AFIT/GE/94M-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/94M-03

DYADIC WAVELET
FEATURES FOR ISOLATED WORD,
SPEAKER DEPENDENT SPEECH RECOGNITION

THESIS

Stephen Ainge
Flight Lieutenant, RAAF

AFIT/GE/94M-03

Approved for public release; distribution unlir ed

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1994	Master's Thesis

4. TITLE AND SUBTITLE
Dyadic Wavelet Features for Isolated Word Speaker Dependent Speech Recognition

5. FUNDING NUMBERS

6. AUTHOR(S)
Stephen Ainge

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/94M-03

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr Cupples
RL/IRAA
32 Hangar Rd
Griffiss AFB NY 13441

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

This research examines the use of dyadic wavelet features for the recognition of speaker dependent isolated word speech. The features were generated using three different wavelet filters – Daubechies 4 coefficient (Db4), Daubechies 20 coefficient (Db20) and a 31 coefficient cubic spline; and three different window lengths – 16ms, 8ms and 4ms. The accuracy of the standard and over-sampled dyadic wavelet methods were compared. The over-sampled dyadic wavelet method using the Db4 scaling function, with a maximum accuracy of 65.5%, was found to be the most accurate of the wavelet methods tested. The accuracy of this over-sampled dyadic Db4 wavelet method was compared to the accuracy of three Fourier feature methods: octave frequency bandwidth, equal bandwidth and Mel scaled bandwidth features. The dyadic wavelet methods did not perform as well as the Fourier methods. The maximum accuracy obtained for the wavelet methods was 65.5%, compared to the maximum accuracy of the octave bandwidth feature Fourier method of 85.6%. The combination of wavelet features and Fourier features was tested. The order of magnitude of the covariance matrices of each set were equalized and the resulting feature vector set classified. It was found that the recognition accuracy of the wavelet plus Fourier feature vectors, 74.0%, was lower than the recognition accuracy of the Fourier-only feature vectors, 84.5%. The inclusion of the wavelet features added information to the system that reduced the recognition effectiveness of the Fourier features.

14. SUBJECT TERMS
Speech Recognition, Pattern Recognition, Wavelets

15. NUMBER OF PAGES
137

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

AFIT/GE/94M-03

DYADIC WAVELET FEATURES FOR ISOLATED WORD, SPEAKER DEPENDENT SPEECH RECOGNITION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Electrical Engineering

Stephen Ainge, B.Eng.

Flight Lieutenant, RAAF

March, 1994

Acknowledgements

I would like to thank my advisor, Captain Dennis Ruck, for his advice and guidance throughout this research effort. It was his enthusiasm for speech recognition that inspired me to undertake research in that field. I would also like to thank Major Greg Warhola for his enthusiasm in the field of wavelets that encouraged me to apply wavelets to the speech recognition problem. Then there is Doctor Steve Rogers' enthusiasm. His enthusiasm is infectious and provided me with motivation when things were not going well.

Finally, and most importantly, I would like to thank my family, my wife Debbie and my children Rebecca, Ben and Nicole. My research effort was more their sacrifice as it was my toil. Without their support and understanding, this research would not have been possible.

Stephen Ainge

Table of Contents

List of Figures

List of Tables

Abstract

This research examines the use of dyadic wavelet features for the recognition of speaker dependent isolated word speech. The features were generated using three different wavelet filters – Daubechies 4 coefficient (Db4), Daubechies 20 coefficient (Db20) and a 31 coefficient cubic spline; and three different window lengths – 16ms, 8ms and 4ms. The accuracy of the standard and over-sampled dyadic wavelet methods were compared. The over-sampled dyadic wavelet method using the Db4 scaling function, with a maximum accuracy of 65.5%, was found to be the most accurate of the wavelet methods tested. The accuracy of this over-sampled dyadic Db4 wavelet method was compared to the accuracy of three Fourier feature methods: octave frequency bandwidth, equal bandwidth and Mel scaled bandwidth features. The dyadic wavelet methods did not perform as well as the Fourier methods. The maximum accuracy obtained for the wavelet methods was 65.5%, compared to the maximum accuracy of the octave bandwidth feature Fourier method of 85.6%. The combination of wavelet features and Fourier features was tested. The order of magnitude of the covariance matrices of each set were equalized and the resulting feature vector set classified. It was found that the recognition accuracy of the wavelet plus Fourier feature vectors, 74.0%, was lower than the recognition accuracy of the Fourier-only feature vectors, 84.5%. The inclusion of the wavelet features added information to the system that reduced the recognition effectiveness of the Fourier features.

DYADIC WAVELET FEATURES FOR ISOLATED WORD, SPEAKER DEPENDENT SPEECH RECOGNITION

I. Introduction

1.1 Background

The accurate recognition of speech is an important area of research. In many cases the most desirable method of interaction between a human user and a machine is speech. An F-16 pilot, in the midst of air-to-air combat, would like to be able to change head-up displays, arm weapons and control communication systems without having to remove his attention from his flying. These functions could be controlled by uttering commands. These speech commands must be accurately recognized by the aircraft control systems. Wavelet analysis is a new area of signal analysis which may lead to improved accuracy in the recognition of speech.

The most popular method of extracting features from speech signals has been the use of Fourier analysis. Fourier analysis decomposes the signal into its frequency components. For example, a continuous square wave can be decomposed into an infinite sum of sinusoids. All naturally occurring signals can be represented by a Fourier decomposition. However, Fourier analysis requires the signal to be processed in large pieces (or windows) in order that good frequency resolution can be achieved. This method uses the Short Time Fourier Transform (STFT). In this method the signal is broken up into overlapping windows. For example a 10 second duration signal may be broken into 19 windows, each of 1 second duration starting on the whole and half second (i.e. 0 - 1, 0.5 - 1.5, 2 - 3, 2.5 - 3.5 . . .). Using the STFT, an accurate representation of the frequency content of each window can be found. However, the accuracy of when each frequency was present can only be determined to within a 0.5 second period. For speech, this time resolution is extremely

poor. Narrowing the window will achieve better time resolution; however, the window length is inversely proportional to the frequency resolution. Therefore, reducing the time window will reduce the frequency resolution. Therefore, the STFT method becomes a trade-off between wide windows for good frequency resolution and narrow windows for good time resolution.

Wavelet analysis, developed in the mid 1980's, provides an alternative method for analyzing speech signals. Wavelet analysis decomposes the signal into levels of different resolutions. Each level is a band-pass filtered version of the original signal. The dyadic method of decomposition, used in this research, produces octave bandwidth versions. That is, at each successive level of decomposition, the low frequency cut-off and the bandwidth is half of the previous level. These levels each provide a different resolution in time and frequency. The first level of decomposition provides high resolution in time for the high frequency components.

Subsequent levels produces lower resolution in time for lower frequency components. Therefore, the resolution provided is directly linked to the frequency band analyzed; fine resolution for fine detail, coarse resolution for coarse detail.

Wavelet analysis would, therefore, seem to be ideal for the analysis of speech signals. Speech signals are comprised of time varying frequency components. It is both the frequency components and the time varying characteristics of the speech signal that will distinguish it from another speech signal. The Fourier transform method of analysis can accurately determine the frequency content. However, the Fourier analysis cannot accurately provide the time varying characteristics. Wavelet analysis can accurately provide both the frequency content and the time varying characteristics.

1.2 Problem Statement

This thesis will investigate the accuracy of a speech recognition system using dyadic wavelet features and will compare this accuracy to the accuracy of the same system using Fourier magnitude features. The wavelet features will be the successive levels of detail coefficients, corresponding to

frequency bands. The Fourier features will be the arithmetic average of the Fourier magnitudes in a given bandwidth.

1.3 Summary of Current Knowledge

The theory of wavelet analysis for signal decomposition, Mallat (5), has not been widely used for the processing of speech signals. One application has been the use of the wavelet transform for pitch detection in speech signals by Kadambe and Boudreaux-Bartels (3). Kadambe and Boudreaux-Bartels reported a maximum of 2% relative error for estimating the pitch period of speech in noise. Sari-Sarraf and Brzakovic (14) classified non-speech, 1-dimensional signals using wavelet analysis. These signals were 1-dimensional signatures obtained by plotting distances from the centroid of an object to its boundary. Sari-Sarraf and Brzakovic reported "highly satisfactory" results. However, these signals were very simplistic, compared to speech signals.

The only evidence of wavelet analysis for speech recognition was a comparison of time-frequency analysis techniques by Reilly and Boashash (12). Reilly and Boashash reported that wavelet analysis performed similarly to other time-frequency analysis techniques in recognizing isolated words from a single speaker database. In all the techniques examined by Reilly and Boashash, the shorter the window length, the better the result. The best results reported were better than 90% for all the techniques. However, Reilly and Boashash did not use the same dyadic decomposition method and feature set that will be used in this research. The minimum feature vector length, used by Reilly and Boashash, was 16. Given that the data set contained speech signals of a maximum length of approximately 16,000 samples, the maximum number of dyadic decomposition levels possible is 14 (i.e. $2^{14} = 16,384$). Reilly and Boashash do not provide any details on the decomposition method and the features that they used.

There is no published research on the use of dyadic wavelet features for speech recognition.

1.4 *Research Objectives*

This thesis will investigate the use of dyadic wavelet features for the recognition of isolated word, speaker dependent speech recognition. The specific objectives of this research effort are as follows:

1. Develop a system that extracts wavelet and Fourier features from the input speech signals, transforms the features into Karhunen-Loève space and classifies the transformed vectors.
2. Compare the system's performance when using the standard and over-sampled wavelet decomposition methods.
3. Compare the system's performance when using different wavelet filters: Daubechies 4 and 20 coefficient and cubic spline filters.
4. Investigate the effect of varying the window length on the system's performance.
5. Investigate the effect, on the system's performance, of augmenting the raw features with simple window features.
6. Compare the system's performance when using wavelet features to the system's performance when using Fourier features.
7. Investigate the system's performance when the feature vector is composed of both wavelet and Fourier features.

1.5 *Approach and Methodology*

This thesis used the TI-46 speech database produced by Texas Instruments. This database contains ten examples of each of the spoken digits ("zero" through "nine") for eight different speakers; four male and four female. The speech recordings were made in a sound proof room, using a microphone positioned to reduce breath effects, and sampled with 12 bit resolution at 12.5kHz.

A brief description of the major processes is given in the following sub-sections.

1.5.1 Wavelet Feature Extraction. Each speech example was decomposed into eleven levels of detail coefficients using the standard dyadic wavelet algorithm. The levels of detail coefficients were expanded, as described in Section 3.4.2. A level is a row of the detail coefficient matrix. The rows of this matrix are a different feature and each column is the feature vector for each point in time, corresponding to the sample time of the original speech example. This decomposition was performed with each of the following filters: the Daubechies 4 coefficient filter,and the Daubechies 20 coefficient filter. The resulting detail coefficient matrix, each column corresponding to a feature vector, was reduced into 16ms windows with 50% overlap. The feature vector for each window was the arithmetic mean of the feature vectors in the window.

The above procedure was repeated using the over-sampled dyadic wavelet algorithm. The over-sampled algorithm allows the analysis to be shift invariant by calculating every component in the detail coefficient matrix; in essence, calculating the standard dyadic wavelet details for every shift of the data. This decomposition was performed with each of the following filters: the Daubechies 4 coefficient filter, the Daubechies 20 coefficient filter, and the cubic spline filter.

The best performing combination of algorithm and filter was then used to repeat the above with window lengths of 8ms and 4ms.

1.5.2 Fourier Analysis. Each speech example was reduced into 16ms windows with 50% overlap. Each window was then Fourier analyzed to produce 2048 Fourier coefficients. The Fourier coefficients were then grouped into 11 octaves. The first octave, first feature coefficient, was the simple arithmetic mean of the last 1024 Fourier coefficients. The next octave, second feature coefficient, was the simple arithmetic mean of the next 512 Fourier coefficients. This process continues until the last octave contains only one Fourier coefficient, the lowest frequency component.

This produces an eleven dimensional feature vector with a similar frequency representation as the wavelet feature vectors.

The above was repeated for window lengths of 8ms and 4ms.

1.5.3 Karhunen-Loève Transform. All the feature vectors, produced by one of the methods for all utterances and for all speakers, were transformed into Karhunen-Loève space. The transformed feature vectors were then used to classify the data. The classification was repeated for one feature, then two features, and so on, until all eleven of the transformed features were retained.

1.5.4 Classification. Two reference utterances are selected from each class, based on the utterance length. This provides 160 references from 80 classes. The classifier is provided with the label of the speaker, reducing the classification to a 10 class problem with 2 references per class.

Each utterance is presented to the Dynamic Time Warp algorithm presented by Parsons (6) and the total cost noted for each of the references. The reference with lowest total cost determines the class assigned to the utterance.

1.6 Thesis Organization

The remainder of this document is organized as follows:

- Chapter II identifies the past and current research in the application of wavelets for speech and 1-dimensional non-speech signals, and past research that was used to perform specific tasks in this research.
- Chapter III describes the experiment conducted.
- Chapter IV presents and analyzes the results of the experiment.
- Chapter V provides conclusions and recommendations from this research effort.
- Appendix A shows the scaling function coefficients used in the experiments.

- Appendix B contains the tabulated results of the experiments.
- Appendices C, D, E and F provide the software listings for the Feature Generation, Karhunen-Loève Transform, Reference Selection, and Classification programs.

1.7 Summary

Wavelet analysis is a new analysis technique that is suited to speech recognition applications. The application of this new technique to the speech recognition problem has not been fully explored in published research. This thesis demonstrates the suitability of the dyadic wavelet analysis technique to the speech recognition problem.

II. Literature Review

2.1 Introduction

This literature review has been based on the following:

1. Journal articles listed in the COMPENDEX PLUS database. The years considered for this review were 1988 to 1992 inclusive. The database was searched on all combinations of the subject keywords: Wavelet(s), Speech, Speech Recognition, and Transient Signal(s).
2. The text: Voice and Speech Processing by Parsons (6).
3. Specific journal articles, outside those parameters given above, for specific tasks required in the experiment.

This literature review will address: a method of isolated word speech recognition using Fourier features, the technique of dyadic wavelet decomposition and some of the more commonly used wavelet filters, 1-dimensional non-speech applications of wavelet transforms, speech applications of wavelet transforms, the Karhunen-Loève transform, dynamic time warping, and speech end point detection.

2.2 Fourier Analysis for Speech Recognition - Pols 1971

In 1971 Pols (8) developed a system for the recognition of a set of 20 Dutch words, with 20 examples of each word. The block diagram of Pols' system is shown in Figure 2.1. Pols' system used 20, $\frac{1}{3}$-octave, bands of Fourier coefficients. These 20 values were used to produce a 17- dimensional feature vector. The three lowest frequency Fourier coefficients were summed and the logarithm of this value used as the first feature. The next two lowest frequency Fourier coefficients were summed and the logarithm of this value used as the second feature. The logarithms of the remaining 15 Fourier coefficients were used as the remaining features.

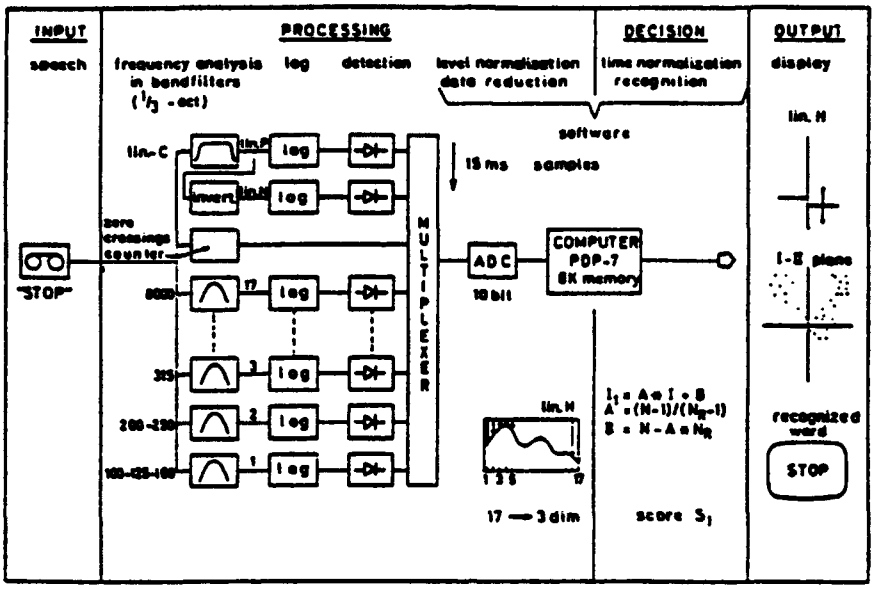

Figure 2.1 Block diagram of Pols' recognition system

Pols calculated the 17 dimensional feature vector, described above, every 15ms. Therefore, there were no overlapping windows. Once the feature vector was determined, the features were level normalized. This was achieved by subtracting each of the features from the logarithm of the largest (positive or negative) overall level. This normalization makes the analysis independent of the loudness of each word. The next step was to reduce the number of features in the feature vectors. This was achieved by a principle components analysis of the covariance matrix generated by the complete test set. Pols used the three highest variance dimensions that resulted from the analysis. This provided him with 78.1% of the total variance. The 17-dimensional feature vectors were then reduced to 3 dimensions, using a principal components analysis.

A reference trace for each of the 20 words was then generated. The 20 examples of a word were linearly time normalized. For example, if the word to be normalized was twice as long as needed, then every second feature vector was removed. Once time normalized, the 20 normalized examples were averaged to produce the reference trace for that word.

The classification of a test trace was determined by measuring the similarity between the test and the references. The highest similarity determined the class of the test. Pols reported a classification accuracy of 98.75%.

The basic method used by Pol's was chosen for this research. The method being: determine the spectral features, reduce the dimension, determine the references, and classify with time normalization.

2.3 Dyadic Wavelet Analysis and Filters

Mallat described the use of wavelets for multiresolution signal decomposition (5). Mallat demonstrated that an orthonormal basis can be built for any finite energy vector space by dilating a scaling function with a coefficient 2^j and translating the resulting function on a grid whose interval is proportional to 2^{-j}. It is the power of 2 dependence that makes this a dyadic decomposition. The scaling function is a user defined function that must:

1. be continuously differentiable;
2. decay 'sufficiently fast' to zero (e.g. finite support); and
3. in the frequency domain, be a low pass filter.

The last of these conditions is satisfied as follows: if $H(\omega)$ is the Fourier transform defined by

$$H(\omega) = \sum_{n=-\infty}^{+\infty} h(n)e^{-jn\omega}$$

with

$$| H(0) | = 1$$

and

$$| H(\omega) |^2 + | H(\omega + \pi) |^2 = 1$$

then H is a low pass filter.

It is the successive application of this filter at successive dilations, that produces successive approximations of the original signal. That is: the original signal is defined as the zeroth level of approximation coefficients with a resolution of 1; the first level of approximation coefficients is the zeroth level of coefficients filtered with a 2^1-dilated $h(n)$ at $\frac{1}{2}$ resolution; the second level of approximation coefficients is the first level of coefficients filtered with a 2^2-dilated $h(n)$ at $\frac{1}{4}$ resolution; and so on. These levels are collectively called the approximation matrix or c matrix. The rows are the successive levels. Using the algorithm above, "the c matrix" can be generated by the following:

$$c_{m,n} = \sum_{k \in Z} c_{m-1,k+2n} h_k \qquad \text{for all } m > 0$$

where Z is the set of all integers, m is the level of decomposition, and n is the coefficient index.

Mallat states that the real interest for a multiresolution representation is the difference, or detail, between successive levels of approximation. This difference information can be found by using the wavelet function. This wavelet function is generated from the scaling function by the following:

$$G(\omega) = e^{-j\omega} \overline{H(\omega + \pi)}$$

and

$$g(n) = (-1)^{1-n} h(1-n)$$

It is the successive application of this filter, at successive dilations, to the approximation levels, that produces the differences between the approximations. That is: the first level of detail coefficients (note: there is no zeroth level of detail coefficients) is the zeroth level of approximation coefficients filtered with a 2^1 dilated $g(n)$ at $\frac{1}{2}$ resolution; the second level of detail coefficients is the first level of approximation coefficients filtered with a 2^2 dilated $g(n)$ at $\frac{1}{4}$ resolution; and so on. These levels are collectively called the detail matrix or d matrix. where the rows are the successive

levels. Using the algorithm above, the d matrix can be generated by the following:

$$d_{m,n} = \sum_{k \in \mathcal{Z}} c_{m-1,k+2n} g_k \qquad \text{for all } m > 0$$

where $\mathcal{Z}$ is the set of all integers, m is the level of decomposition, and n is the coefficient index.

Mallat also showed that the original signal can be perfectly reconstructed from the detail matrix and the lowest resolution approximation coefficients. This is possible since each detail level is the difference between the approximation at that level and the next higher resolution approximation level. That is: Let $A_i f$ be the i^{th} level of approximation of f and let $D_i f$ be the i^{th} level of detail of f. Then

$$A_4 f + D_4 f = A_3 f \qquad \text{and} \qquad A_3 f + D_3 f = A_2 f \qquad \text{and so on}$$

which means

$$A_4 f + D_4 f + D_3 f + D_2 f + D_1 f = A_0 f$$

perfect reconstruction of the original signal.

The method described by Mallat (5) can be extended to produce an over-sampled multiresolution representation, which is discussed by Sari-Sarraf and Brzakovic (14). The multiresolution analysis described by Mallat is not shift invariant. That is; if the original signal was shifted in time, different approximation and detail matrices would be generated. This results because of the 2^j dilation of the jth level. That is each successive level is down-sampled by 2. To overcome this shift invariance, the decomposition must appear to have been performed for all possible shifts of the original signal. To physically perform the decomposition for all possible shifts is computationally impractical, since almost all the coefficients would be recalculated in one or more of the different shifts. Suzuki (17) shows the equations to produce the c and d coefficients, that simulates the decomposition of all possible shifts. These equations only calculate each coefficient once. These

equations are:

$$c_{m,n} = \sum_{k \in Z} c_{m-1,n+2^{m-1}k} h_k \qquad \text{for all } m > 0$$

$$d_{m,n} = \sum_{k \in Z} c_{m-1,n+2^{m-1}k} g_k \qquad \text{for all } m > 0$$

These equations no longer down-sample each successive level. The coefficients are still calculated on a 2^j dilation, thereby the multiresolution representation is preserved. From the stand point of perfect reconstruction, the over-sampled representation contains a lot of redundant information. The standard representation meets the minimum requirement for perfect reconstruction. These extra coefficients are not required. Obviously, the number of computations required to calculate the over-sampled representation is many times that of the standard representation. This is a cost that must be weighed against the benefit of shift invariance.

Having determined the decomposition algorithm, the next decision is that of the choice of filters. Daubechies (1) developed a set of filters of varying length. The most common of these filters vary, in steps of 2, from 4 coefficients to 20 coefficients. Daubechies demonstrated that these filters meet the requirements discussed by Mallat (5).

Another family of filters are the cubic spline wavelets, described by Kadambe and Boudreaux-Bartels (2). A cubic spline wavelet is constructed using H and G filters that meet the requirements discussed by Mallat (5). The requirement for the cubic spline representation of the functions is that it be twice differentiable. These filters can be made to be symmetric or anti-symmetric. This is simply a result of the filters chosen to be approximated by the cubic spline. A real and symmetric $H(\omega)$ produces a complex and anti-symmetrical $G(\omega)$.

There is a countably infinite number of other possible filters. Two of the more popular are the Haar and the Gaussian. The following definitions are taken from Kadambe and Boudreaux-Bartels

(2). The Haar is the most simple of wavelets being defined as:

$$g(n) = \begin{cases} 1 & \text{if } 0 < n < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} < n < 1 \\ 0 & \text{otherwise} \end{cases}$$

The Gaussian is defined as:

$$g(n) = e^{jn\omega_0} \exp\left(-\frac{n^2}{2}\right)$$

where ω_0 is a constant.

2.4 Wavelet Applications for 1-Dimensional Non-Speech Signals

The following are examples of the application of wavelet analysis to 1-dimensional non-speech signals.

Sari-Sarraf and Brzakovic (14) discuss classifying signals with five types of deformities using a shift-invariant MultiScale WAvelet Representation (MSWAR); discussed earlier as the over-sampled representation. The five deformities are described below.

1. Type 1 - A shifted reference signal.
2. Type 2 - A shifted reference signal with a dc offset.
3. Type 3 - A shifted, linearly transformed reference signal.
4. Type 4 - A shifted reference signal plus additive Gaussian noise.
5. Type 5 - A shifted reference signal with local defects.

Signals, with these deformities, were presented to a system comprising of three modules; representation, measurement and classification. The representation module is where the MSWAR was used to decompose the signal into its detail and approximation levels of coefficients. The measurement module is where the distances, between the signal and the reference, to be used in

the classification module were calculated. The classification module is where the decision rules are applied. The particular measurements and decision rules are dependent on the type of deformity.

The Representation Module is the feature extraction module. The standard MultiResolution WAvelet Representation (MRWAR) is not shift-invariant, however, the MSWAR is shift-invariant. This shift-invariance is achieved by using the assumption that the signal is periodic, and therefore, all shifts and convolutions are circular. Then any comparison, between a signal and a shifted version, can be made using this circularity. The cost of this shift-invariance is the increase in the amount of calculations required in the decomposition. An example of the difference is: for a 16 sample signal the MRWAR requires the calculation of 15 coefficients (8+4+2+1), the MSWAR requires the calculation of 64 coefficients (16+16+16+16).

The Measurement Module calculates the distance between the signal and the reference. The distance calculations required are different for each type of deformity. All the calculations are based on the sum of the square difference.

The Classification Module is where the distances are examined and the decision rules applied to determine whether the signal is similar to the reference. The decision rules are different for each type of deformity.

Sari-Sarraf and Brzakovic claim that they have applied this method to "*a variety of reference and observed signals and found the results to be highly satisfactory.*"

Petropulu (7) discusses the ability of the Discrete Wavelet Transform (DWT) to detect transient signals with unknown waveforms and arrival times. The presence of a transient is indicated by a peak in the DWT domain. The location of that peak, relative to the transient, depends on the dilation and translation parameters used. The DWT of an observed signal, that contains multiple transients, will be the sum of the DWT for each transient.

Therefore, each transient, even if they overlap, will appear at positions in the DWT proportional to their appearance in time.

The basic wavelets used by Petropulu are the one-sided exponential windowed sine function and the time limited one-sided exponential windowed sine function. The second type of wavelet allows the detection characteristic to be easily derived since, by selecting an appropriate translation parameter, the DWT coefficients of a zero mean Gaussian process can be made uncorrelated.

The detection statistic is discussed, by Petropulu, for two hypotheses. The first is that the observed sequence was white Gaussian noise. The second is that the observed sequence was a number of transients plus white Gaussian noise. For white Gaussian noise the DWT coefficients have zero mean and a covariance matrix C. For transients the DWT coefficients have non-zero mean. Therefore, Petropulu shows that the likelihood ratio is chi-square for white Gaussian noise only and non-central chi-square for transients plus white Gaussian noise.

Petropulu reported that the DWT was used to analyze six signals: two pairs of transients, of similar form as the basic wavelets, with three different levels of white Gaussian noise. In each case the DWT was able to indicate the transient locations, even with up to $-10dB$ signal to noise ratio and with the transients partially overlapped.

Tuteur (18) describes the detection of abnormalities in electrocardiogram traces using wavelet analysis. Tuteur shows that a wavelet analysis can successfully detect ventricular late potentials (VLP), which are an indicator of life-threatening arrhythmias. The difficultly in detecting these VLPs is that they can be obscured by external interference or muscle artifact noise. The wavelet analysis allows the VLPs to be extracted from the background noise. This appears to be the same type of application that is discussed by Petropulu (7).

Sinha and Tewfik (16) discuss the use of optimized wavelets for the coding of audio signals. The authors claim little degradation in the audio quality of 16 bit 24 Ksamples/sec speech and music signals when they encode them at 8-9 Kbits/sec using the approximate wavelet coefficients.

Sinha and Tewfik claim that the Discrete Wavelet Transform (DWT) offers significant promise for compression of audio signals. Sinha and Tewfik state *It (DWT) provides a good approximation to the Karhunen-Loève Transform, whereby a basis may be chosen to match a given signal.*

Sinha and Tewfik propose two wavelet optimization techniques for audio compression. The first optimization uses a fixed quantization step and optimizes the magnitude of the wavelet transform coefficients. The second optimization quantizes each wavelet transform coefficient using an adaptive scheme. These optimization techniques, without simplification, are highly complex. The authors claim that the second technique proves to be a simpler solution than the first. However, several simplifying assumptions are made. The resulting coefficients are the approximate wavelet coefficients. The authors' test results indicate that these approximate wavelet coefficients lead to highly compressed high quality wavelet representations.

2.5 Wavelet Applications for Speech Signals

The following are examples of the application of wavelet analysis to speech signals. The final article reviewed in this section, Reilly and Boashash (12), is the only article, that was found, that discusses speech recognition using wavelet analysis. Kadambe and Boudreaux-Bartels (3) describe the use of the Dyadic Wavelet Transform (DyWT) to determine the pitch period in speech and compares it to the Cepstrum and Autocorrelation methods.

The DyWT is a discrete wavelet transform that uses a scale parameter that consists of integer powers of 2. The important property of the DyWT for pitch detection in speech is it produces coefficients, whose magnitude have local maxima around points of discontinuity in the analyzed signal or its derivatives. In speech, the glottal closure causes sharp changes in the derivative of the air flow in the glottis and transients. A really sharp change will exhibit local maxima across several dyadic scales. Therefore, this method looks for these local maxima, where they appear in

at least two consecutive scales (levels of decomposition). The time interval between two of these local maxima is then the pitch period.

The authors found, experimentally, that only three levels of decomposition needed to be calculated to determine the pitch period. Which set of three consecutive scales are required is determined by the wavelet used. The authors used a cubic spline wavelet and found they needed scales 2^3, 2^4 and 2^5. This significantly reduced the computational complexity of the analysis.

This method was tested and compared to the Cepstrum and Autocorrelation methods using:

1. synthesized nasal sound /n/,
2. synthesized sound /i/ (as in heed),
3. synthesized voiced fricative /v/ (as in verve),
4. "a" spoken by a male,
5. "the seat is weeping the boat" spoken by an English speaking native male American, and
6. "when the sun light" spoken by an American female.

Kadambe and Boudreaux-Bartels found that the DyWT method, when compared to both the other methods, was *"more accurate, more robust to noise, and no more computationally expensive for determining pitch period and more accurate for segmenting speech into voiced and unvoiced."*

Kadambe and Boudreaux-Bartels (2) also compared the relative performance of the Haar, spline, Gaussian, and minimum phase (e.g. Daubechies) wavelets for the estimation of pitch periods in speech signals. The pitch period was estimated by measuring the time between glottal closures detected by the Dyadic Wavelet Transform (DyWT). The DyWT is a useful tool for the analysis of speech signals since it is linear and capable of detection of sharp and slow global and local variations. The effectiveness of this method of analysis is dependent on the choice of the wavelet function used.

The glottal closures were detected by finding abrupt variation in the speech using a wavelet that is the first derivative of a smoothing function. Then a sharp variation in the speech signal is shown by a local maximum in the DyWT and a slow variation is shown by a local minimum. Therefore the pitch period can be estimated by measuring the time interval between two successive local maxima.

The DyWT pitch detector, proposed in this paper, was tested on: a synthesized voiced fricative /v/, and the continuous speech "the seat is weeping the boat" spoken by an English speaking American male. The results of these tests were:

1. Voiced Fricative /v/. 95% relative accuracy for the Haar wavelet 98% relative accuracy for the spline wavelet 20% relative accuracy for the Gaussian wavelet 90% relative accuracy for the minimum phase wavelet
2. "the seat" spoken. The spline wavelet accurately estimated the pitch period. The Haar, Gaussian, and minimum phase wavelets were not accurate in estimating the pitch period.

This paper therefore concludes that the spline wavelet is superior to the Haar, Gaussian, and minimum phase wavelets for estimating the pitch period of synthetic and spoken continuous speech.

Reilly and Boashash (12) compared several time-frequency signal analysis techniques for speech recognition. The techniques compared are the short time Fourier transform, the auto-regressive model spectrum estimation, reflection coefficients, the wavelet transform, the windowed discrete Wigner-Ville distribution, the Choi-Williams distribution, and the Zhao-Atlas-Marks distribution.

The test set was the seventeen speaker, isolated word, letters and digits database produced by Texas Instruments. However, Reilly and Boashash only used the data for one speaker, a female. Twenty words were used; the ten digits (zero through nine) and ten command words. All words in the database were recorded in a sound proof room using a microphone positioned to minimize breath effects. The recordings were 12 bit resolution sampled at 12.5kHz. The training set contained ten

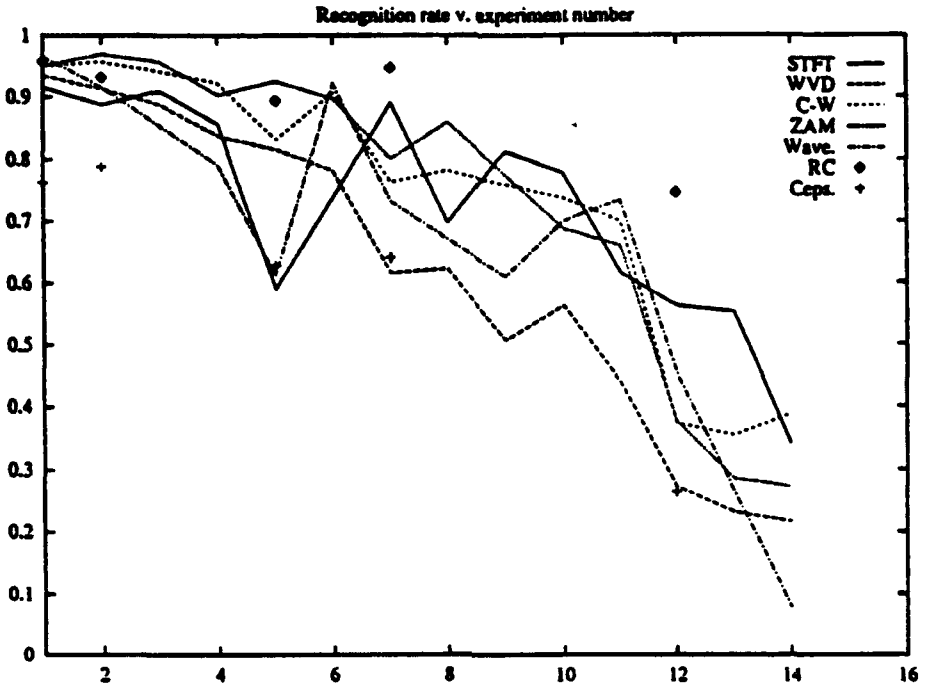

Figure 2.2 Experimental results of Reilly's and Boashash's recognition system

utterances per word and the test set contained sixteen utterances per word. Although, not stated by Reilly and Boashash, it is assumed that this is the TI-46 database.

The method used was to generate feature vectors from each of the training utterances, by the method under test, and present them to a Kohonen self-organizing network. Word models were then produced by forming a histogram of all the quantized feature frames for each word. The feature vectors were then produced from each test utterance and matched to the word models. The closest match determining the class of the utterance. This process was repeated for vector lengths, ranging from 16 features to 64 features, and window lengths, ranging from 2.6ms to 41ms. The results of this testing is shown in Figure 2.2.

Reilly and Boashash concluded:

1. the Fourier technique out-performed the other techniques for larger windows,

2. the auto regressive technique performed consistently well,

3. the Choi-Williams and Zhao-Atlas-Marks techniques out-performed the Fourier technique for smaller windows,

4. the Wigner-Ville technique only performed better than the Fourier technique for small windows, and

5. smaller windows were more successful than larger windows.

The thing to note from Reilly and Boashash is their lack of any detail regarding the wavelet technique and the features used. The database used, assuming it was the TI-46 database, contains utterances with a maximum length of 16,000 samples. A dyadic representation of these signal can have, at most, 14 levels.

$$\text{number of levels} = \frac{\log(\text{length})}{\log(2)}$$

Therefore, the 16, 32, and 64 features that Reilly and Boashash used, must have been:

1. different features, and/or

2. generated by some other method

than those used in this thesis.

2.6 Karhunen-Loève Transform

The Karhunen-Loève transform was used as the method of dimension reduction in this thesis. This method was chosen simply because it provides a reliable dimension reduction and the code necessary for its implementation was readily available. Parsons (6) discusses how the Karhunen-Loève transformation does not necessarily produce features that maximize class separation. Maximum class separability is discussed by Parsons (6), Shartle (15) and Keller (4). However, these techniques were not considered in this research. The following is a summary of the Karhunen- Loève transformation method described in Parsons (6).

The Karhunen-Loève transform is a coordinate transform that uncorrelates the original features, producing uncorrelated transformed features. Parsons emphasizes that this method does not necessarily improve class separability. The covariance matrix $\mathcal{W}$ of a set of features contains the variance of each feature, the diagonal components, and the covariance between the features, the off-diagonal components. If $\mathcal{W}$ can be transformed such that the off-diagonal components became zero, then the transformed features would be uncorrelated. This transformation can be accomplished by rotating the coordinate system. If

$$\mathcal{A}^T \mathcal{W} \mathcal{A} = \text{diag}(\lambda_1 , \lambda_2 , \ldots , \lambda_n) = \Lambda$$

Then $\mathcal{A}$ is the transformation matrix that performs the desired rotation, and the λ_i are the transformed feature variances. This is achieved by finding the eigenvectors and eigenvalues of the covariance matrix. The eigenvalues being the variances of the transformed features (λ_i) and the eigenvectors (a_i) form the transformation matrix $\mathcal{A}$.

$$\mathcal{A} = [a_1 , a_2 , \ldots , a_n]$$

Software routines from *Numerical Recipes in C* (9) were used, in this research, to find the eigenvalues and eigenvectors.

2.7 Dynamic Time Warping

There were several methods of classification considered for this research: Dynamic Time Warp (DTW), Hidden Markov Model (HMM), and Neural Network (NN). The decision as to which method to use was relatively easy. HMMs and NNs require extensive training time before any classification takes place (10)(13). Dynamic Time Warp, on the other hand, immediately begins classifying. Since this research focuses on the comparison of different feature sets and not on the

optimal classification technique, the training time required for HMMs and NNs was considered unnecessary. The DTW method used in this research is that given by Parsons (6), since the necessary code was readily available. The following is a summary of the Dynamic Time Warp method given in Parsons.

Dynamic time warping is the nonlinear time normalization of time-varying features. The inputs to the warping process are two sequences generated by time-varying functions,

$$\mathcal{P} = p_1 , p_2 , \ldots , p_m \qquad \text{and}$$

$$\mathcal{Q} = q_1 , q_2 , \ldots , q_n \qquad .$$

One of these functions is the reference function. The other is the test function. These two functions form the two dimensions of a lattice. The warping process determines a minimum cost path through the lattice of points.

The process of finding the minimum cost path through the lattice is accomplished by calculating the difference, or cost, between the two corresponding function values at each point in the lattice. The minimum cost path is the path through the lattice which has the minimum accumulated cost. The number of possible paths is limited by the following constraints:

1. The path must be monotonic.
2. The path must match the endpoints of the two functions. That is, the path must pass through $[p_1, q_1]$ and $[p_m, q_n]$.
3. The path cannot skip any points.
4. The path must stay within a defined boundary within the lattice, usually a parallelogram. This limits the maximum amount of warp permissible.

It is the second of these constraints that provides a measure of the difference between the test and reference functions. The total accumulated cost of the minimum cost path at the point $[p_m, q_n]$ is the cost of classifying the test as being the same as the reference.

More details of the DTW algorithm are given in the following chapter.

2.8 Speech End Point Detection - Rabiner and Sambur 1975

Rabiner and Sambur (11) discuss a simple method of finding the start and stop locations of speech in noise. This method is based on two simple measurements, energy and zero crossing rate.

The first tasks in this method are to determine the statistics of the background and the peak energy in the recording. This method assumes that the first 100ms of the recording interval contains no speech. From these statistics, a zero crossing threshold and two energy thresholds, upper and lower, are set. The method's first guess at the endpoint uses the energy thresholds. The start point has energy that exceeds the lower threshold, then continues to increase until it exceeds the upper threshold, before falling below the lower threshold. The end point is guessed the same way, except that the testing starts from the end of the recording and progresses back toward the beginning. Rabiner and Sambur claim that these first guess end points are conservative and therefore the true endpoints lie outside these first guesses. The true end points are then estimated by examining 250ms on the earlier side of the start point guess and 250ms later than the end point guess, determining the number of intervals that exceed the zero crossing rate threshold. If this threshold is exceeded 3 or more times then the endpoints are set to the earliest, for the start point, or latest, for the end point, that the threshold was exceeded.

Rabiner and Sambur claim that this algorithm made no gross errors when tested using a 54 word vocabulary read by two males and two females. They did report some small errors, but these did not effect human recognition. More significantly for this research, Rabiner and Sambur

reported an essentially error free test of 10 speakers each repeating the ten digits from zero to nine in 10 separate sessions (a similar database to that used in this research).

2.9 Conclusions

Only one example of documented research, Reilly and Boashash (12), in the area of this thesis was found. This research used a method of feature extraction that produced many more features than that used in this thesis. This thesis uses the dyadic wavelet decomposition method to extract frequency band features. There being only one example of documented research in this area demonstrates how little is known of dyadic wavelet features being used for isolated speech recognition.

It is this lack of knowledge that provided the motivation for this thesis. The following chapter describes the experiments used in this thesis to demonstrate the suitability of the dyadic wavelet decomposition method to speaker dependent isolated word speech recognition.

III. Experiment

3.1 Introduction

The experiment designed for this research was isolated word speaker dependent recognition. The basic method used was to extract features from the utterances, reduce the dimensionality of the features, and classify the resulting features. The focus of this research was to show the effect of the feature extraction method on the classification rate. The classification rate of each of the following feature extraction methods was calculated.

1. standard dyadic wavelet decomposition using:
   (a) Daubechies 4 coefficient (Db4) scaling function and
   (b) Daubechies 20 coefficient (Db20) scaling function;
2. over-sampled dyadic wavelet decomposition using:
   (a) Db4 scaling function,
   (b) Db20 scaling function and
   (c) cubic spline scaling function;
3. Fourier decomposition using:
   (a) octave width frequency bands,
   (b) equal width frequency bands and
   (c) Mel bands.

The choice of Daubechies and cubic spline scaling functions for this research was based on the findings of Kadambe and Boudreaux-Bartels (2). Kadambe and Boudreaux-Bartels found that the spline and Daubechies wavelets performed well (better than 90% accuracy) for the estimation of the pitch period in speech. These scaling functions have, therefore, been shown to have had success for

a speech application. The choice of the Db4 and Db20 scaling functions, from all the Daubechies scaling functions, was purely to show the effect of filter length on the recognition accuracy.

3.2 Test Data

The test data set was a database of isolated spoken words produced by Texas Instruments, the TI-46 database. The database consisted of four men and four women each providing ten examples of the ten spoken digits, zero through nine. This provided a total of 800 utterances with 80 classes. This research assumes that the identity of the speaker is known. Therefore, the test set was eight subsets of 100 utterances with 10 classes.

Each utterance in the test set was recorded in a sound proof room with a microphone positioned to reduce breath noise. Therefore, the data was essentially clean speech. The utterances were provided in 12 bit resolution at a sampling rate of 12.5kHz.

3.3 System Algorithm

The total system was split into four logical blocks. These blocks were as follows:

1. Feature Extraction. The feature extraction process decomposes the speech data into feature vectors, representing windows of speech.
2. Karhunen-Loève Transform. The raw feature vectors are transformed into Karhunen-Loève space.
3. Reference Selection. The reference, or template, utterances of transformed feature vectors are determined.
4. Classification. All the transformed utterances are compared to the reference utterances. The reference utterance found to be closest to the test utterance determines the utterance class.

A more detailed explanation of each of the four processes is given below.

3.4 *Feature Generation*

The feature extraction process takes the stored speech utterances and produces the raw feature vectors. The raw features could be of three types: standard wavelet, over-sampled wavelet, and Fourier analysis. The "C" program is contained in Appendix C.

The speech data provided in the TI-46 database must first be converted from 12 bit signed two's complement to floating point. Each utterance is pared to remove the non-speech periods at the beginning and the end of the recording, using the end point detection algorithm of Rabiner and Sambur (11). The analysis window length, either 16ms, 8ms, or 4ms, and the window overlap, 50%, determines the number of analysis windows. Each of these analysis windows is decomposed by the technique selected.

3.4.1 Fourier Features. For each window, in the utterance, a Hamming windowing function is applied and the Fourier coefficients calculated (256 for the Mel scale features, 4096 otherwise). The Fourier coefficients are then grouped into bins according to the grouping scheme to be tested. The three schemes tested in this research were: 11 single octave frequency bands, 11 equal width frequency bands, and 19 equal width Mel bands. The Mel band scheme was based on the method used by Pols (8). The conversion from frequency to Mels is given, from Parsons (6), by:

$$\text{Mels} = \frac{1000}{\log 2} \log(1 + \frac{\text{freq}}{1000})$$

The Mel band features are further modified by summing the lowest 3 bands to produce the first feature. This process results in 17 Mel features. These features may be augmented by additional features. An augmenting feature was used in some of the tests. This extra feature was the number of zero crossings within the window.

3.4.2 Wavelet Features. The method of feature generation for the standard and oversampled wavelet methods are very similar. Therefore, both methods will discussed together with the differences mentioned when necessary.

The pared utterance is padded with zeroes, to a length of the next power plus one of 2 (i.e. if $2^{n-1} < \text{length} < 2^n$ then the length is padded to 2^{n+1}). Zero padding prevents any possibility of the scaling and wavelet functions overlapping the beginning and the end of the utterance during the convolution. Overlap is possible, if the number of zeroes at the end of the utterance is less than the length of the functions, because of the circular convolution process. The scaling function is defined, and the wavelet function is then generated from it by:

$$g(n) = (-1)^{M-1-n} h(M-1-n) \qquad \text{for } 0 \leq n < M$$

where M is the length of $h(n)$. The $h(n)$ and $g(n)$ functions are energy centered, to the closest coefficient. That is: equal, or close to equal, energy in the function lay on either side of the origin. Energy centering is important to keep time equivalence between the successive levels of decomposition (17). The energy centered h and g functions are then used to generate 11 levels of details, the d matrix. Only 11 levels are used because the dyadic decomposition produces approximately octave bands. Therefore the 11^{th} level contains, approximately, frequencies between $\frac{\text{sampling rate}}{2^{12}} = 3\text{Hz}$ and $\frac{\text{sampling rate}}{2^{11}} = 6\text{Hz}$. This frequency band is sufficiently low and narrow to be considered the lower bound for this research.

For the standard wavelet decomposition, the basic convolution equations are:

$$c_{m,n} = \sum_{k=0}^{M-1} c_{m-1,k+2n} h_k$$

$$d_{m,n} = \sum_{k=0}^{M-1} c_{m-1,k+2n} g_k$$

where $1 \leq m \leq 11$ is the level of decomposition, $0 \leq n < \frac{N}{2^m}$ is the coefficient number, and M is the length of the functions. This produces a d matrix with calculated elements in the first $\frac{N}{2^m}$ elements of each row. N is the length of the padded utterance. For example: the first row has calculated elements in the first half of the row, the second row has calculated elements in the first quarter of the row, etc.. An example of this scheme is shown below.

$$\text{for signal samples} = [x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7]$$

$$\text{detail matrix} = \begin{bmatrix} d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} & 0 & 0 & 0 & 0 \\ d_{2,0} & d_{2,1} & 0 & 0 & 0 & 0 & 0 & 0 \\ d_{3,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

To allow each column to represent a feature vector for each point in time, the d coefficients must be expanded to fill the matrix. This is achieved by repeating each coefficient 2^m times. Shown below, for the above example.

$$\text{detail matrix} = \begin{bmatrix} d_{1,0} & d_{1,0} & d_{1,1} & d_{1,1} & d_{1,2} & d_{1,2} & d_{1,3} & d_{1,3} \\ d_{2,0} & d_{2,0} & d_{2,0} & d_{2,0} & d_{2,1} & d_{2,1} & d_{2,1} & d_{2,1} \\ d_{3,0} & d_{3,0} & d_{3,0} & d_{3,0} & d_{3,0} & d_{3,0} & d_{3,0} & d_{3,0} \end{bmatrix}$$

For the over-sampled wavelet decomposition, the basic convolution equations are:

$$c_{m,n} = \sum_{k=0}^{M-1} c_{m-1,n+2^{m-1}k} h_k$$

$$d_{m,n} = \sum_{k=0}^{M-1} c_{m-1,n+2^{m-1}k} g_k$$

where $1 \leq m \leq 11$ is the level of decomposition, $0 \leq n < N$ is the coefficient number, N is the padded length of the utterance, and M is the length of the functions. This produces a full d matrix.

Each column of the d matrix is a feature vector. The d matrix is broken into windows (groups of columns) and a Hamming function applied to each row in the window. Since each column represents one unit of time ($\frac{1}{\text{sampling rate}}$) the number of columns in the window is easily determined. The feature vector for each window is the arithmetic mean of the Hamming window functioned feature vectors in it.

The Hamming window is a method of reducing the frequency distortion caused by analyzing a small segment of a longer signal. Without modifying the sample magnitudes in a window, the signal has effectively been convolved with a rectangle function. In the frequency domain, the window's true frequency response is multiplied by a sinc function. This causes distortion of the true frequency response. The Hamming window is a half cycle, cosine function. The resulting frequency response, when using the Hamming window instead of the rectangle window, is less distorted. This produces a "truer" frequency representation of the signal within the window.

3.5 Karhunen-Loève Transformation

The Karhunen-Loève transform process takes the raw feature vectors and transforms them into the Karhunen-Loève space and produces new reduced dimension feature vectors. The "C" program is contained in Appendix D. The Karhunen-Loève space is a space whose dimensions are those of the uncorrelated features calculated from the raw features. The reduction is based on the magnitude of the variance of these new features. Only the n largest variance new features are retained in the new n-dimension feature vectors. Separate tests were conducted for reduced dimensions 1 through to the number of raw features.

The Karhunen-Loève transformation is achieved by first calculating the mean feature vector. This mean feature vector is used to statistically center the feature vectors while the covariance matrix is calculated. The eigenvalues and eigenvectors of the covariance matrix are determined. The eigenvalues and corresponding eigenvectors are sorted into descending eigenvalue order. The

number of dimensions required to be kept, n, is defined. The first n sorted eigenvectors are used to transform the raw features into the Karhunen-Loève space.

It is here where, individually, each test utterance can be shown to be, essentially, independent from the complete test set. If a single utterance is removed from the test set, the resulting eigenvalues and eigenvectors are the same as those generated from the complete test set. This is possible because there are 800 utterances in the test set. This independence is an important factor in the accuracy of the classification results. Essentially, the utterances, individually, have not been seen by the system, except for the reference utterances (discussed next), prior to classification.

3.6 Reference Selection

The reference selection program produces a script file that will copy two transformed utterances per class to a references directory, producing a total of 160 references. The "C" program is contained in Appendix E. These two utterances per class will be used as the references for that class in the classification process. The sole basis on which the references are selected is length. The reference utterance lengths are set to be:

1. the longest in the class shorter than 75% of the mean, and
2. the shortest in the class longer than 125% of the mean.

The choice of these parameters was to ensure that any given utterance would be within a factor of two of the length of at least one of the references. An ideal reference would contain the essence of every possible example from within its class. An example would be the mean utterance. Therefore, every test utterance, within its class, would be close to the reference. However, in this case no such reference is determined. There is a possibility that one reference chosen from a class may be a unique example of that class. The chance of two references chosen from a class being both unique examples is less. Therefore, the more references chosen from each class the better the chance to eliminate the unique examples. On the other hand, the system's generality relies on testing against

as many examples as possible. With only ten examples per class, two references was considered the best compromise. Given that two references were to be used, the selection criteria above was chosen.

For the test set used, and with 16ms 50% overlapped windows, the mean utterance length is approximately 80. Therefore, the first reference per class is the longest utterance in that class that is shorter than 60, and the second reference is the shortest utterance in that class longer than 100. If no utterance in the class meets one of the criteria then the reference is the next closest utterance.

3.7 Dynamic Time Warp Classification

The dynamic time warp classification process takes the set of transformed utterances and classifies each of them as belonging to one of ten classes, given the identity of the speaker. The "C" program is contained in Appendix F. The basic program was converted from the Fortran dynamic time warp program, "warp", in Parsons (6).

The classification of a test utterance is achieved by finding the cost of the minimum accumulated cost path through a lattice of points for each reference for that speaker. Each point is the cost, or difference, between the corresponding vectors in a reference and a test utterance. The class of the reference which has the minimum cost is the class allocated to the test utterance. The classification accuracy is determined by counting the number of correct classifications, subtracting the number of reference utterances, and dividing the result by the total number utterances less the number of references and expressing this result as a percentage.

$$\text{classification accuracy} = \frac{(\text{number correct} - \text{number of references}) \times 100}{\text{total number of utterances} - \text{number of references}}$$

The method of obtaining the minimum cost path has two major components: the limit on the possible minimum cost paths, and the actual cost calculation and accumulation. The limits on the possible minimum cost paths, in this algorithm, are as follows.

1. The path must stay within the parallelogram described by

$$\text{for coordinates } x \text{ and } y \qquad \text{lower limit } \leq y \leq \text{upper limit} \qquad \text{for all } x$$

$$\text{where} \qquad \text{lower limit} = \max(1, x - \text{offset})$$

$$\text{and} \qquad \text{upper limit} = \min(n, x + \text{offset})$$

$$\text{and} \qquad \text{offset} = \max(50, |n - m|)$$

$$\text{and} \qquad \text{n, m} = \ \# \text{ of vectors in reference and test}$$

An example of this parallelogram is shown in Figure 3.1.

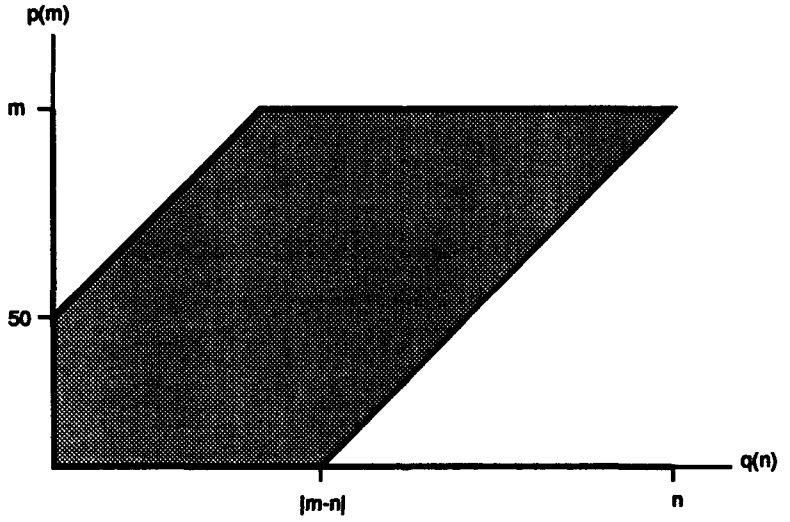

Figure 3.1 Example of dynamic time warp parallelogram with $n > m$

2. The preceding point, predecessor, in the path must have coordinates less than or equal than those of that point. For example, the predecessor to coordinate $(4,5)$ can only be $(4,4)$, $(3,5)$ or $(3,4)$, shown in Figure 3.2.

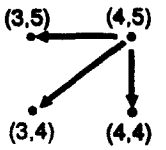

Figure 3.2 Example of valid dynamic time warp predecessors

The actual cost calculation and accumulation is achieved by, commencing at coordinate $(1,1)$, calculating the Euclidean distance between the corresponding test vector and each valid reference vector and accumulating that cost, at each point, for its minimum cost predecessor. This process is repeated until all the test vectors have been compared to all the valid reference vectors.

The accumulated cost at the point (m,n) is the minimum cost.

3.8 Experiment Runs

The above system was used to produce results for the experiments shown in Table 3.1.

Table 3.1 Experiment Parameters

Experiment Number	*Analysis Type*	*Augmented*		*Filter*		*Window Length*
		Zero Xing	*Energy*	*Type*	*Length*	
1	standard wavelet	Yes	Yes	Daubechies	4	16ms
2	standard wavelet	Yes	Yes	Daubechies	20	16ms
3	over-sampled wavelet	Yes	Yes	Daubechies	4	16ms
4	over-sampled wavelet	Yes	Yes	Daubechies	20	16ms
5	over-sampled wavelet	Yes	Yes	Cubic Spline	31	16ms
6	over-sampled wavelet	Yes	No	Daubechies	4	16ms
7	over-sampled wavelet	No	No	Daubechies	4	16ms
8	over-sampled wavelet	No	No	Daubechies	4	8ms
9	over-sampled wavelet	No	No	Daubechies	4	4ms
10	over-sampled wavelet	Yes	Yes	Daubechies	4	4ms
11	over-sampled wavelet	Yes(x4)	Yes(x4)	Daubechies	4	4ms
12	Fourier 11 octave	No	No	n/a	n/a	16ms
13	Fourier 11 octave	No	No	n/a	n/a	8ms
14	Fourier 11 octave	No	No	n/a	n/a	4ms
15	Fourier 11 octave	Yes	No	n/a	n/a	4ms
16	Fourier 11 equal	No	No	n/a	n/a	4ms
17	Fourier 17 Mel	Yes	No	n/a	n/a	4ms
18	Fourier 11 octave and over-sampled wavelet	No No	No No	n/a Daubechies	n/a 4	4ms 4ms
19	Fourier 11 octave(x10) and over-sampled wavelet	No No	No No	n/a Daubechies	n/a 4	4ms 4ms

3.9 Fourier plus Wavelet Feature Vectors

In order to use feature vectors with Fourier coefficients and wavelet coefficients, the covariances of the coefficients produced by each method must be of a similar magnitude. If one method produces covariances of many magnitudes different to the other, then it will totally dominate the classification. Therefore, to "equalize" the covariances of each method, the coefficients of one of the methods was scaled by the ratio magnitudes of the covariances of both methods. It was found that the magnitude of the largest covariance in the covariance matrix of the Fourier coefficients

(Experiment 14) was, approximately, 10^{-9} times that of the largest of the wavelet covariances (Experiment 9). In order to "equalize" the covariances, the Fourier coefficients were scaled by 10^{9} for Experiment 18 and by 10^{10} for Experiment 19.

3.10 Conclusion

This chapter described the experiments that were used in this thesis to demonstrate the suitability of the dyadic wavelet decomposition method to speaker dependent isolated word speech recognition. The following chapter presents the results of these experiments.

IV. Results and Analysis

4.1 Wavelet Features

4.1.1 Standard Vs Over-sampled Wavelet Features. Figure 4.1 shows the system's classification accuracy using the standard, and over-sampled dyadic wavelet decomposition methods, Experiments 1 and 3. Both used zero crossing and energy augmentation, the Daubechies 4 coefficient scaling function (Db4), and a 16ms window length. It can be seen directly in Figure 4.1, that the over-sampled decomposition method clearly out-performs the standard decomposition method for these parameters. The peak performance for the over-sampled method, 65.5%, occurred when 2 KLT coefficients were used. This peak is 36% higher than the peak of the standard wavelet method, 48.1%, which occurred when 7 KLT coefficients were used.

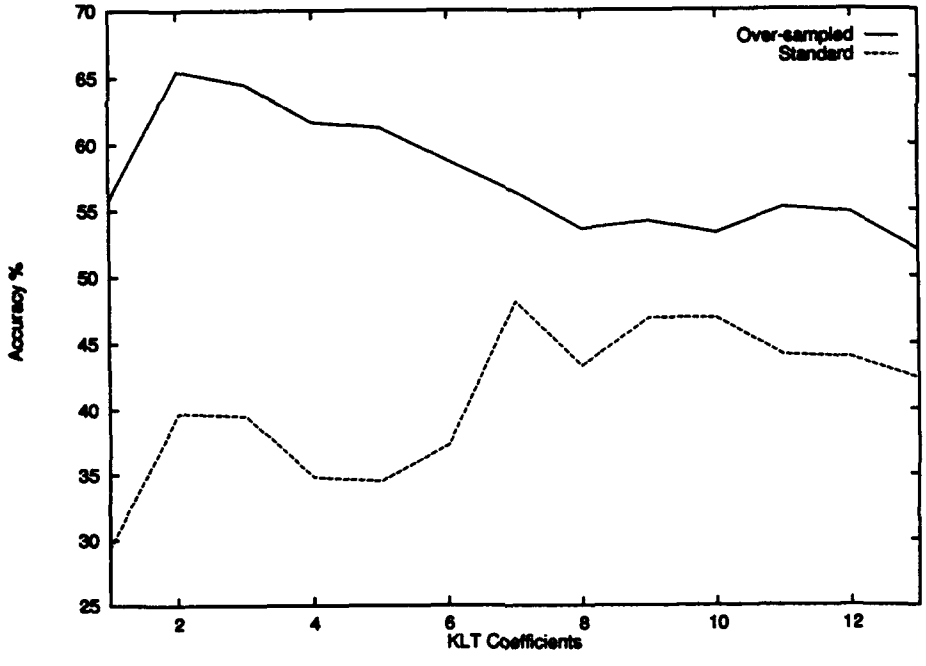

Figure 4.1 Classification accuracy using standard and over-sampled dyadic wavelet decomposition using Db4, augmentation, and 16ms windows (Experiments 1 and 3)

The relative amount of variance retained in classifying the data, in Experiments 1 and 3, is shown in Figure 4.2. Here it can be seen that the Karhunen-Loève transformation produces larger maximum variance features for the over-sampled decomposition method. This shows that the over-sampled representation is more compact than the standard representation, in this case.

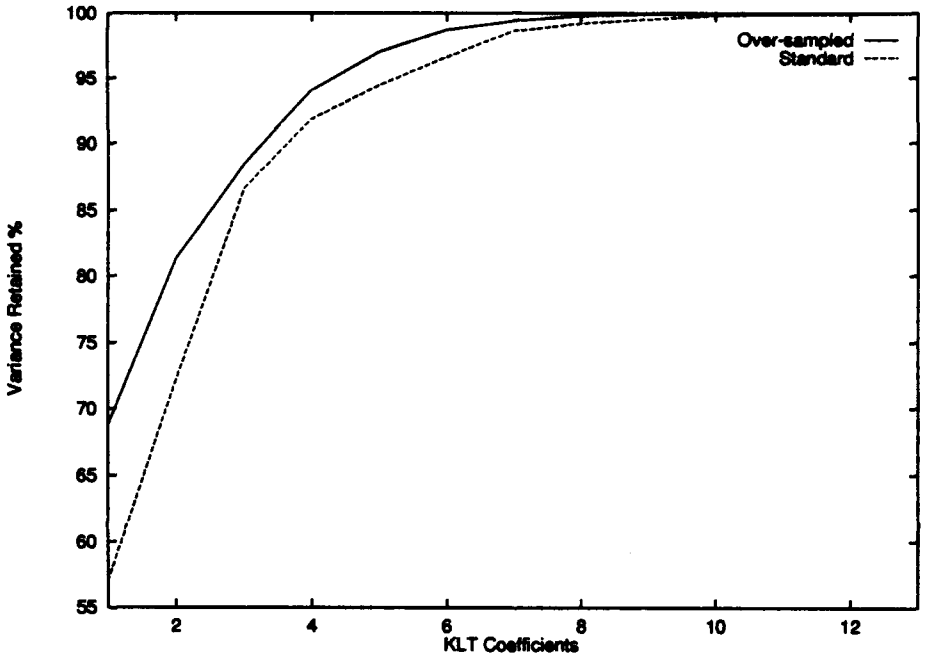

Figure 4.2 Relative variance retained for Experiments 1 and 3

Figure 4.3 shows the system's classification accuracy using the standard, and over-sampled dyadic wavelet decomposition methods, Experiments 2 and 4. Both used zero crossing and energy augmentation, the Daubechies 20 coefficient scaling function (Db20), and a 16ms window length. It can be seen in Figure 4.3, that the over-sampled decomposition method out-performs the standard decomposition method for these parameters. This better performance is not as clear cut as in the case of the Db4 scaling function. The standard method does perform better than the over-sampled method when more coefficients are retained. However, the peak performance of the over-sampled method, 65.5% at 2 KLT coefficients, is 5% better than the peak of standard method, 62.3% at 6 KLT coefficients.

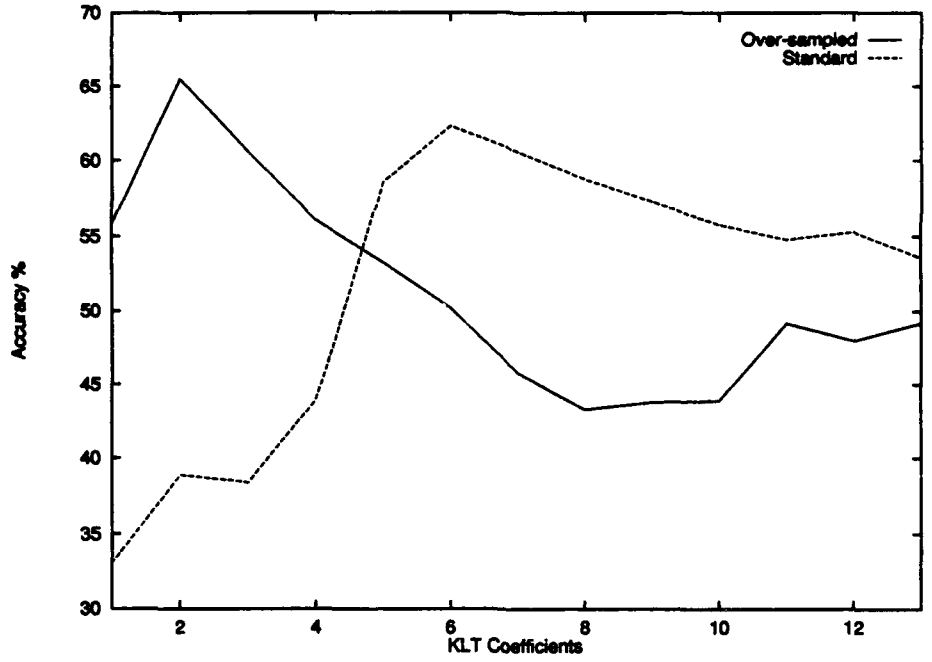

Figure 4.3 Classification accuracy using standard and over-sampled dyadic wavelet decomposition using Db20, augmentation, and 16ms windows (Experiments 2 and 4)

The relative amount of variance retained in classifying the data, in Experiments 2 and 4, is shown in Figure 4.4. Here it can also be seen that the Karhunen-Loève transformation produces larger maximum variance features for the over-sampled decomposition method; more compact representation. Interestingly, the maximum separation between these plots is when the over-sampled method is performing better than the standard method; i.e. before 5 KLT coefficients retained.

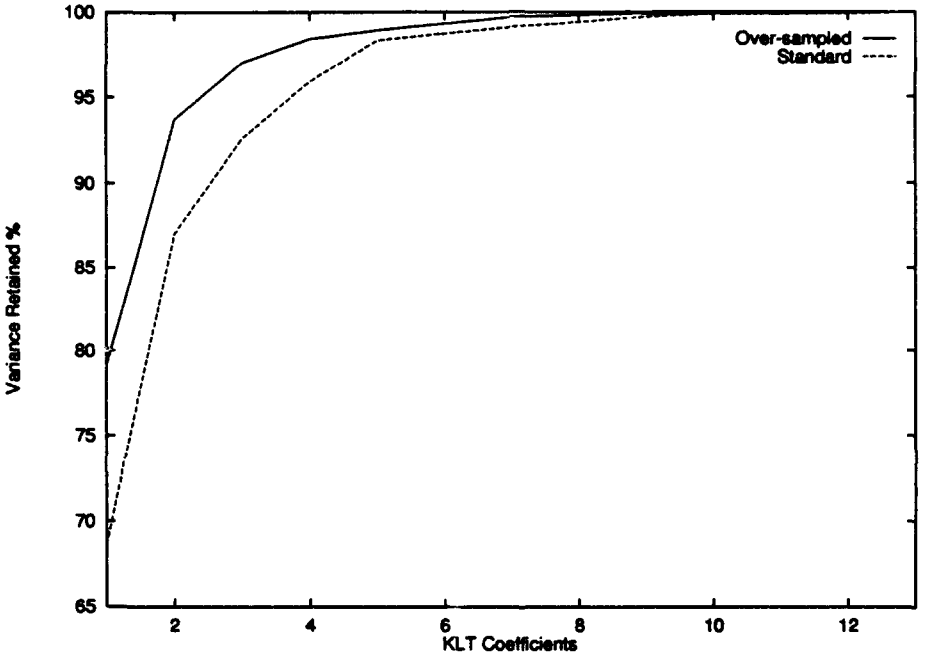

Figure 4.4 Relative variance retained for Experiments 2 and 4

Based on the results presented in this section, the over-sampled dyadic wavelet decomposition was used as the "best" wavelet method for the remaining experiments.

4.1.2 Comparison of Scaling Functions. Figure 4.5 shows the system's classification accuracy using the over-sampled dyadic wavelet decomposition methods with three types of scaling function, Experiments 3, 4 and 5. The three scaling functions used were the Db4, Db20, and a 31 coefficient symmetric cubic spline. The coefficients of these functions are given in Appendix A. Each decomposition uses zero crossing and energy augmentation and a 16ms window length. It can be seen in Figure 4.5, that the Db4 function produces more accurate results than the cubic spline, which is more accurate than the Db20. The peak performance, using each type of scaling function, occurred when using 2 KLT coefficients. That peak was 65.5% for both Daubechies functions and 65.3% for the cubic spline function. The performance falls off faster for the cubic spline than the Db4, and even faster for the Db20 scaling function.

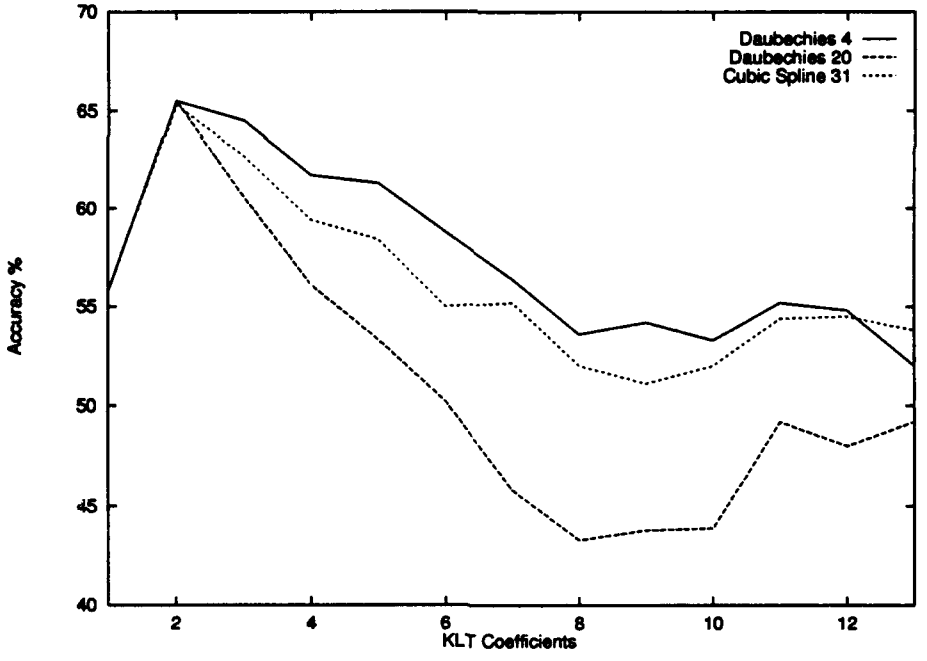

Figure 4.5 Classification accuracy of over-sampled dyadic wavelet decomposition using Db4, Db20, and Cubic Spline; with augmentation and 16ms windows (Experiments 3, 4 and 5)

Figure 4.6 shows the relative amount of variance retained in classifying the data in Experiments 3, 4 and 5. This figure shows the opposite of the trend noticed in the previous experiments. In this case the larger the maximum variance features, the worse the performance. This plot shows that the Db20 decomposition is a more compact representation of the data than the cubic spline representation, which is more compact than the Db4 representation.

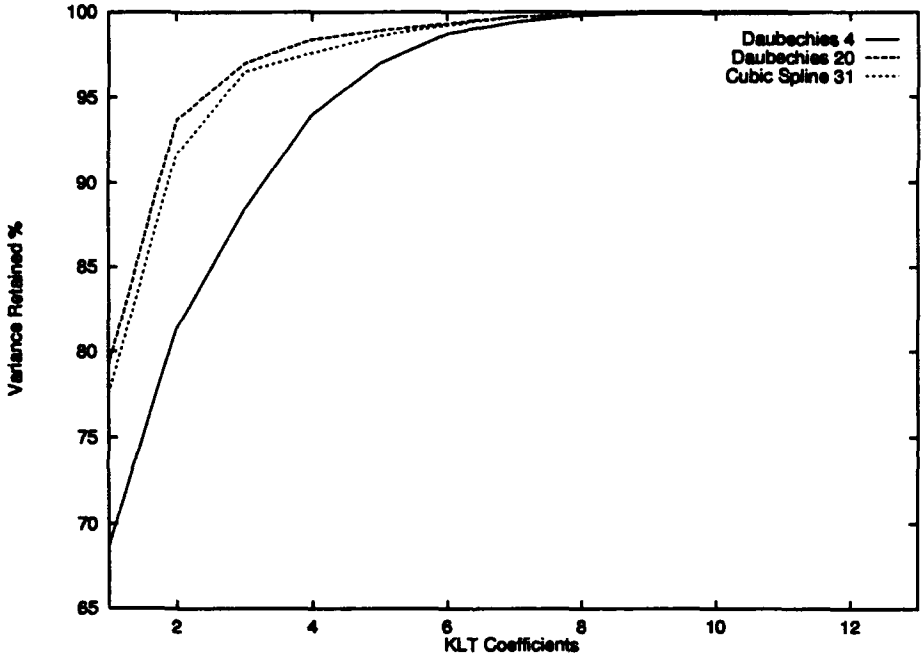

Figure 4.6 Relative variance retained for Experiments 3, 4 and 5

Based on the results presented in this section, the Db4 scaling function was used as the "best" scaling function for the remaining experiments.

4.1.3 Effect of Augmentation. Figure 4.7 shows the system's classification accuracy using the over-sampled dyadic wavelet decomposition methods with:

1. zero crossing and energy augmentation (Experiment 3),
2. zero crossing only augmentation (Experiment 6), and
3. no augmentation (Experiment 7).

Each decomposition uses the Db4 scaling function, and a 16ms window length. It can be seen in Figure 4.7, that the unaugmented features were significantly improved by adding the zero crossing data to the feature vectors. The addition of the energy data produced a small improvement, when compared to the effect of the zero crossing data. Peak performance, of 65.5%, occurred when using 2 KLT coefficients and both augmenting features. This was 12% better than zero crossing augmentation only, 58.3% at 1 KLT coefficient, and 74.7% better than no augmenting features, 37.5% at 9 KLT coefficients.

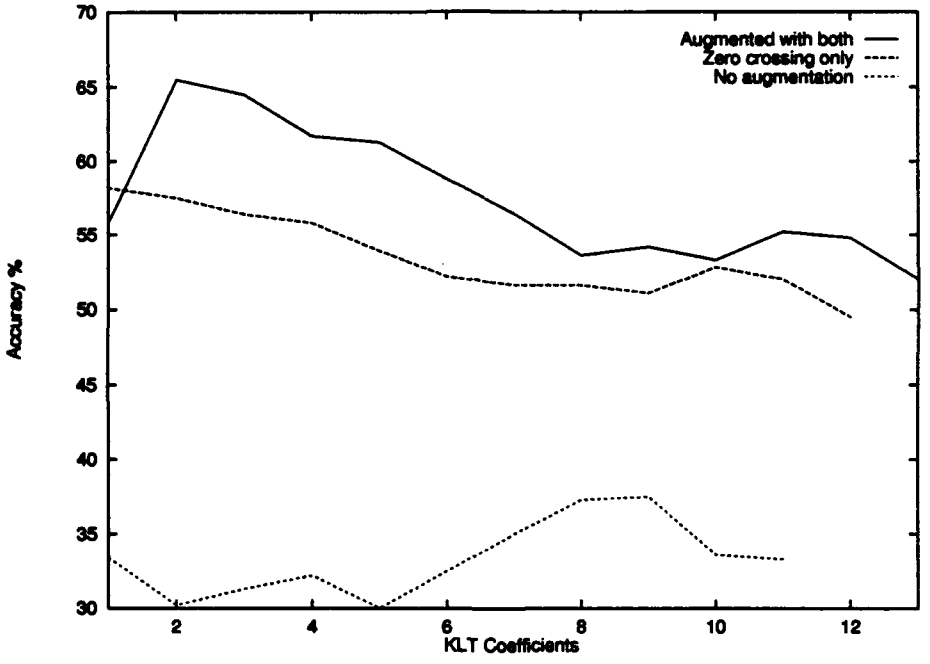

Figure 4.7 Classification accuracy of over-sampled dyadic wavelet decomposition using Db4; with and without augmentation and with 16ms windows (Experiments 3, 6 and 7)

Figure 4.8 shows the relative variance retained when classifying the data in Experiments 3, 6 and 7. This figure shows that the addition of the zero crossing data produces a more compact representation of the data than the unaugmented features. However, adding an additional feature did not produce an even more compact representation. In fact, the addition of the energy data reduced the compactness of the representation.

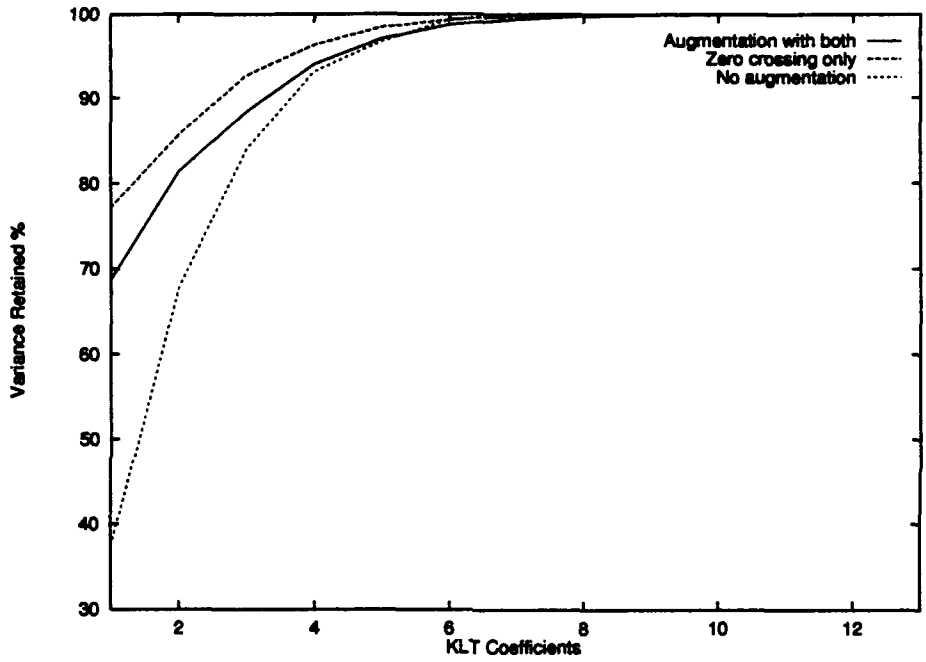

Figure 4.8 Relative variance retained for Experiments 3, 6 and 7

Based on the results presented in this section, the addition of augmenting features can improve the performance of a wavelet based classification system.

4.1.4 Effect of Reduced Window Size. Figure 4.9 shows the system's classification accuracy using the over-sampled dyadic wavelet decomposition method with the Db4 scaling function, no augmenting features, and with:

1. a 16ms window length (Experiment 7),
2. an 8ms window length (Experiment 8), and
3. a 4ms window length (Experiment 9).

This figure shows that the system's performance increased as the window length decreased. Peak performance, 51.3%, occurs when using 2 KLT coefficients with a 4ms window length. This is 24.2% better than the 8ms window, 41.3% at 9 KLT coefficients, and 36.8% better than the 16ms window, 37.5% at 9 KLT coefficients.

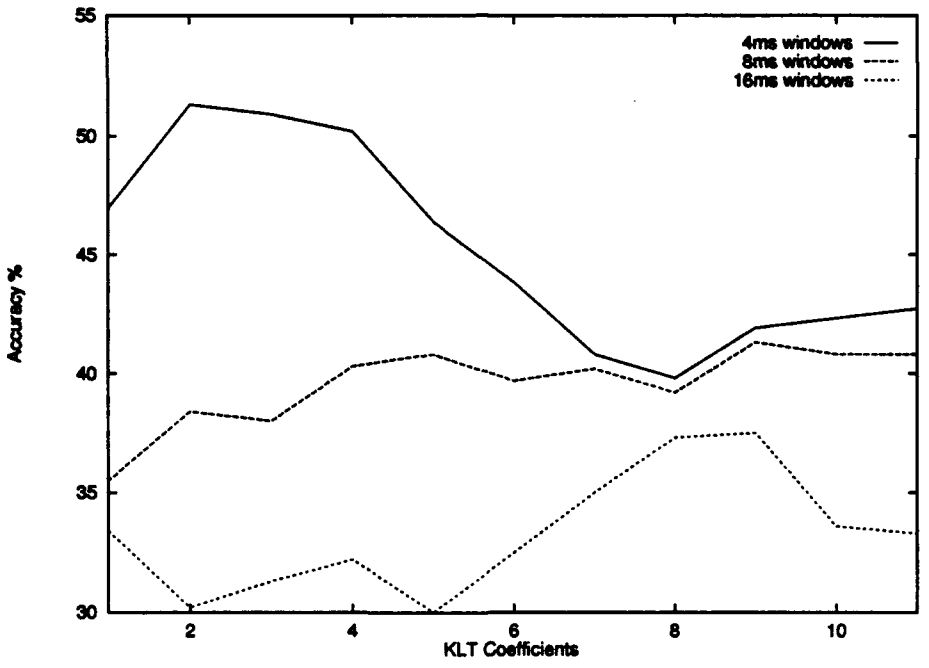

Figure 4.9 Classification accuracy of over-sampled dyadic wavelet decomposition using Db4; without augmentation and with 16ms, 8ms and 4ms windows (Experiments 7, 8 and 9)

Figure 4.10 shows the relative variance retained when classifying the data in Experiments 7, 8 and 9. This figure shows that the reduction of the window length does not necessarily lead to a more compact representation of the data. Based on this result and similar results in the previous experiments, there appears to be no direct link between the compactness of the representation and the classification accuracy, when comparing wavelet methods.

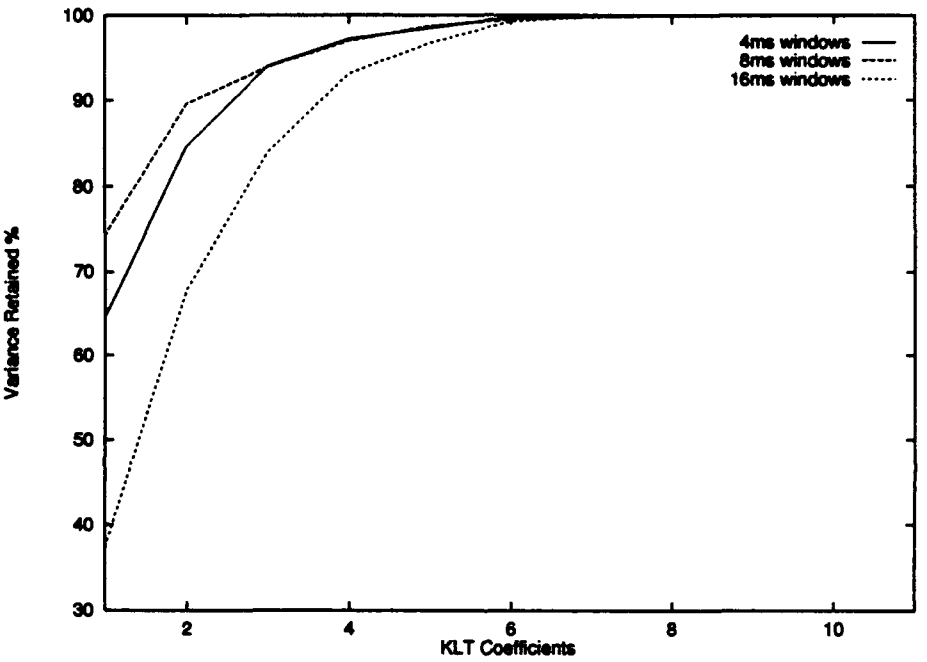

Figure 4.10 Relative variance retained for Experiments 7, 8 and 9

Based on the results presented in this section, the 4ms window is considered the best of the window sizes considered for the wavelet method used.

4.1.5 Augmentation and Reduced Window Size. Figure 4.11 shows the system's classification accuracy using the over-sampled dyadic wavelet decomposition method with zero crossing and energy augmentation, Db4 scaling function, and:

1. 16ms window length (Experiment 3),
2. 4ms window length (Experiment 10), and
3. 4ms window length with the augmentation multiplied by 4 (Experiment 11).

Also shown is the unaugmented classification accuracy using a 4ms window length (Experiment 9). Experiment 11 has its augmentation multiplied by 4, this increases the zero crossing and energy values to the same relative magnitudes that were obtained with the 16ms window.

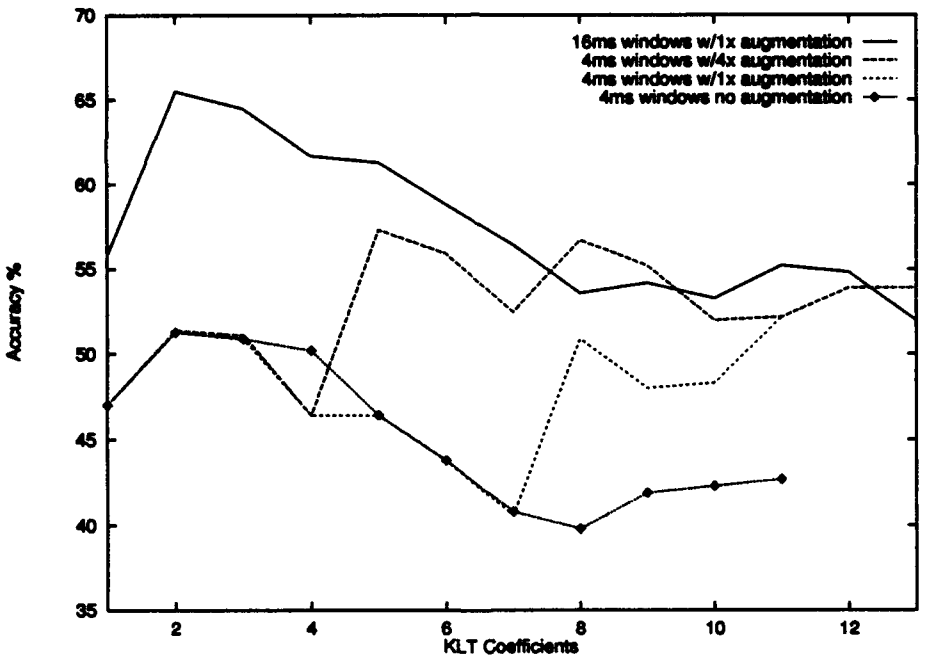

Figure 4.11 Classification accuracy of over-sampled dyadic wavelet decomposition using Db4; with augmentation and with 16ms and 4ms windows (Experiments 3, 10 and 11)

It can be seen in Figure 4.11, that the relationship between the statistics of the augmenting features and the coefficient statistics is crucial to optimizing the classification rate. The relative statistics of the augmenting features for the 16ms window case were better suited to the im-

provement of the classification rate, shown in Figure 4.7. Where as, the relative statistics of the augmenting features for the 4ms window cases were not significant in aiding the classification. This can be seen in Figure 4.11 by the locations of the increases in the classification rate. For the 16ms window length, the augmenting features affected the classification rate from the first KLT coefficient. The 4ms window length case, without artificially increasing the variance, the augmenting features affected the classification from the eighth KLT coefficient. When the variance of the augmenting features was artificially increased, the effect on the classification occurred from the fifth KLT coefficient. This demonstrates the requirement for some kind of feature fusion, where the statistics of different feature sets are made compatible to maximize classification.

Figure 4.12 shows the relative variance retained when classifying the data in Experiments 3, 10 and 11. This figure shows that variations in the statistics of the augmenting features do not significantly affect the compactness of the representation of the data.

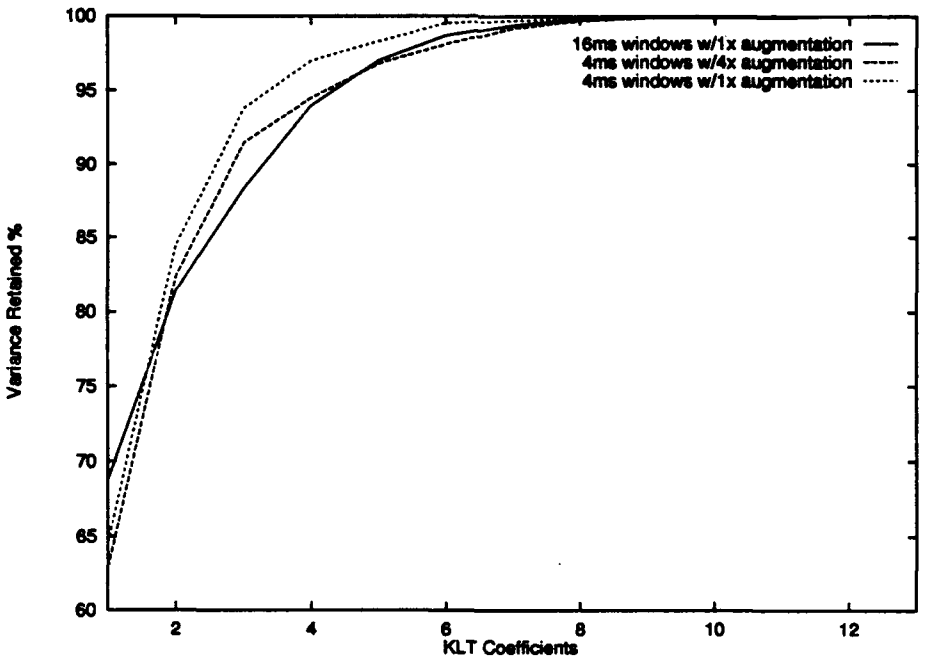

Figure 4.12 Relative variance retained for Experiments 3, 10 and 11

4.2 Fourier Features

4.2.1 Effect of Reduced Window Size. Figure 4.13 shows the system's classification accuracy using 11 single octave frequency bands of Fourier coefficients with no augmenting features, and with window lengths of 16ms, 8ms and 4ms (Experiments 12, 13 and 14 respectively).

This figure shows that the system's performance did not follow any particular trend as the window length decreased. The overall performance for each of these window lengths remained within a 10% tolerance. Peak performance, 84.5%, occurs when using 9 KLT coefficients with a 4ms window length. This is 2.8% better than the 16ms window, 82.2% at 9 KLT coefficients, and 3.4% better than the 8ms window, 81.7% at 9 KLT coefficients.

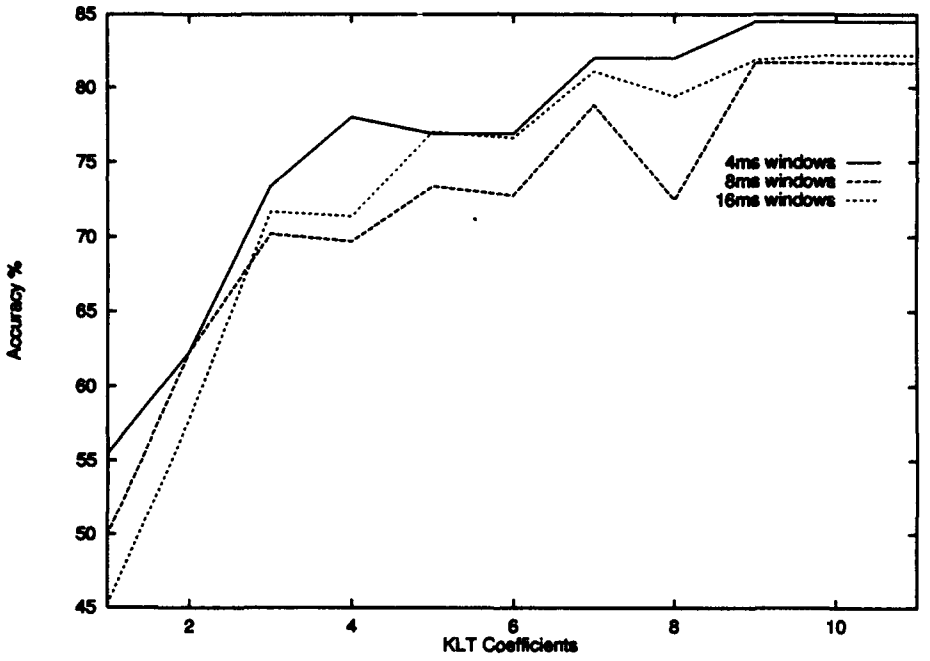

Figure 4.13 Classification accuracy of Fourier features using octave frequency bands; without augmentation and with 16ms, 8ms and 4ms windows (Experiments 12, 13 and 14)

Figure 4.14 shows the relative variance retained when classifying the data in Experiments 12, 13 and 14. This figure shows that the reduction of the window length does not necessarily lead to a more compact representation of the data. The reduction of window length from 16ms to 8ms made almost no difference in the compactness of the representation of the data. However, the 4ms window does produce a more compact representation, approximately 7% at its maximum difference at 2 KLT coefficients.

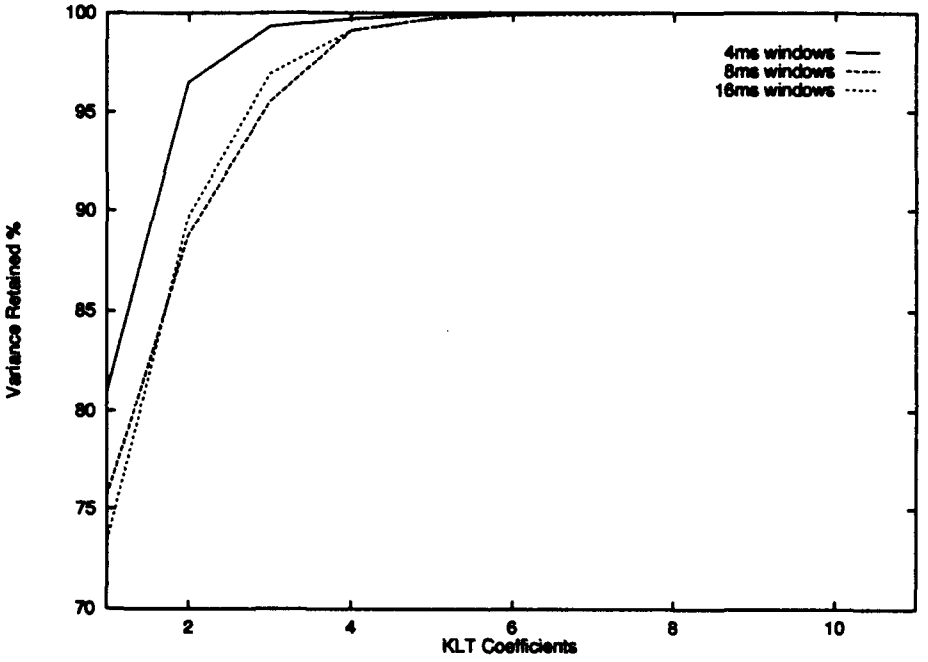

Figure 4.14 Relative variance retained for Experiments 12, 13 and 14

4.2.2 Effect of Augmentation. Figure 4.15 shows the system's classification accuracy using 11 single octave frequency bands of Fourier coefficients with a 4ms window length and:

1. no augmentation (Experiment 14), and
2. zero crossing only augmentation (Experiment 15).

It can be seen in Figure 4.15, that the unaugmented features were not significantly improved by adding the zero crossing data to the feature vectors. Peak performance, of 85.6%, occurred when using 9 KLT coefficients and the zero crossing feature. This was 1.3% better than no augmentation, 84.5% at 9 KLT coefficients.

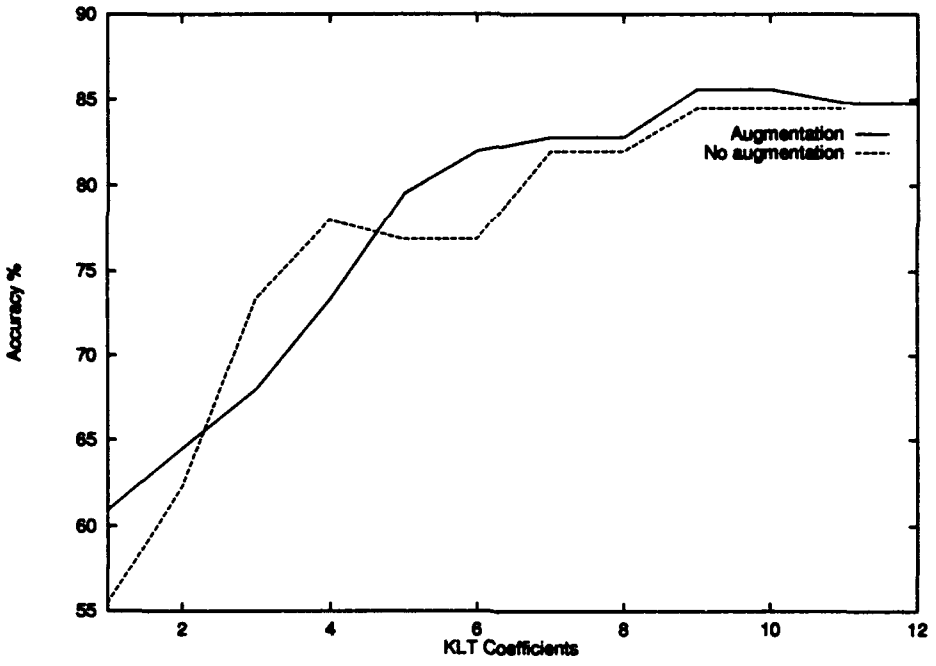

Figure 4.15 Classification accuracy of Fourier features using octave frequency bands; with and without augmentation and with 4ms windows (Experiments 14 and 15)

Figure 4.16 shows the relative variance retained when classifying the data in Experiments 14 and 15. This figure shows that the zero crossing augmentation did lead to a very compact representation of the data, 100% (within 0.000001%) after 2 KLT coefficients. The interesting point to note from this figure and Figure 4.15 is that even though, virtually, all the variance was used after 2 KLT coefficients, the classification performance continued to increase with particularly small increases in the variance added to the data. This is seen in the relative classification rates for 2 KLT coefficients, 64.5%, and 9 KLT coefficients, 85.6%, with their corresponding variance percentage difference of less than 0.000001%. In this case, the very fine detail in the low variance features contained the information that separated the classes.

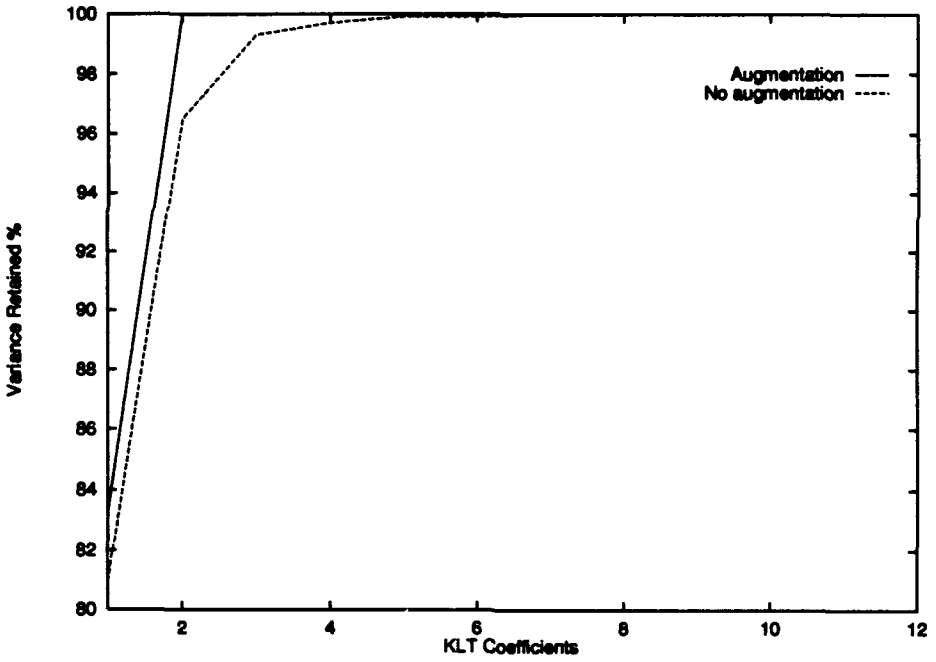

Figure 4.16 Relative variance retained for Experiments 14 and 15

4.2.3 Comparison of Fourier Grouping Methods. Figure 4.17 shows the system's classification accuracy using Fourier coefficients with a 4ms window length grouped in:

1. 11 single octave frequency bands (Experiment 14),
2. 11 equal width frequency bands (Experiment 16), and
3. 17 Mel scale frequency bands with zero crossing augmentation (Experiment 17).

It can be seen in Figure 4.17, that the augmented Mel scale features performed significantly better than the other two grouping methods. Interestingly, the equal width frequency bands method performed slightly better than the octave bands method. Peak performance, of 95.8%, occurred when using 6 KLT coefficients and augmented Mel scale frequency bands. This was 12.3% better than the equal width frequency bands method, 85.3% at 7 KLT coefficients, and 13.4% better than the octave frequency bands method, 84.5% at 9 KLT coefficients.

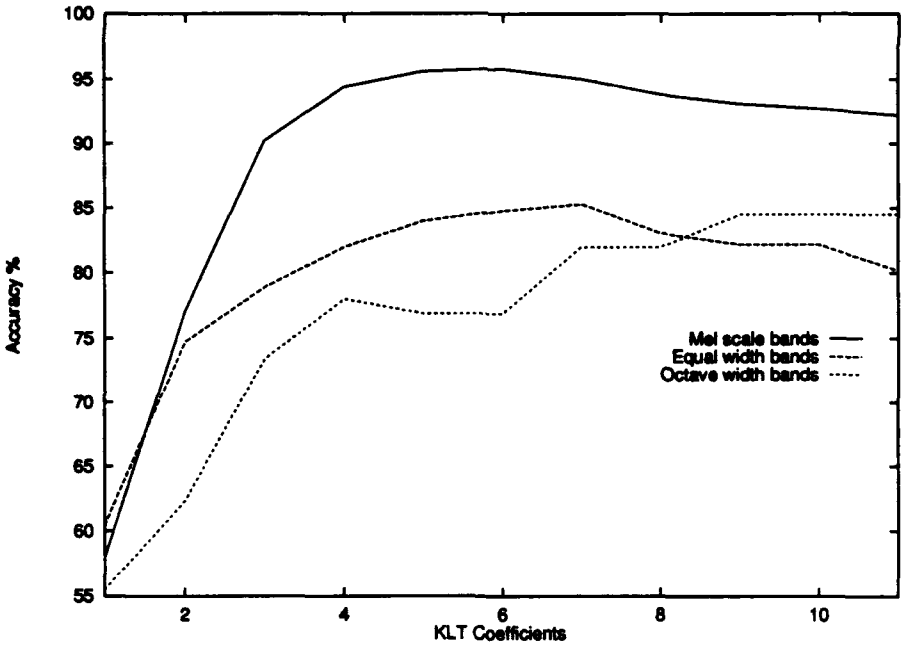

Figure 4.17 Classification accuracy of Fourier features using octave, and equal frequency bands without augmentation, and Mel frequency bands with augmentation; with 4ms windows (Experiments 14, 16 and 17)

Figure 4.18 shows the relative variance retained when classifying the data in Experiments 14, 16 and 17. This figure shows that the Mel scale frequency band method, the best classification method, has the least compact representation of these methods. The octave and equal width frequency band methods have similar compactness, they also have similar classification rates.

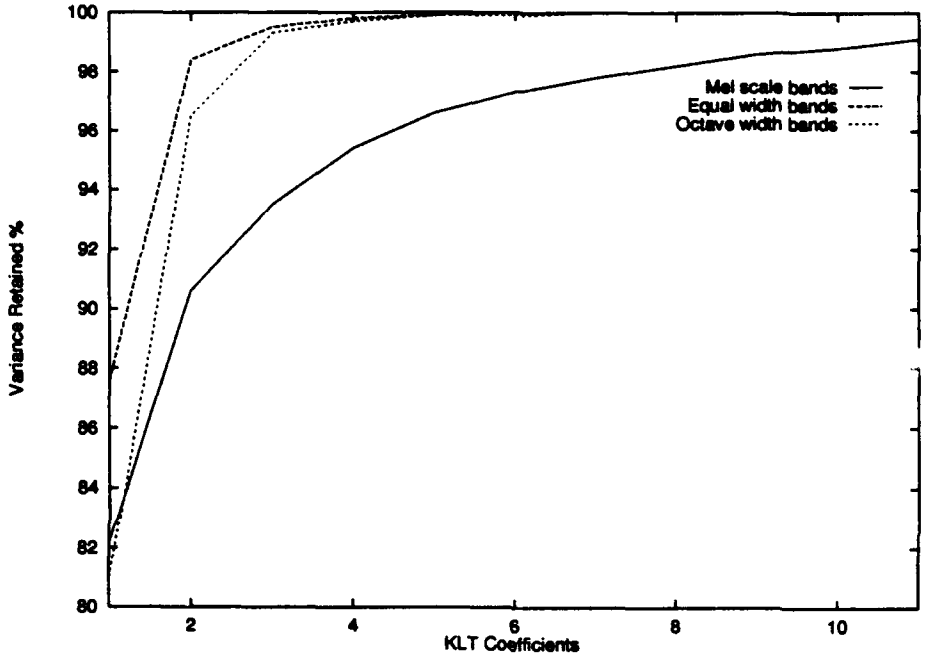

Figure 4.18 Relative variance retained for Experiments 14, 16 and 17

4.3 Comparison between Wavelets, Fourier and Wavelets plus Fourier

Figure 4.19 shows the system's classification accuracy combining the over-sampled dyadic wavelet decomposition method with Db4 scaling function, and 11 single octave frequency bands of Fourier coefficients with a 4ms window length and no augmentation with:

1. equal order of magnitude covariances (Experiment 18) and
2. Fourier coefficients one order of magnitude larger than the wavelet coefficients (Experiment19).

Also show are the classification accuracies for the two methods individually (Experiments 9 and 14).

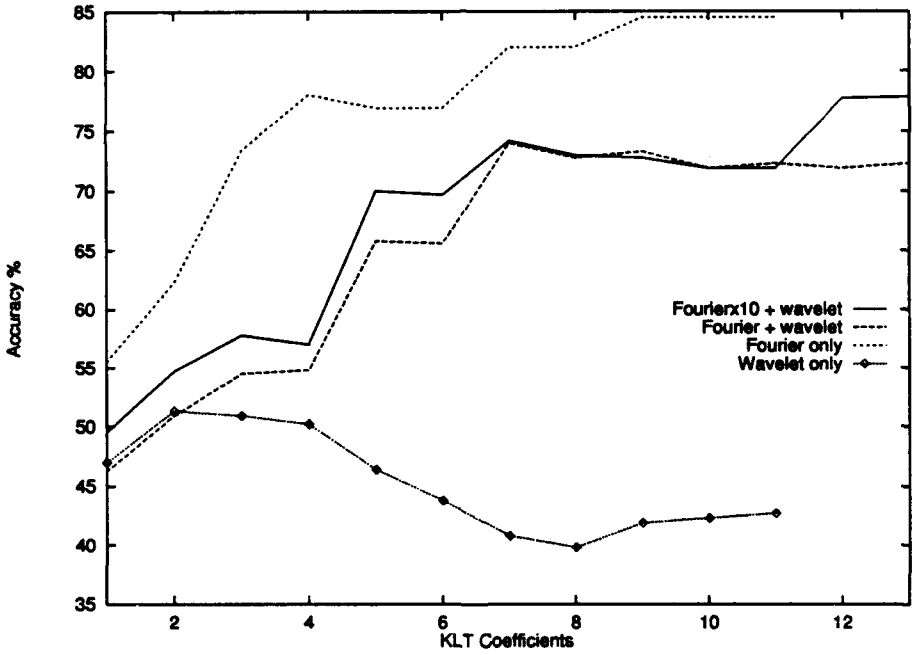

Figure 4.19 Classification accuracy of combined Fourier and Wavelet features using octave Fourier bands and Db4 scaling function; without augmentation and with 4ms windows (Experiments 9, 14, 18 and 19)

It can be seen in Figure 4.19, adding wavelet coefficients to Fourier coefficients reduces the classification rate that was achieved for Fourier only. Improving the weighting of the covariances of

the Fourier coefficients does improve the performance over equal weighting. However, this improvement was not sufficient to reach the "Fourier only" results. The addition of the wavelet coefficients appears to have confused the classifier.

Figure 4.20 shows the relative variance retained when classifying the data in Experiments 9, 14, 18 and 19. This figure shows that both Fourier plus wavelet experiments were almost identical in their compactness, which was also very similar to the "wavelet only" experiment. As with Figure 4.19, the "Fourier only" representation is dominated by the effect of the wavelets.

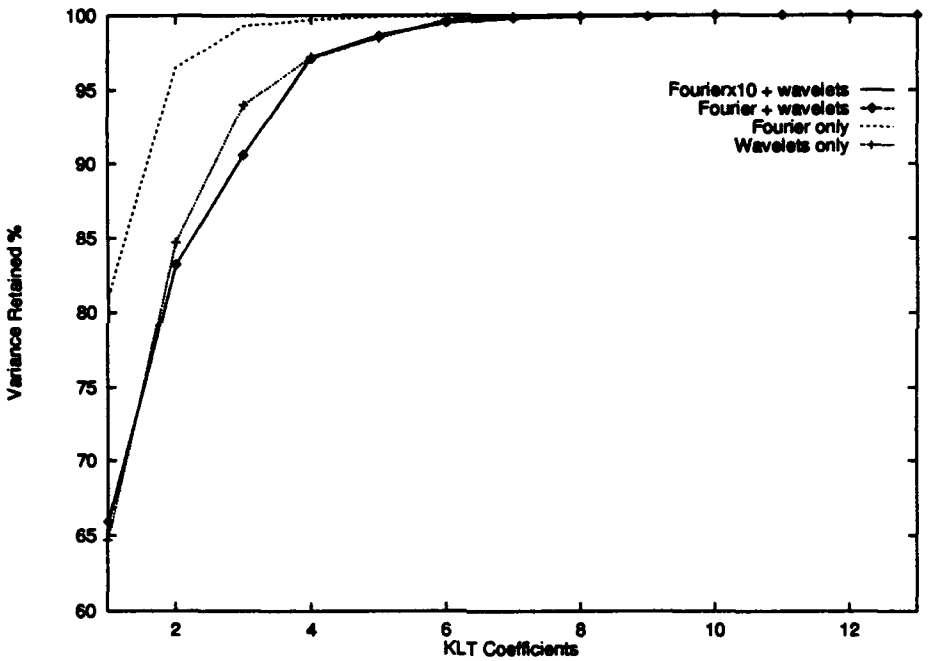

Figure 4.20 Relative variance retained for Experiments 9, 14, 18 and 19

Figure 4.21 shows the system's classification accuracy using:

1. the combined over-sampled dyadic wavelet decomposition method with Db4 scaling function, and 11 single octave frequency bands of Fourier coefficients with equal order of magnitude covariances and with a 4ms window length and no augmentation (Experiment 18),

2. the combined over-sampled dyadic wavelet decomposition method with Db4 scaling function, and 11 single octave frequency bands of Fourier coefficients with the covariance of the Fourier coefficients one order of magnitude larger and with a 4ms window length and no augmentation (Experiment 19) and

3. the best of the individual methods tested, 17 Mel scaled frequency band Fourier coefficients with a 4ms window length and zero crossing augmentation (Experiment 17).

Figure 4.21 shows the Mel scaled Fourier features clearly out perform the wavelet and Fourier combinations.

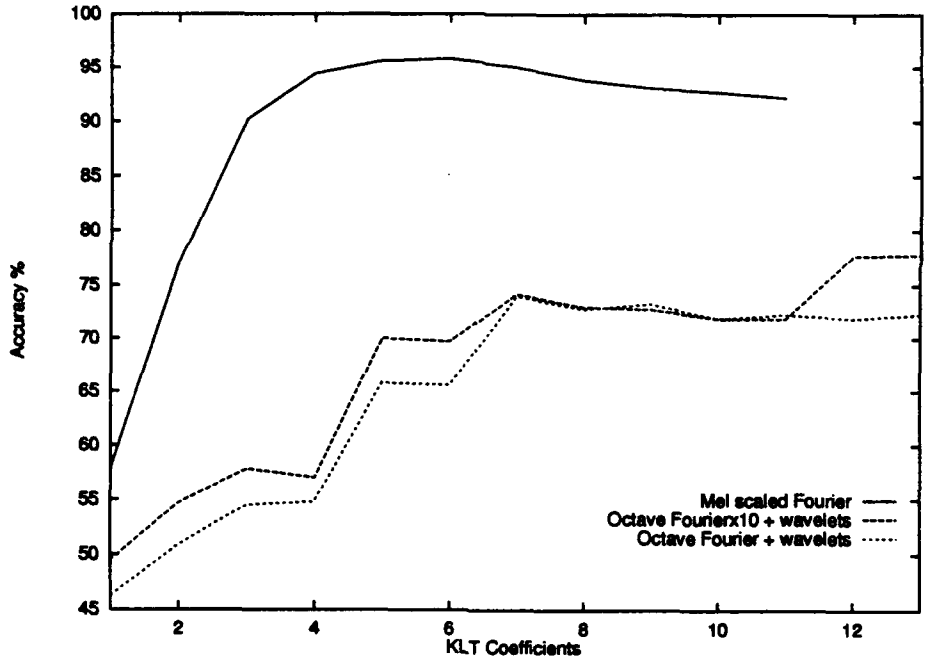

Figure 4.21 Classification accuracy of: Fourier and Wavelet features using octave Fourier bands and Db4 scaling function without augmentation, and Mel Fourier features with augmentation; with 4ms windows (Experiments 17, 18 and 19)

Figure 4.22 shows the relative variance retained when classifying the data in Experiments 17, 18 and 19. This figure does not show any useful relationship between the methods.

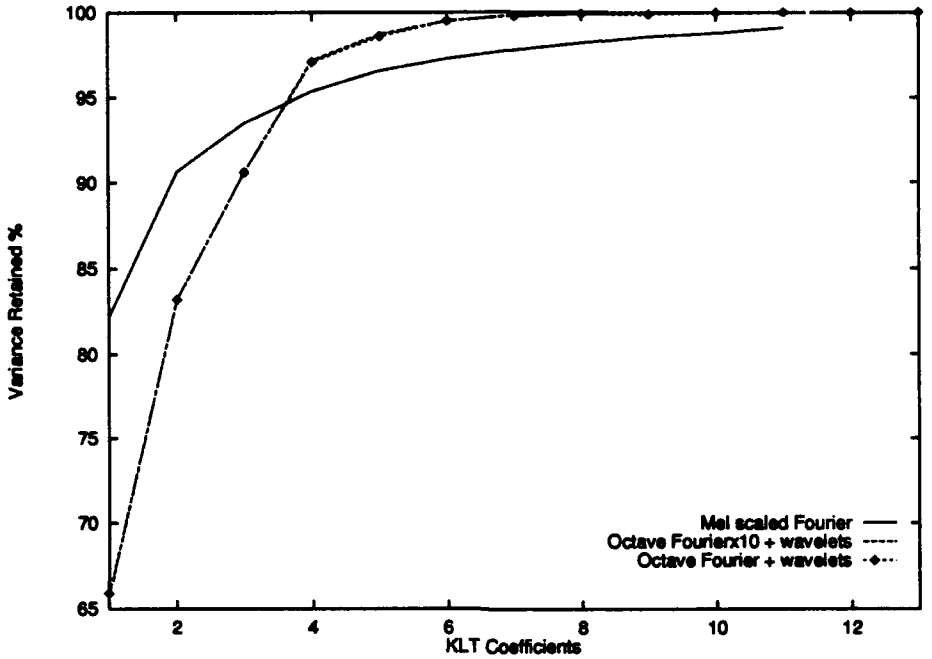

Figure 4.22 Relative variance retained for Experiments 17, 18 and 19

4.4 Summary

The system performance was compared for the standard and over-sampled wavelet decomposition methods. The over-sampled wavelet decomposition method was found to be the more accurate method. This improved accuracy is attributed to the shift-invariance properties of the over-sampled method. The standard method is not shift-invariant.

The system performance for three different wavelet filter types: Daubechies 4 (Db4) and 20 (Db20) coefficient and cubic spline was compared. All three filter types produced the same peak performance. However, the Db4 filter maintained a higher performance than the other two types for all quantities of KLT coefficients. The over-sampled wavelet decomposition using the Db4 scaling function was found to be the more accurate of the wavelet methods tested.

Using the over-sampled Db4 method the effect of varying the window length on the system performance was investigated. As shown in other research (12), the performance improved as the window length reduced. A similar investigation of the effect of reduced window length on a Fourier method, 11 single octave frequency bandwidth features, did not show any trend.

The raw features of the over-sampled Db4 method (4ms and 16ms window lengths) and the 11 octave Fourier (4ms window length) method were augmented with simple window features. The effect on the system's accuracy was different for different window lengths. The 16ms window length over-sampled Db4 method showed significant improvement when the augmenting features were included. Both methods, when using a 4ms window length and augmenting features, showed only minor deviation from the unaugmented accuracy. This highlights the need for a sound method of "feature fusion." Combining feature sets with different statistics can cause one set to totally dominate the other.

The maximum system accuracy when using only wavelet features was 51.3%, compared to the maximum accuracy of 85.3%, when using only Fourier features.

The feature sets of the over-sampled Db4 method and the 11 octave Fourier method were "fused" using simple order of magnitude equalization of their individual covariance matrices. The classification accuracy decreased from the accuracy of the individual 11 octave Fourier method and improved on the accuracy of the individual over-sampled Db4 method. Increasing, by one, the order of magnitude of the Fourier covariances over the wavelet variances did improve the classification accuracy. However, this improvement still does not better the accuracy of the Fourier method on its own.

V. Conclusions and Recommendations

5.1 Conclusions

A system was developed that extracted wavelet and Fourier features from input speech signals from the TI-46 database, transformed the features into Karhunen-Loève space and classified the transformed vectors. This system extracted the speech data from each utterance and decomposed that speech into Fourier, standard or over-sampled dyadic wavelet feature vectors. Each feature vector produced represented a window of the speech.

This research found that the over-sampled wavelet decomposition method produced more accurate results than the standard wavelet decomposition method. This improved accuracy is attributed to the shift-invariance properties of the over-sampled method. The standard method is not shift-invariant.

Of the three different wavelet filter types used in this research: Daubechies 4 (Db4) and 20 (Db20) coefficient and cubic spline, the Db4 scaling function was found to be the more accurate. The Db4 scaling function produced a less compact representation of the data than the other functions. Therefore, less fine detail can produce a more accurate recognizer.

As shown in other research (12), the recognition accuracy, when using wavelet features, improved as the window length is reduced. This effect was not evident for the 11 single octave frequency bandwidth Fourier features.

The effect of augmenting the decomposition features with simple window features was different for different window lengths. The relative statistics of the decomposition features and the augmenting features changed (e.g. if a window was halved in length, the maximum number of zero crossings in that window was halved). Combining feature sets with different statistics can cause one set to totally dominate the other.

This research found, for the method and the filters used, dyadic wavelet features did not perform as well as Fourier features for the recognition of speaker dependent isolated word speech. It must be emphasized that the method of feature extraction for the wavelet and Fourier systems were not the same. The windowing technique was different for each case. For the case of Fourier, the speech was windowed and the Fourier coefficients calculated. This means that only information within the window contributed to the coefficients. For the wavelet case, the wavelet coefficients were calculated for the complete utterance and then the windowing was performed. This means that the entire signal contributed to the coefficients. A re-run of Experiments 9 and 14, using a maximum of six features, with identical windowing techniques is discussed in Appendix G.

Simple "fusion" of the feature sets of the over-sampled Db4 method and the 11 octave Fourier method produced more accurate recognition than wavelet-only method. However, the recognition accuracy was less than the accuracy of the Fourier-only method. The addition of the wavelet feature set to the Fourier feature set introduced confusion. That is, it introduced information that reduced the classification system's ability to separate the classes.

Based on this research, the dyadic wavelet repr[illegible]tation, produced by the method used in this thesis, is not suited to the accurate recognitio[illegible]eaker dependent isolated word speech. The reasons for the wavelet's apparent poor performance could range from the difference in the windowing technique to the unsuitability of wavelets for speech recognition. The method and the filters chosen for this thesis form a very small subset of the possible methods and filters. This prevents any generalized conclusions being made on the suitability of dyadic wavelets for the recognition of speaker dependent isolated word speech.

5.2 Recommendations

This research only scratches the surface of the possibility that there exists a wavelet method that can out-perform all other methods for speech recognition. By its very nature, wavelet analysis

has many variations. There are a countably infinite number of different scaling functions. One or more of these functions may produce the "ideal" representation. There are many possible methods of wavelet decomposition that could be used (e.g. techniques using decomposition within each decomposition level and non-dyadic decomposition). Therefore, it is recommended that other types of wavelet filters and wavelet decomposition methods be investigated.

This thesis used the Karhunen-Loève Transform (KLT) to reduce the dimensions of the feature space. This transformation assumes that the variance of the transformed features is the key to recognition performance. KLT does not guarantee improved class separability. Other transformation methods should be investigated to determine the method that maximizes class separability.

Further research is also recommended in the area of optimizing the window length of a wavelet-based system. This research showed the trend that as the window length decreased from 16ms to 4ms, the accuracy increased.

Appendix A. Scaling Function Coefficients

Table A.1 Daubechies 4 Coefficient Scaling Function

Coefficient	
Number	*Value*
0	0.482962913145
1	0.836516303738
2	0.224143868042
3	-0.129409522551

Table A.2 Daubechies 20 Coefficient Scaling Function

Coefficient	
Number	*Value*
0	0.026670057901
1	0.188176800078
2	0.527201188932
3	0.688459039454
4	0.281172343661
5	-0.249846424327
6	-0.195946274377
7	0.127369340336
8	0.093057364604
9	-0.071394147166
10	-0.029457536822
11	0.033212674059
12	0.003606553567
13	-0.010733175483
14	0.001395351747
15	0.001992405295
16	-0.000685856695
17	-0.000116466855
18	0.000093588670
19	-0.000013264203

Table A.3 31 Coefficient Cubic Spline Scaling Function

Coefficient	
Number	*Value*
0	-0.000927198731456
1	-0.00110373982038
2	0.00188213352389
3	0.00218671237014
4	-0.00388242526559
5	-0.00435383945776
6	0.00820147720599
7	0.00868529481307
8	-0.017982320981
9	-0.017176315492
10	0.0420683514407
11	0.0320808974702
12	-0.110037018388
13	-0.0502017246714
14	0.433922633589
15	0.76613005376
16	0.433922633589
17	-0.0502017246714
18	-0.110037018388
19	0.0320808974702
20	0.0420683514407
21	-0.017176315492
22	-0.017982320981
23	0.00868529481307
24	0.00820147720599
25	-0.00435383945776
26	-0.00388242526559
27	0.00218671237014
28	0.00188213352389
29	-0.00110373982038
30	-0.000927198731456

Appendix B. Tabulated Results

Table B.1 Results of Experiment 1

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	29.4	57.1
2	39.7	72.2
3	39.5	86.6
4	34.8	91.8
5	34.5	94.4
6	37.3	96.6
7	48.1	98.6
8	43.3	99.2
9	46.9	99.5
10	46.9	99.8
11	44.1	99.9
12	43.9	99.9
13	42.3	100.0

Table B.2 Results of Experiment 2

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	33.1	68.7
2	38.9	87.0
3	38.4	92.6
4	43.9	95.9
5	58.6	98.3
6	62.3	98.7
7	60.6	99.1
8	58.8	99.4
9	57.3	99.7
10	55.8	99.9
11	54.8	99.9
12	55.3	99.9
13	53.6	100.0

Table B.3 Results of Experiment 3

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	55.8	68.8
2	65.5	81.4
3	64.5	88.4
4	61.7	94.0
5	61.3	97.0
6	58.8	98.7
7	56.4	99.4
8	53.6	99.8
9	54.2	99.9
10	53.3	99.9
11	55.2	99.9
12	54.8	100.0
13	52.0	100.0

Table B.4 Results of Experiment 4

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	55.8	79.2
2	65.5	93.7
3	60.6	97.0
4	56.1	98.4
5	53.3	98.9
6	50.2	99.3
7	45.8	99.7
8	43.3	99.8
9	43.8	99.9
10	43.9	99.9
11	49.2	99.9
12	48.0	100.0
13	49.2	100.0

Table B.5 Results of Experiment 5

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	55.8	77.5
2	65.3	91.7
3	62.7	96.5
4	59.4	97.6
5	58.4	98.6
6	55.0	99.2
7	55.2	99.7
8	52.0	99.8
9	51.1	99.9
10	52.0	99.9
11	54.4	99.9
12	54.5	100.0
13	53.8	100.0

Table B.6 Results of Experiment 6

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	58.3	77.2
2	57.5	85.7
3	56.4	92.7
4	55.8	96.3
5	53.9	98.4
6	52.2	99.3
7	51.6	99.8
8	51.6	99.9
9	51.1	99.9
10	52.8	99.9
11	52.0	100.0
12	49.5	100.0

Table B.7 Results of Experiment 7

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	33.4	37.4
2	30.2	67.9
3	31.3	84.0
4	32.2	93.1
5	30.0	96.7
6	32.5	99.3
7	35.0	99.8
8	37.3	99.9
9	37.5	99.9
10	33.6	99.9
11	33.3	100.0

Table B.8 Results of Experiment 8

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	35.2	74.3
2	38.4	89.6
3	38.0	93.9
4	40.3	97.0
5	40.8	98.7
6	39.7	99.5
7	40.2	99.8
8	39.2	99.9
9	41.3	99.9
10	40.8	100.0
11	40.8	100.0

Table B.9 Results of Experiment 9

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	47.0	64.7
2	51.3	84.7
3	50.9	94.0
4	50.2	97.2
5	46.4	98.5
6	43.8	99.7
7	40.8	99.9
8	39.8	99.9
9	41.9	99.9
10	42.3	100.0
11	42.7	100.0

Table B.10 Results of Experiment 10

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	47.0	64.6
2	51.3	84.5
3	50.9	93.8
4	46.4	97.0
5	46.4	98.3
6	43.8	99.5
7	40.6	99.7
8	50.9	99.8
9	48.0	99.9
10	48.3	99.9
11	52.2	100.0

Table B.11 Results of Experiment 11

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	47.0	63.0
2	51.4	82.4
3	51.1	91.5
4	46.4	94.6
5	57.3	96.8
6	55.9	98.1
7	52.5	99.2
8	56.7	99.7
9	55.2	99.9
10	52.0	99.9
11	52.2	99.9
12	53.9	100.0
13	53.9	100.0

Table B.12 Results of Experiment 12

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	45.2	73.4
2	57.8	89.7
3	71.7	96.9
4	71.4	99.1
5	77.0	99.7
6	76.6	99.9
7	81.1	99.9
8	79.4	99.9
9	81.9	99.9
10	82.2	100.0
11	82.2	100.0

Table B.13 Results of Experiment 13

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	50.0	75.7
2	62.3	88.8
3	70.2	95.5
4	69.7	99.1
5	73.4	99.7
6	72.8	99.9
7	78.8	100.0
8	72.5	100.0
9	81.7	100.0
10	81.7	100.0
11	81.7	100.0

Table B.14 Results of Experiment 14

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	55.5	81.0
2	62.3	96.5
3	73.4	99.3
4	78.0	99.7
5	76.9	99.9
6	76.9	99.9
7	82.0	100.0
8	82.0	100.0
9	84.5	100.0
10	84.5	100.0
11	84.5	100.0

Table B.15 Results of Experiment 15

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	60.9	83.3
2	64.5	100.0
3	68.0	100.0
4	73.3	100.0
5	79.5	100.0
6	82.0	100.0
7	82.8	100.0
8	82.8	100.0
9	85.6	100.0
10	85.6	100.0
11	84.8	100.0
12	84.8	100.0

Table B.16 Results of Experiment 16

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	60.5	87.6
2	74.7	98.4
3	78.9	99.5
4	82.0	99.8
5	84.0	99.9
6	84.8	99.9
7	85.3	99.9
8	83.1	100.0
9	82.2	100.0
10	82.2	100.0
11	80.2	100.0

Table B.17 Results of Experiment 17

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	58.0	82.2
2	77.0	90.6
3	90.2	93.5
4	94.4	95.4
5	95.6	96.6
6	95.8	97.3
7	95.0	97.8
8	93.8	98.2
9	93.1	98.6
10	92.7	98.8
11	92.2	99.1

Table B.18 Results of Experiment 18

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	46.3	65.9
2	50.9	83.2
3	54.3	90.6
4	54.8	97.1
5	65.8	98.6
6	65.6	99.5
7	74.0	99.8
8	72.8	99.9
9	73.3	99.9
10	71.9	100.0
11	72.3	100.0
12	71.9	100.0
13	72.3	100.0

Table B.19 Results of Experiment 19

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	49.5	65.8
2	54.9	83.2
3	57.8	90.6
4	57.0	97.2
5	70.0	98.7
6	69.7	99.5
7	74.2	99.8
8	73.0	99.0
9	72.8	100.0
10	71.9	100.0
11	71.9	100.0
12	77.7	100.0
13	77.8	100.0

Appendix C. Feature Generation Program

```
/*  ************************************************************
Program Name : preproc.c

Thesis Task :  Isolated Word, Speaker Dependent, Speech Recognition
               Using Wavelet Analysis

Program Task : Preprocess the speech data in preparation for the
               KLT then the Classification routines

Author :  FLTLT Steve Ainge
          (with assistance from FLTLT D. Neale Prescott for
   much of the framework code)

Last Update :       14 Nov 93

              *         *         *         *          *

   INPUT - An ASCII file containing the filenames of NIST format
         speech files

   OUTPUT - (1) A binary file containing all the feature vectors
              for all of the files

            (2) An ASCII file which contains the cumulative
         feature vector count at the end of each word


 ************************************************************** */

#define NRANSI
#include <nr.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <nrutil.h>

#define MAX_SAMPLES 30000
#define MAX(a,b)    (((a) > (b)) ? (a) : (b))
#define f_len "4"

/* Define complex structure for Fourier analysis */

typedef struct {
    float real, imag;
} COMPLEX;

/*            FUNCTION FORWARD DECLARATIONS                    */

float calculate_window_energy(float *ptr, int samples_per_window);
```

```
int calc_win_zero_cross(float *ptr, int samples_per_window); float
convert_12_bit_to_float(unsigned short int temp_in);
int get_NIST_header_value(char NIST_header[1024],char
find_string[]);
void fft(COMPLEX *x, int m);
void ham(COMPLEX *x, int n);

/* ------------------------------------------------------------ */

main(int argc, char *argv[])
{

FILE *list_file, *data_file, *index_file, *feature_file, *infp;

COMPLEX *samp;

char      analysis_type[50], data_file_name[100], spkr_dir[5],
          temp_name[50], NIST_header[1024], filt_name[50];

int  counta, status, filecount = 0, file_min = 0, file_max = 0,
     len, i = 0, j = 0, k = 0, m = 0, sample_rate, num_stat_wins,
     win_zc, fixed_zc_threshold = 50, fft_length = 4096,
     old_bin_number, num_freq, new_bin_number, samples_per_window,
     utt_length, lo_level, hi_level, num_levels, count,
     filt_len = 4, off_set, coeff, win_start, num_ftrs, rec_len,
     max_windows, max_stat_windows, utterance_start,
     utterance_stop, tracker, M1, low_flag = 0, high_flag = 0,
     digit_count;

unsigned short int  temp_in;

unsigned long int   total_FV = 0, index_array[1500];

float     signal[MAX_SAMPLES], norm_mean = 0, window_size = 0.010,
          stat_win_size = 0.10, zc_mean, silence_mean, win_energy,
          peak_win_energy, *signal_ptr, I1, I2,
          upper_energy_threshold, lower_energy_threshold, tempflt,
          upper_freq, mel_energy[25], poww, feature_vector[40],
          work, *wksp, **approx, **detail, *h, *g, ham_factor,
          ham_val;

double  zc_std, silence_std, zc_threshold;

/*.....................................................*/

if(argc==3)
     {
     list_file=fopen(argv[1],"rb");
     if(!list_file)
          {
          printf("\n\n\n\nError opening the list file\n\n");
```

```
            exit(1);
            }
        }
else
        {
        printf( "\n\n\nPlease enter the input file name and the type
                of analysis\n\n");
        printf("Syntax : preproc <input file name> <analysis type
                f/w/o>\n\n\n");
        exit(1);
        }

analysis_type[0] = *argv[2];
analysis_type[1] = '\0';
if(strcmp(analysis_type,"f")==0)
        {
        strcpy(analysis_type,"FOURIER");
        }
else if(strcmp(analysis_type,"w")==0)
        {
        strcpy(analysis_type,"WAVELET");
        }
else if(strcmp(analysis_type,"o")==0)
        {
        strcpy(analysis_type,"OVERSAMPLED WAVELET");
        }
else
        {
        printf("\n\n\nAnalysis type not recognized.\n\n\n");
        exit(1);
        }

        lo_level = 0;
        hi_level = 10;
        num_levels = hi_level - lo_level + 1;

feature_file=fopen("results/the.dat","wb");
if(!feature_file)
        {
        printf("Cant open the feature file\n");
        exit(1);
        }
```

```
/*.....................................................*/
/*          ALLOCATE STORAGE FOR THE DFT               */
samp = (COMPLEX *) calloc(fft_length, sizeof(COMPLEX));
if(!samp)
     {
     printf("\n  Could not allocate sample memory.\n");
     exit(1);
     }

/*.....................................................*/

while(!feof(list_file))
     {
     status = fscanf(list_file,"%s",&temp_name);
     if(status != 1) break;
     strcpy(data_file_name, "");
     len = 100 - strlen(data_file_name);
     strncat(data_file_name, temp_name, len);

/*  *********************************************************

       Open the data file and read out the NIST header

   ********************************************************* */

     data_file = fopen(data_file_name,"rb");
     if(!data_file)
          {
          printf("\nError opening %s in open_read\n",
                  data_file_name);
          return(NULL);
          }

     status = fread( NIST_header, 1, 1024, data_file);
     if(status != 1024)
          {
          printf("\nError reading header of file %s\n",
                  data_file_name);
          exit(1);
          }

/* *********************************************************

     Read
       the number of data points in the file,
       the maximum sample amplitude
       the minimum sample amplitude

      from the header info
```

```
 ********************************************************* */
rec_len = get_NIST_header_value(NIST_header,
                "sample_count -i ");
     if(rec_len > MAX_SAMPLES)
          {
          printf("File too long : %s [%d] : truncated to %d
                samples\n", data_file_name, rec_len, MAX_SAMPLES);
         rec_len = MAX_SAMPLES;
          }
     sample_rate = get_NIST_header_value(NIST_header,
                   "sample_rate -i ");
     file_min = get_NIST_header_value(NIST_header,
                 "sample_min -i ");
     file_max = get_NIST_header_value(NIST_header,
                 "sample_max -i ");
     file_max = MAX(abs(file_max),abs(file_min));
     window_size   = 0.010;     /* 10 millisecond window */

/* *********************************************************

    Convert from 12 bit twos complement to 16 bit signed
    integers

 ********************************************************* */
for(i = 0;i < MAX_SAMPLES;i++) signal[i] = 0.0;
     for(i = 0 ; i < rec_len ; i++)
          {
          fread( &temp_in, 2, 1, data_file);
          signal[i] = convert_12_bit_to_float(temp_in)/file_max;
        }

/* *********************************************************

    Tidy up the files that are open etc

 ********************************************************* */

     fclose(data_file);

     filecount++;

     samples_per_window = (int) ((float)sample_rate * window_size);
     max_stat_windows = (int) (rec_len/samples_per_window);
     num_stat_wins = (int) (stat_win_size/window_size);
     peak_win_energy = 0;

/* *********************************************************

    Calculate the statistics of the energy and the
    zero crossing  for the SILENCE.
    This is based on the article by L.R.Rabiner and M.R. Sambur
```

```
    "An Algorithm for Determining the Endpoints of Isolated
     Utterances"
    The Bell System Technical Journal, Vol.54, Feb.1975, p.297-315
    Interpretation by : D.N.Prescott May 93

   ********************************************************** */

     zc_mean      = 0;
     zc_std       = 0;
     silence_mean = 0;
     silence_std  = 0;
     peak_win_energy = 0;

     for(i=0; i <= num_stat_wins; i++)
          {
          signal_ptr = &signal[i * samples_per_window];
          win_zc = calc_win_zero_cross(signal_ptr,
                    samples_per_window);
          signal_ptr = &signal[i * samples_per_window];
          win_energy = calculate_window_energy( signal_ptr,
                    samples_per_window);
          if(win_energy > peak_win_energy)
               peak_win_energy = win_energy;
          zc_mean += win_zc;
          zc_std  += win_zc * win_zc;
          silence_mean += win_energy;
          silence_std  += win_energy * win_energy;
          }
   /*end of for i = 1 to num_stat_wins */

/* ..................................................... */

     zc_mean = zc_mean/num_stat_wins;
     zc_std = (zc_std/num_stat_wins - zc_mean*zc_mean);
     zc_std = sqrt(fabs(zc_std));
     silence_mean = silence_mean/num_stat_wins;
     silence_std = (silence_std/num_stat_wins -
                    silence_mean*silence_mean);
     silence_std = sqrt(fabs(silence_std));
     if(fixed_zc_threshold < (int) (zc_mean + 2*zc_std) )
          {
          zc_threshold = fixed_zc_threshold;
          }
     else
          {
          zc_threshold = zc_mean + 2*zc_std;
          }

/* ************************************************************
  Calculate the peak energy in a window for the
      whole utterance.
```

```
*****************************************************************/

     for(i= (num_stat_wins+1); i < max_stat_windows; i++)
          {
          signal_ptr = &signal[i * samples_per_window];
          win_energy = calculate_window_energy( signal_ptr,
                  samples_per_window);
          if(win_energy > peak_win_energy)
               peak_win_energy = win_energy;
          if(peak_win_energy >= (float) samples_per_window*
                             abs(file_max))
               printf("Something wrong , too much energy %f :
               %s\n",peak_win_energy,data_file_name);
          }

/* ****************************************************************
     Set the energy thresholds for the start and stop
      finding routine using energy as the first estimate
   **************************************************************** */

     I1 = 0.03 * (peak_win_energy - silence_mean) + silence_mean;
     I2 = 4* silence_mean;
     if(I1 > I2)
          {
          lower_energy_threshold = I2;
          }
     else
          {
          lower_energy_threshold = I1;
          }
     upper_energy_threshold = 5 * lower_energy_threshold;

/* ****************************************************************
      Find the START of the utterance using energy
   **************************************************************** */

     low_flag = 0;
     high_flag = 0;
     i = num_stat_wins +1;
     signal_ptr = &signal[i*samples_per_window];
     while(!(low_flag && high_flag) )
          {
          while(!low_flag)
               {
               win_energy = calculate_window_energy( signal_ptr,
                           samples_per_window);
               if(win_energy >= lower_energy_threshold)
                    {
                    low_flag = 1;
                    }
               else
```

```
            {
            i++;
            if( i >= max_stat_windows )
                 {
                 i = num_stat_wins +1;
                 low_flag = 1;
                 high_flag = 1;
                 printf("Could not find a start. Start is
                     set at 100ms :%s\n",data_file_name);
                }
            else
                 {
                 signal_ptr += samples_per_window;
                 }
            }

       }

counta = i;

while(!high_flag & low_flag)
       {
       win_energy = calculate_window_energy( signal_ptr,
                    samples_per_window);
       if(win_energy >= upper_energy_threshold)
            {
            high_flag = 1;
            }
       else
            {
            counta++;
            if( counta >= max_stat_windows )
                 {
                 counta = num_stat_wins +1;
                 low_flag = 1;
                 high_flag = 1;
                 printf("Could not find a start. Start is
                     set at 100ms :%s\n",data_file_name);
                }
            else
                 {
                 signal_ptr += samples_per_window;
                 }
            }

       if(win_energy < lower_energy_threshold)
            low_flag = 0;

       }

if(counta == i)
```

```
                utterance_start = i -1;
            else
                utterance_start = counta;

            }

/* ****************************************************************

       Find the STOP of the utterance using energy

   **************************************************************** */

      low_flag = 0;
      high_flag = 0;
      i = max_stat_windows;
      signal_ptr = &signal[i*samples_per_window];
      while(!(low_flag && high_flag) )
           {
           while(!low_flag)
                {
                win_energy = calculate_window_energy( signal_ptr,
                             samples_per_window);
                if(win_energy >= lower_energy_threshold)
                     {
                     low_flag = 1;
                     }
                else
                     {
                     i--;
                     if( i <= num_stat_wins )
                          {
                          i = max_stat_windows;
                          low_flag = 1;
                          high_flag = 1;
                          printf("Could not find a start. Stop is
                             set at EOF :%s\n",data_file_name);
                      }
                     else
                          {
                          signal_ptr -= samples_per_window;
                          }
                     }

                }

           counta = i;

           while(!high_flag & low_flag)
                {
                win_energy = calculate_window_energy( signal_ptr,
                             samples_per_window);
```

```
                if(win_energy >= upper_energy_threshold)
                     {
                     high_flag = 1;
                     }
                else
                     {
                     counta--;
                     if( counta <= num_stat_wins )
                          {
                          counta = max_stat_windows;
                          low_flag = 1;
                          high_flag = 1;
                          printf("Could not find a start. Stop is
                             set at EOF :%s\n",data_file_name);
                      }
                     else
                          {
                          signal_ptr -= samples_per_window;
                          }
                     }

                if(win_energy < lower_energy_threshold)
                     low_flag = 0;

                }

           if(counta == i)
                {
                utterance_stop = i +1;
                }
           else
                {
                utterance_stop = counta;
                }

           }

     if( (utterance_stop - utterance_start) < 1 )
          {
          utterance_start = num_stat_wins +1;
          utterance_stop = max_stat_windows;
          printf("The stop was earlier than the start !!!
                   %s\n",data_file_name);
          }

/* ************************************************************
      Find the START of the utterance using Zero crossings
************************************************************ */

     signal_ptr = &signal[(utterance_start -1) *
                    samples_per_window];
```

```
     M1 = 0;
     if( utterance_start > (25+num_stat_wins) )
          {
          i = 25;
          }
     else
          {
          i = utterance_start - num_stat_wins;
          }

     for(counta = 0; counta < i; counta++)
          {
          win_zc = calc_win_zero_cross(signal_ptr,
                    samples_per_window);
          if(win_zc > (int) zc_threshold)
               {
               M1++;
               tracker = utterance_start -1 -counta;
               }

          signal_ptr -= samples_per_window;

          }

     if(M1 >= 3) utterance_start = tracker;

/* ********************************************************************
      Find the STOP of the utterance using Zero crossings
   ******************************************************************** */

     signal_ptr = &signal[(utterance_stop +1) *
                    samples_per_window];
     M1 = 0;
     if( utterance_stop < (max_stat_windows - 25) )
          {
          i = 25;
          }
     else
          {
          i = max_stat_windows - utterance_stop;
          }

     for(counta = 0; counta < i; counta++)
          {
          win_zc = calc_win_zero_cross(signal_ptr,
                    samples_per_window);
          if(win_zc > (int) zc_threshold)
               {
               M1++;
               tracker = utterance_stop +1 +counta;
               }
```

```
            signal_ptr += samples_per_window;

            }

        if(M1 >= 3) utterance_stop = tracker;

        if( (utterance_stop - utterance_start) < 1 )
            {
            utterance_start = num_stat_wins +1;
            utterance_stop = max_stat_windows;
            printf("The stop was earlier than the start !!!
                    %s\n",data_file_name);
            }

/* ************************************************************** */

        max_windows = samples_per_window * (utterance_stop -
                    utterance_start);
        utt_length = max_windows;
        utterance_start *= samples_per_window;
        window_size   = 0.004;      /* 4 millisecond window */
        samples_per_window = (int) ((float)sample_rate * window_size);
        utterance_start = (int) (utterance_start/samples_per_window);
        max_windows = (int) (max_windows/samples_per_window);
        max_windows = (max_windows*2)-1;
        signal_ptr = &signal[utterance_start*samples_per_window];

        if(strcmp(analysis_type,"FOURIER")==0)
            {

/* ********************************************************************
      Calculate the DFT
       Convert to Mel scale
       Count zero crossings per window
       Form feature vector

   ****************************************************************** */

            for(counta = 0; counta < max_windows; counta++)
                {
                for(i = 0; i< samples_per_window; i++)
                    {
                    samp[i].real = signal_ptr[i];
                    samp[i].imag = 0;
                    }
```

```
                for (i = samples_per_window ; i < fft_length ; i++)
                    {
                     samp[i].real = 0;
                     samp[i].imag = 0;
                     }

                win_zc = calc_win_zero_cross(signal_ptr,
                          samples_per_window);

                signal_ptr += samples_per_window/2;

/* ..................................................
          calculate the DFT magnitude
   .................................................. */

                ham(&samp[0], samples_per_window);

                fft(samp, 12);

                norm_mean = 4.0/(double)(fft_length * fft_length);
                for (i = 0 ; i < fft_length ; i++)
                     {
                     tempflt  = samp[i].real * samp[i].real;
                     tempflt += samp[i].imag * samp[i].imag;
                     tempflt *= norm_mean;
                     samp[i].real = tempflt;
                     }

/* ......   Mel bin conversion from DFT coefficients .............

      1. Find the maximum frequency represented by the DFT
      2. Convert the maximum DFT frequency to Mel using
           Max_mel = (1000/log(2)) * log( 1 + max_freq/1000 )
      3. Now divide the Max_mel into the required number of bins
      4. In this case 150.4 is the number for 6250Hz divided by 19
         mel bins
      Refer to Speech Processing by T. Parsons p.73
.................................................... */  /*
           old_bin_number = 0;
                for(i=1; i <= 19; i++)
                     {
                     poww = (i*150.42/3321.93);
                     poww = (float)  pow(10.0, poww);
                     upper_freq = (float) 1000*(poww  -1);
                     new_bin_number = floor(upper_freq*128/6250);
                     mel_energy[i] = 0;
                     for(j=old_bin_number; j < new_bin_number; j++)
                         mel_energy[i] += samp[j].real;
                     old_bin_number = new_bin_number;
                     }
```

```
*/
                for(i=0; i < 40; i++) feature_vector[i] = 0.0;

                num_freq = (int)(fft_length/2);
                for(i=0;i<11;i++)
                     {
                     num_freq = (int)(num_freq/2);
                     if(num_freq<=1) num_freq = 1;
                     for(j=num_freq;j<num_freq*2;j++)
                          feature_vector[i] += samp[j].real;
                     feature_vector[i] = feature_vector[i]/
                                         (float)num_freq;
                     }

/* Add the first three mel bins together as per the article by Pols
1971. This is done to concentrate all the first formant energy in
one feature */
/*
                for(i=1;i< 4; i++)
                     feature_vector[0] += mel_energy[i];
                feature_vector[0] =
                     10*log10(MAX(feature_vector[0],1.e-14));

                for(i=4;i<=19; i++)
                     feature_vector[i-3] =
                          10*log10(MAX(mel_energy[i],1.e-14));

                feature_vector[17] = win_zc/1.0;
*/
/* ****************************************************************

       Print out the feature vectors

       [0..16] contains the log magnitude of the Mel bin

       [17] contains the number of zero crossings that occur per
       window

   **************************************************************** */

                for(j=0;j<11;j++)
                     fwrite(&feature_vector[j],sizeof(float),
                         1,feature_file);

                num_ftrs = 11;
/* end of : for(counta = 0; counta < max_windows; counta++) */
           }

          }
     else
          {
```

```
/*   Calculate the ODWT coefficients and determine the average
 coefficients for each window.
*/
          for(i = 1;i <= 32;i++)
               {
               if(utt_length <= (int)pow(2.0,(double)i) )
                    {
                    count = (int)pow(2.0,(double)(i+1));
                    break;
                    }
               }

/*   Define data workspace vector. */
          wksp = vector(0,count - 1);

/*   Copy data vector into work space. */
          for (i = 0;i < utt_length;i++)
               {
               wksp[i] = signal[i+(utterance_start*
                              samples_per_window)];
               }

/*   Pad vector with zeroes. */
          for(i = utt_length;i < count;i++)
               {
               wksp[i] = 0.0;
               }

/*   Define data vectors. */
          approx = matrix(0,hi_level,0,count - 1);
          detail = matrix(0,hi_level,0,count - 1);
          h = vector(0,filt_len - 1);
          g = vector(0,filt_len - 1);

/*   Get H filter file name. */
          strcpy(filt_name,"filters/db");
          strcat(filt_name,f_len);
          strcat(filt_name,".dat");

/*   Read H filter file into vector. */
          if((infp = fopen(filt_name,"r")) == NULL)
               {
               printf("Cannot open file\n");
               exit(1);
               }

          for(i = 0;i < filt_len;i++)
               {
               fscanf(infp,"%f",&h[i]);
               }
```

```
          fclose(infp);

/*   Generate G filter. */
          for(i = 0;i < filt_len;i++)
               {
               g[i] = (float)pow(-1.0,(double)(filt_len-1-i)) *
                      h[filt_len-1-i];
               }

          if(strcmp(analysis_type,"WAVELET")==0)
          {
/* ************************************************************ */
/*              Standard Wavelet Transform.                     */
/* ************************************************************ */

          for(i = 0;i <= hi_level;i++)
               {
               coeff = (int)((double)count/pow(2.0,(double)(i+1)));
               if(coeff>(int)(filt_len/2))
                    {
                    off_set = (coeff*2);
                    }
               for(j = 0;j < coeff;j++)
                    {
                    approx[i][j] = 0.0;
                    detail[i][j] = 0.0;
                    for(k = 0;k < filt_len;k++)
                         {
                         approx[i][j] += (wksp[(k+(2*j)+off_set-1)
                                      % (2*coeff)]) * h[k];
                         detail[i][j] += (wksp[(k+(2*j)+off_set-2)
                                      % (2*coeff)]) * g[k];
                         }
                    }
               for(j=0;j<coeff;j++)
                    {
                    wksp[j] = approx[i][j];
                    }
               }

/*   Fill up the coefficient matrix and convert coefficients to
absolute values. */

          for(i = 0;i <= hi_level;i++)
               {
               m = count - 1;
               for(k = (int)((double)count/pow(2.0,
                  (double)(i+1)))-1;k >= 0;k--)
                    {
                    for(j = 0;j < (int)pow(2.0,(double)(i+1));j++)
                        {
```

```
                    detail[i][m] = detail[i][k];
                    m--;
                    }
               }
          }

/* ************************************************************ */
        }
         else
         {
/* ************************************************************ */
/*             Over-sampled Wavelet Transform.                  */
/* ************************************************************ */

         for(i = 0;i <= hi_level;i++)
              {
              off_set = count;
              for(j = 0;j < count;j++)
                   {
                   approx[i][j] = 0.0;
                   detail[i][j] = 0.0;
                   coeff = (int)pow(2.0,(double)i);
                   for(k = 0;k < filt_len;k++)
                        {
                        approx[i][j] += wksp[(j+(coeff*k)+
                                  off_set-1) % count] * h[k];
                     detail[i][j] += wksp[(j+(coeff*k)+
                                off_set-2) % count] * g[k];
                   }
                    }
              for(j=0;j<count;j++)
                   {
                   wksp[j] = approx[i][j];
                   }
              }

/* ************************************************** */
    }

         for(i = 0;i < num_levels;i++)
              {
              for(j = 0;j < utt_length;j++)
                   {
                   detail[i][j] = detail[i+lo_level][j];
                   }
              }

/*   Break detail matrix into windows containing the average of the
    coefficients in the window. */
         for(i = 0;i < num_levels;i++)
              {
```

```
                for(j = utt_length;j < count;j++)
                     {
                     detail[i][j] = 0.0;
                     }
                }

           ham_factor = 8.0*(float)(atan(1.0)/
                          (double)(samples_per_window-1));

           for(i=0;i<max_windows;i++)
                {
                for(j=0; j < 40; j++) feature_vector[j] = 0.0;
                win_start = (int)(i*samples_per_window/2);
                for(j=0;j<num_levels;j++)
                     {
                     work = 0.0;
                     for(k=0;k<samples_per_window;k++)
                          {
                          ham_val = 0.54-(0.46* (float)cos(
                                    (double)ham_factor*(double)k));
                          work += detail[j][k+win_start]*ham_val;
                          }
                     feature_vector[j] = work/
                                         (float)samples_per_window;
                    }
/*
                win_zc = calc_win_zero_cross(signal_ptr,
                          samples_per_window);
                feature_vector[num_levels] = (float)win_zc/1000.0;
                win_energy = calculate_window_energy(signal_ptr,
                             samples_per_window);
                feature_vector[num_levels+1] =
                          (float)win_energy/1000.0;
*/
                signal_ptr += samples_per_window/2;
                for(j=0;j<num_levels;j++)
                     fwrite(&feature_vector[j],sizeof(float),
                         1,feature_file);

                num_ftrs = num_levels;
                }

           free_vector(wksp,0,count - 1);
           free_matrix(approx,0,hi_level,0,count - 1);
           free_matrix(detail,0,hi_level,0,count - 1);
           free_vector(h,0,filt_len - 1);
           free_vector(g,0,filt_len - 1);

           }

      total_FV += max_windows;
```

```
     index_array[filecount+1] = total_FV;

/* ************************************************************** */

/*end of : while(!feof(list_file)) */
     }


/* ................. output the index file .................*/

index_file=fopen("results/index.dat","w");
if(!index_file)
     {
     printf("Cant open the index file\n");
     exit(1);
     }

index_array[0] = total_FV;
index_array[1] = num_ftrs;
for(i=0;i <= filecount+1; i++)
     fprintf(index_file,"%d\n", index_array[i]);

/* ..........................................................*/


fclose(list_file);
fclose(feature_file);
fclose(index_file);
free(samp);

return(0);

/* end of : main */
}

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

```
/* ********************************************************************
   Accepts a pointer to an array of floats and then calculates
   the absolute energy as defined eqn (1) in article by Rabiner
   and Sambur, Bell System Technical Journal, Vol.54, No.2,
   Feb.1975, pp 297-315
  ******************************************************************** */

float calculate_window_energy(float *ptr, int samples_per_window)
{
   int i = 0;
   float win_energy = 0;

   for (i = 0 ; i < samples_per_window ; i++)
   {
     if(ptr[i] > 0)
        win_energy += ptr[i];
     else
        win_energy -= ptr[i];
    }

    return( win_energy );
}

/* ********************************************************************
   Accepts a pointer to an array of floats and then calculates
   the number of zero crossings per window
 ******************************************************************** */

int calc_win_zero_cross(float *ptr, int samples_per_window) {
int win_zc = 0,
       counta = 0;

   unsigned short int  oldflag = 0,
                newflag = 0;

   if(ptr[0] >= 0.0)
     oldflag = 1;
     else
     oldflag = 0;

   for (counta = 1 ; counta < samples_per_window ; counta++)    {
     if(ptr[counta] >= 0.0)
      newflag = 1;
      else
      newflag = 0;

      if((oldflag != newflag)) win_zc++;

      oldflag = newflag;
   }
```

```
return(win_zc);

}

/* ********************************************************************
   This procedure accepts a 12 bit signed twos complement
   and returns a floating point number.
   Specifically written to convert NIST/TIMIT files and
   is preceded by the following :
    fread( &temp_in, 2, 1, data_file);

   Author : D.N.Prescott May 93
   ******************************************************************** */

float convert_12_bit_to_float(unsigned short int temp_in)
{
   int data_in;
   float signal;

     if( temp_in & 2048 )
     {
        temp_in = -temp_in & 2047;
        data_in  = temp_in;
        data_in  = -1 * data_in;
        signal = (float) data_in;
     }
     else
        signal = (float)(temp_in);
   return(signal);
}

/* ********************************************************************
  This procedure reads out the values stored in the NIST header
  such as number of samples in the file, file max, file min,
  sampling rate etc.
   Specifically written to convert NIST/TIMIT files and
   is preceded by the following :
    status = fread( NIST_header, 1, 1024, data_file);
    if(status != 1024)
    {
     printf("\nError reading header of file %s\n",data_file_name);
     exit(1);
    }

   Author : D.N.Prescott May 93
   ******************************************************************** */

int get_NIST_header_value(char NIST_header[1024],
                          char find_string[])
{
```

```
    char NIST_string[17],
      count[5],
      *temp_ptr;

    int i = 0;

    strcpy (NIST_string,find_string);
    temp_ptr = strstr( NIST_header, NIST_string);
    temp_ptr += strlen(NIST_string);
    strcpy(count, "");

    while(temp_ptr[i] != ' ')
    {
       count[i] = temp_ptr[i];
       i++;
    }
    count[i+1] = '\0';

    return( atoi(count) );
}

/*
*********************************************************************
fft - In-place radix 2 decimation in time FFT
Requires pointer to complex array, x and power of 2 size of FFT, m
(size of FFT = 2**m).  Places FFT output on top of input COMPLEX
array.

Taken directly from "DIGITAL SIGNAL PROCESSING IN C" by
 PRENTICE HALL, 1992
********************************************************************* */

void fft(COMPLEX *x, int m)
{
    static COMPLEX *w;     /* used to store the w complex array */
    static int mstore = 0; /* stores m for future reference */
    static int n = 1;      /* length of fft stored for future */

    COMPLEX u,temp,tm;
    COMPLEX *xi,*xip,*xj,*wptr;

    int i,j,k,l,le,windex;

    double arg,w_real,w_imag,wrecur_real,wrecur_imag,wtemp_real;

    if(m != mstore) {

/* free previously allocated storage and set new m */

     if(mstore != 0) free(w);
     mstore = m;
```

```
    if(m == 0) return;        /* if m=0 then done */

/* n = 2**m = fft length */

    n = 1 << m;
    le = n/2;

/* allocate the storage for w */

    w = (COMPLEX *) calloc(le-1,sizeof(COMPLEX));
    if(!w) {
        printf("\nUnable to allocate complex W array\n");
        exit(1);
    }

/* calculate the w values recursively */

    arg = 4.0*atan(1.0)/le;          /* PI/le calculation */
    wrecur_real = w_real = cos(arg);
    wrecur_imag = w_imag = -sin(arg);
    xj = w;
    for (j = 1 ; j < le ; j++) {
        xj->real = (float)wrecur_real;
        xj->imag = (float)wrecur_imag;
        xj++;
        wtemp_real = wrecur_real*w_real - wrecur_imag*w_imag;
        wrecur_imag = wrecur_real*w_imag + wrecur_imag*w_real;
        wrecur_real = wtemp_real;
    }
   }

/* start fft */

   le = n;
   windex = 1;
   for (l = 0 ; l < m ; l++) {
    le = le/2;

/* first iteration with no multiplies */

    for(i = 0 ; i < n ; i = i + 2*le) {
        xi = x + i;
        xip = xi + le;
        temp.real = xi->real + xip->real;
        temp.imag = xi->imag + xip->imag;
        xip->real = xi->real - xip->real;
        xip->imag = xi->imag - xip->imag;
        *xi = temp;
    }

/* remaining iterations use stored w */
```

```
    wptr = w + windex - 1;
    for (j = 1 ; j < le ; j++) {
        u = *wptr;
        for (i = j ; i < n ; i = i + 2*le) {
         xi = x + i;
         xip = xi + le;
         temp.real = xi->real + xip->real;
         temp.imag = xi->imag + xip->imag;
         tm.real = xi->real - xip->real;
         tm.imag = xi->imag - xip->imag;
         xip->real = tm.real*u.real - tm.imag*u.imag;
         xip->imag = tm.real*u.imag + tm.imag*u.real;
         *xi = temp;
        }
        wptr = wptr + windex;
    }
    windex = 2*windex;
   }

/* rearrange data by bit reversing */

    j = 0;
    for (i = 1 ; i < (n-1) ; i++) {
     k = n/2;
     while(k <= j) {
         j = j - k;
         k = k/2;
     }
     j = j + k;
     if (i < j) {
         xi = x + i;
         xj = x + j;
         temp = *xj;
         *xj = *xi;
         *xi = temp;
     }
    }
}

/*********************************************************************
ham - Hamming window
Scales both real and imaginary parts of input array in-place.
Requires COMPLEX pointer and length of input array.
*********************************************************************/

void ham(COMPLEX *x, int n)
{
  int    i;
  double ham,factor;
```

```
  factor = 8.0*atan(1.0)/(n-1);
  for (i = 0 ; i < n ; i++){
    ham = 0.54 - 0.46*cos(factor*i);
    x->real *= ham;
    x->imag *= ham;
    x++;
  }
}

#undef NRANSI
```

Appendix D. Karhunen-Loève Transform Program

```
/*  ************************************************************
Program Name : klt.c

Thesis Task :  Isolated Word, Speaker Dependent, Speech Recognition
               Using Wavelet Analysis

Program Task : Transform the speech features in to KLT space in
               preparation for the Classification routine

Author :  FLTLT Steve Ainge
          (with assistance from LT John G. Keller and
          FLTLT D. Neale Prescott for much of the framework code)

Last Update :       14 Nov 93

             *         *         *         *          *

Inputs:   the.dat (file containing feature vectors of all
          utterances and windows)
          index.dat

Outputs:  eigvectors.dat (file containing weighted eigenvectors
          used to transform test vectors into new feature space)
          ones...zeros (files containing the transformed vectors
          corresponding to one through zero utterances.

*********************************************************************/

#include<stdio.h>
#include <malloc.h>
#include <string.h>
#include "nrutil.h"
#include <math.h>

#define MSE_ENERGY 1.0

main()
{
FILE *trainfile, *handle, *eigfile, *newvectorfile, *index_file;

int  numftrs, numvectors, nrot, i, j, k, l, m, n, window_count,
     old_w_count, utterance_count, spkr, numberofvectors[11],
     numberofwindows[60], value, digit, len, status;

float     *eigvalues, **eigvectors, *meanvector, **trainvectors,
          **covar, keepvalue, covarsum, num, ftrsum, eigtotal,
          keeptotal, vectotal, vecsubtotal, singlemeanvector;

char *numberfile[10], *filehandle, temp_str[30];
```

```
/*
Allocate space for our matrices and arrays, and do any other
necessary initialization.
*/

m = 1;
keepvalue = eigtotal = vectotal = 0.0;
covarsum = num = 0.0;
numberfile[0] = "00";
numberfile[1] = "01";
numberfile[2] = "02";
numberfile[3] = "03";
numberfile[4] = "04";
numberfile[5] = "05";
numberfile[6] = "06";
numberfile[7] = "07";
numberfile[8] = "08";
numberfile[9] = "09";

/*
Bring in the header information that will tell us how many vectors
and windows are associated with each word (all repetitions). */

if ((trainfile = fopen("results/the.dat", "rb")) == NULL)
     {
     printf("\nUnable to open the file containing the training
             data.");
     exit(0);
     }

if ((index_file = fopen("results/index.dat", "r")) == NULL)
     {
     printf("\nUnable to open file 'index.dat.'");
     exit(0);
     }

/*
Read in the number of vectors and features from the training data
file, allocate space for the vectors and matrices using that
information, and then read in the next 20 values, which will be the
number of vectors and windows corresponding to each individual
word.
*/

fscanf(index_file, "%d", &numvectors);
fscanf(index_file, "%d", &numftrs);
meanvector = vector(1, numftrs);
eigvalues = vector(1, numftrs);
eigvectors = matrix(1, numftrs, 1, numftrs);
trainvectors = matrix(1,numvectors, 1, numftrs);
covar = matrix(1, numftrs, 1, numftrs);
```

```
/*
Bring in all the training data and place it in a two dimensional
array "trainvectors."
*/

for (i = 1;i <= numvectors;i++)
     {
     for (j = 1;j <= numftrs;j++)
          {
          fread(&num,sizeof(float),1,trainfile);
          trainvectors[i][j] = num;
          }
     }
fclose(trainfile);

/*
Next, we will build the mean vector, which will contain the mean of
each feature over the entire set of input feature vectors.  Do it
by averaging feature 1 over all vectors, then feature 2, etc. */

ftrsum = 0.0;
for (i = 1;i <= numftrs;i++)
     {
     for (j = 1;j <= numvectors;j++)
          {
          ftrsum += trainvectors[j][i];
          }
     meanvector[i] = ftrsum/numvectors;
     ftrsum = 0.0;
     }
/*
Build the covariance matrix.
*/

covarsum = 0.0;
for (i = 1;i <= numftrs;i++)
     {
     for (j = i;j <= numftrs;j++)
          {
          for (k = 1;k <= numvectors;k++)
               {
               covarsum += (trainvectors[k][i] - meanvector[i]) *
                       (trainvectors[k][j] - meanvector[j]);
               }
          covar[i][j] = covar[j][i] = covarsum/numvectors;
          covarsum = 0.0;
          }
     }

/*
```

```
Now do some eigenstuff.  Determine the eigenvalues and vectors,
then sort them in order of descending eigenvalue.
*/

jacobi(covar, numftrs, eigvalues, eigvectors, &nrot);
eigsrt(eigvalues, eigvectors, numftrs);
free_matrix(covar,1,numftrs,1,numftrs);

/*
Now let's do the transformation to reduce the dimensionality of the
original vectors.
*/

for (i = 1;i <= numftrs;i++)
     {
     eigtotal += eigvalues[i];
     }

/*
m is the number of features you wish to keep
*/

m = 1;
for(i = 1;i <= m;i++)
     {
     keepvalue += eigvalues[i]/eigtotal;
     }

/*************************************************************
Save the eigenvectors we'll be using into a file.  The file will be
used to redimensionalize test vectors.

The mean vector is also saved
*************************************************************/
if ((eigfile = fopen("results/eigvectors.dat", "w")) == NULL)
     {
     printf("\nUnable to open file 'eigvectors.dat.'");
     exit(0);
     }

/* ............ Print out the number of dimension ...........*/

fprintf(eigfile, "%d\n", m);

/* ................ Print out the mean vector ..............*/

for(i = 1; i <= numftrs; i++)
  fprintf(eigfile, "%12.8f\n", meanvector[i]);

/* ..........................................................
Weight the values of the eigenvectors by multiplying by the
```

```
reciprocal of the corresponding eigenvalue's contribution to the
total.
.....................................................*/

for (i = 1;i <= m;i++)
     {
     for (j = 1;j <= numftrs;j++)
          {
          eigvectors[j][i] = eigvectors[j][i]/sqrt(eigvalues[i]);
          fprintf(eigfile,"%12.8f\t", eigvectors[j][i] );
          }
     fprintf(eigfile, "\n");
     }
fclose(eigfile);

/*
Create new files containing the transformed feature vectors */

fscanf(index_file, "%d",&window_count);

vectotal = 0.0;
utterance_count = 0;
digit = 0;
spkr = 1;

strcpy(temp_str,"results/");
strncat(temp_str, "01_0_",5);
strncat(temp_str, numberfile[digit],2);

if ((newvectorfile = fopen( temp_str, "w") ) == NULL)
     {
     printf("\nUnable to open the file for writing the transformed
               vectors.");
     exit(0);
     }

old_w_count = 0;

fprintf(newvectorfile,"%d\n%d\n", (window_count - old_w_count), m);

for (i = 1;i <= numvectors;i++)
     {
     for (j = 1;j <= m;j++)
          {
          for (k = 1;k <= numftrs;k++)
               {
               vectotal += (trainvectors[i][k] - meanvector[k]) *
                              eigvectors[k][j];
               }
          fprintf(newvectorfile,"%f\n", vectotal);
          vectotal = 0.0;
```

```
    }

if( (i % window_count) == 0)
    {
    old_w_count = window_count;
    status = fscanf(index_file,"%d", &window_count);
    if(status != 1) break;

    utterance_count++;

    if( ( (spkr * utterance_count) % 80) == 0)
        digit++;

    if( (utterance_count % 10 ) == 0)
        {
        utterance_count = 0;

        if( (spkr % 8) == 0 )
            spkr = 1;
        else
            spkr++;

        }

    fclose( newvectorfile);

    strcpy(temp_str,"results/");

    switch(spkr)
        {
        case 1 : strncat(temp_str, "01_",3);break;
        case 2 : strncat(temp_str, "02_",3);break;
        case 3 : strncat(temp_str, "03_",3);break;
        case 4 : strncat(temp_str, "04_",3);break;
        case 5 : strncat(temp_str, "05_",3);break;
        case 6 : strncat(temp_str, "06_",3);break;
        case 7 : strncat(temp_str, "07_",3);break;
        case 8 : strncat(temp_str, "08_",3);break;
        }

    switch( utterance_count)
        {
        case 0 : strncat(temp_str, "0_",2);break;
        case 1 : strncat(temp_str, "1_",2);break;
        case 2 : strncat(temp_str, "2_",2);break;
        case 3 : strncat(temp_str, "3_",2);break;
        case 4 : strncat(temp_str, "4_",2);break;
        case 5 : strncat(temp_str, "5_",2);break;
        case 6 : strncat(temp_str, "6_",2);break;
        case 7 : strncat(temp_str, "7_",2);break;
        case 8 : strncat(temp_str, "8_",2);break;
```

```
                case 9 : strncat(temp_str, "9_",2);break;
                }

          strncat(temp_str, numberfile[digit],2);

          if( (newvectorfile = fopen( temp_str, "w") ) == NULL)
                {
                printf("\nUnable to open the file for writing the
                        transformed vectors.");
                exit(0);
                }
          fprintf(newvectorfile,"%d\n%d\n", (window_count -
                    old_w_count), m);

          }

     }

fclose(index_file);
fclose(newvectorfile);

printf("\tKLT - Dimension reduced from %d to %d retaining %f of the
         energy\n",numftrs,m,keepvalue);

free_vector(eigvalues, 1, numftrs);
free_vector(meanvector, 1, numftrs);
free_matrix(eigvectors,1,numftrs,1,numftrs);
free_matrix(trainvectors,1, numvectors, 1, numftrs);

return(0);
}
```

Appendix E. Reference Selection Program

```
#include<stdio.h>
#include <string.h>
\*              60 & 100 for 16ms window
                120 & 200 for 8ms window
                240 & 400 for 4ms window */
#define LOWER_PROTOTYPE 240
#define UPPER_PROTOTYPE 400

/* ************************************************************

   Author : D.N.Prescott
   Date : May 1993

   This program : nameout.c -

   Input file - is the INDEX_FILE produced by the feature
                extraction program. The file contains the
                cumulative feature vector number for each of the
                eight speakers, for each of the ten digits, and
                each of the ten utterances

   Output file - None. STDOUT

   Operation - The length of each word is placed in an 3 dimension
               array called COUNT -

               index [0] is the speaker
               index [1] is the digit
               index [3] is the utterance

               The program then searches the array looking for a
               prototype for each digit for each speaker. The
               result of the search is placed in the array CLOSEST.

               The program then outputs the name of the prototypes
               chosen and the number of feature vectors for that
               prototype

************************************************************************/

main()
{
int  val = 0,
     old_val = 0,
     utterance,
     window_count, best[2],
     old_w_count,
     spkr,closest[9][10][4];

int      value, digit, len, count[9][10][10], FLAG = 0;
```

```
char *numberfile[10], *filehandle, temp_str[30], temp_str2[30];

FILE  *index_file;

/* ..........................................................*/

numberfile[0] = "00";
numberfile[1] = "01";
numberfile[2] = "02";
numberfile[3] = "03";
numberfile[4] = "04";
numberfile[5] = "05";
numberfile[6] = "06";
numberfile[7] = "07";
numberfile[8] = "08";
numberfile[9] = "09";

/* ..........................................................*/

if ((index_file = fopen("results/index.dat", "r")) == NULL)
{
   printf("\nUnable to open file 'index.dat.'");
   exit(0);
}

old_w_count = 0;

for( spkr = 1; spkr < 9;  spkr++)
{
   for(digit = 0; digit < 10; digit++)
   {
      for(utterance = 0; utterance < 10; utterance++)
      {
      fscanf(index_file, "%d",&window_count);
      count[spkr][digit][utterance] = window_count - old_w_count;
      old_w_count = window_count;
      }
   }
}

fclose(index_file);

/* ..........................................................*/

for (spkr = 1;spkr < 9; spkr++)
{
   for(digit = 0; digit < 10; digit++)
   {
      old_val = 0;
      FLAG = 0;
```

```
        best[0] = 0;
        best[1] = 500;

        for( utterance = 0; utterance < 10; utterance++)
        {
        val = count[spkr][digit][utterance];

        if( ((val - LOWER_PROTOTYPE) > 0 ) && (val < best[1]) )
           {
           best[0] = utterance;
           best[1] = val;
           }

        if( (val <= LOWER_PROTOTYPE) && (val > old_val) )
           {
           closest[spkr][digit][0] = utterance;
           old_val = val;
           closest[spkr][digit][1] = val;
           FLAG = 1;
           }
        }
        if(!FLAG)
        {
        closest[spkr][digit][0] = best[0];
        closest[spkr][digit][1] = best[1];
        }

     }
}

/* ..........................................................*/

for (spkr = 1;spkr < 9; spkr++)
{
   for(digit = 0; digit <10; digit++)
   {
      old_val = 500;
      FLAG = 0;

      best[0] = 0;
      best[1] = 0;

      for( utterance = 0; utterance < 10; utterance++)
      {
      val = count[spkr][digit][utterance];

      if( ((UPPER_PROTOTYPE - val) > 0 ) && (val > best[1]) )
         {
         best[0] = utterance;
         best[1] = val;
```

```
            }

         if( (val >= UPPER_PROTOTYPE) && (val < old_val) )
            {
            closest[spkr][digit][2] = utterance;
            old_val = val;
            closest[spkr][digit][3] = val;
            FLAG = 1;
            }
         }
         if(!FLAG)
         {
         closest[spkr][digit][2] = best[0];
         closest[spkr][digit][3] = best[1];
         }
      }
   }

   /* ..........................................................*/

   printf("#! /bin/csh\n");

   for(digit = 0; digit < 10; digit++)
   {
      for (spkr = 1;spkr < 9; spkr++)
      {
         utterance = closest[spkr][digit][0];

         strcpy(temp_str,"references/");
         len = strlen("ref1");
         strncat(temp_str, "ref1",len);

         strcpy(temp_str2,"results/");

         switch(spkr)
         {
             case 1 : strncat(temp_str,  "1",1);
                   strncat(temp_str2, "01_",3);
                   break;

             case 2 : strncat(temp_str,  "2",1);
                   strncat(temp_str2, "02_",3);
                   break;

             case 3 : strncat(temp_str,  "3",1);
                   strncat(temp_str2, "03_",3);
                   break;

             case 4 : strncat(temp_str,  "4",1);
                   strncat(temp_str2, "04_",3);
                   break;
```

```
    case 5 : strncat(temp_str,  "5",1);
             strncat(temp_str2, "05_",3);
             break;

    case 6 : strncat(temp_str, "6",1);
             strncat(temp_str2, "06_",3);
             break;

    case 7 : strncat(temp_str,  "7",1);
             strncat(temp_str2, "07_",3);
             break;

    case 8 : strncat(temp_str,  "8",1);
             trncat(temp_str2, "08_",3);
             break;
}

switch(utterance)
{
    case 0 : strncat(temp_str2, "0_",2);break;
    case 1 : strncat(temp_str2, "1_",2);break;
    case 2 : strncat(temp_str2, "2_",2);break;
    case 3 : strncat(temp_str2, "3_",2);break;
    case 4 : strncat(temp_str2, "4_",2);break;
    case 5 : strncat(temp_str2, "5_",2);break;
    case 6 : strncat(temp_str2, "6_",2);break;
    case 7 : strncat(temp_str2, "7_",2);break;
    case 8 : strncat(temp_str2, "8_",2);break;
    case 9 : strncat(temp_str2, "9_",2);break;
}

switch(digit)
{
    case 0 : strncat(temp_str, "0",1);break;
    case 1 : strncat(temp_str, "1",1);break;
    case 2 : strncat(temp_str, "2",1);break;
    case 3 : strncat(temp_str, "3",1);break;
    case 4 : strncat(temp_str, "4",1);break;
    case 5 : strncat(temp_str, "5",1);break;
    case 6 : strncat(temp_str, "6",1);break;
    case 7 : strncat(temp_str, "7",1);break;
    case 8 : strncat(temp_str, "8",1);break;
    case 9 : strncat(temp_str, "9",1);break;
}

len = strlen(numberfile[digit]);
strncat(temp_str2,numberfile[digit] ,len);

printf("cp %s %s\n", temp_str2, temp_str   );
```

```
utterance = closest[spkr][digit][2];

strcpy(temp_str,"references/");
len = strlen("ref2");
strncat(temp_str, "ref2",len);

strcpy(temp_str2,"results/");

switch(spkr)
{
    case 1 : strncat(temp_str,  "1",1);
             strncat(temp_str2, "01_",3);
             break;

    case 2 : strncat(temp_str,  "2",1);
             strncat(temp_str2, "02_",3);
             break;

    case 3 : strncat(temp_str,  "3",1);
             strncat(temp_str2, "03_",3);
             break;

    case 4 : strncat(temp_str,  "4",1);
             strncat(temp_str2, "04_",3);
             break;

    case 5 : strncat(temp_str,  "5",1);
             strncat(temp_str2, "05_",3);
             break;

    case 6 : strncat(temp_str, "6",1);
             strncat(temp_str2, "06_",3);
             break;

    case 7 : strncat(temp_str,  "7",1);
             strncat(temp_str2, "07_",3);
             break;

    case 8 : strncat(temp_str,  "8",1);
             strncat(temp_str2, "08_",3);
             break;
}

switch(utterance)
{
    case 0 : strncat(temp_str2, "0_",2);break;
    case 1 : strncat(temp_str2, "1_",2);break;
    case 2 : strncat(temp_str2, "2_",2);break;
    case 3 : strncat(temp_str2, "3_",2);break;
    case 4 : strncat(temp_str2, "4_",2);break;
    case 5 : strncat(temp_str2, "5_",2);break;
```

```
            case 6 : strncat(temp_str2, "6_",2);break;
            case 7 : strncat(temp_str2, "7_",2);break;
            case 8 : strncat(temp_str2, "8_",2);break;
            case 9 : strncat(temp_str2, "9_",2);break;
        }

        switch(digit)
        {
            case 0 : strncat(temp_str, "0",1);break;
            case 1 : strncat(temp_str, "1",1);break;
            case 2 : strncat(temp_str, "2",1);break;
            case 3 : strncat(temp_str, "3",1);break;
            case 4 : strncat(temp_str, "4",1);break;
            case 5 : strncat(temp_str, "5",1);break;
            case 6 : strncat(temp_str, "6",1);break;
            case 7 : strncat(temp_str, "7",1);break;
            case 8 : strncat(temp_str, "8",1);break;
            case 9 : strncat(temp_str, "9",1);break;
        }

        len = strlen(numberfile[digit]);
        strncat(temp_str2,numberfile[digit] ,len);

        printf("cp %s %s\n", temp_str2, temp_str   );

    }
    printf("\n\n");
}

return(0);

}
```

Appendix F. Dynamic Time Warp Classification Program

```
/*  ************************************************************
Program Name : classify.c

Thesis Task :  Isolated Word, Speaker Dependent, Speech Recognition
               Using Wavelet Analysis

Program Task : To classify the files that are produced by the KLT
               routine, using 2 prototype utterances.

Author :  FLTLT Steve Ainge
          converted from Fortran-77 subroutine WARP in
               Voice and Speech Processing
               by Thomas Parsons
               McGraw-Hill 1987
               Appendix B

Last Update :       14 Nov 93

            *        *        *        *          *

     Implements DP time-warping procedure to align two patterns,
     A & B. Returns resultant minimum cost & the warping function
     (called PATH). Stages in DP process are samples of pattern A.
     Maximum template length is limited by COST & PRED arrays.

     Variables:
          Inputs
          a    first pattern
          m    # of points in A
          b    second pattern
          n    # of points in B
          Outputs
          tcost     cost of optimum path
          path      encoded optimum path
          k         length of optimum path
          Internal
          cost      array of recent accrued costs
          pred      array of predecessor points

     To minimize storage requirements:
     (a) Accrued costs are retained for only two most recent rows
         of points; at the end of a row, costs for row 2 (current
         row) replace costs for row 1 (previous row), which are
         no longer needed.
     (b) Paths and predecessor-point coordinates are compressed
         into a single 16-bit integer by means of the function,
         PFUNC.
*/

#include <math.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define maxnum(i,j) i>j ? i: j
#define minnum(i,j) i<j ? i: j

/*..................... FUNCTION.........................*/

double cfunc(float x[], float y[],int mfeatures)
{
     int l;
     double c;
     c = 0.0;
     for(l=1;l<=mfeatures;l++)
          {
          c += (x[l]-y[l])*(x[l]-y[l]);
          }
     c = sqrt(c);

     return (c);
}

/*..................... FUNCTION .........................*/

long int pfunc(int i, int j)
{
     return ((65536*(long int)i)+(long int)j);
}

/*..................... MAIN.........................*/

main(int argc,char *argv[])
{
FILE      *in1,*in2,*test_file;

int  i,ii,j,k,kk,m,n,ref,digit,lim1,lim2,
     mvectors,nvectors,mfeatures,nfeatures,best_match,status,
correct = 0,errors = 0,offset;

long int path[2000],pred[1000][1000];

float     x[20],y[20],a[1000][20],b[1000][20],accuracy;

double    BIG = 10e37,c,c1,c2,c3,tcost,cost[3][1000],
          best_match_cost;

char      ch[2],test_file_name[50],ref_file_name[50],spkr[2];

/* Check all command line arguments present */
if(argc!=2)
     {
```

```
     printf("Need to provide name of file containing test list.");
     exit(1);
     }
/*.......... Open the file of files to be classified ..........*/

/* Open file to get test files. */
if((test_file =fopen(argv[1],"r"))==NULL)
     {
     printf("Cannot open file\n");
     exit(1);
     }
     /* Read in files. */
while(!feof(test_file))
     {
     status = fscanf(test_file,"%s",&test_file_name);
     if(status != 1) break;

/*..............Open the file to be classified ..................*/

     /* Open files to get data. */
     if((in1 =fopen(test_file_name,"r"))==NULL)
          {
          printf("Cannot open test file\n");
          exit(1);
          }

     /* Read in files. */
     fscanf(in1,"%d",&mvectors);
     fscanf(in1,"%d",&mfeatures);
     for(m=1;m<=mvectors;m++)
          for(i=1;i<=mfeatures;i++)
               {
               fscanf(in1,"%f",&a[m][i]);
               }
     fclose(in1);

     /* Initialize variables. */
     best_match = 10;
     best_match_cost = BIG;

/*.............. Open the REFERENCE files  ...................*/

     spkr[0] = test_file_name[9];
     spkr[1] = '\0';

     for(digit=0;digit<=9;digit++)
          {
          for(ref=1;ref<=2;ref++)
               {

               strcpy(ref_file_name,"references/ref");
```

```
                if(ref == 1) ch[0] = '1';
                if(ref == 2) ch[0] = '2';
                ch[1] = '\0';
                strcat(ref_file_name,ch);

                strcat(ref_file_name,spkr);
                if(digit == 0) ch[0] = '0';
                if(digit == 1) ch[0] = '1';
                if(digit == 2) ch[0] = '2';
                if(digit == 3) ch[0] = '3';
                if(digit == 4) ch[0] = '4';
                if(digit == 5) ch[0] = '5';
                if(digit == 6) ch[0] = '6';
                if(digit == 7) ch[0] = '7';
                if(digit == 8) ch[0] = '8';
                if(digit == 9) ch[0] = '9';
                ch[1] = '\0';
                strcat(ref_file_name,ch);

                if((in2 =fopen(ref_file_name,"r"))==NULL)
                     {
                     printf("Cannot open ref file %s\n",
                               ref_file_name);
                     exit(1);
                     }

                /* Read in files. */
                fscanf(in2,"%d",&nvectors);
                fscanf(in2,"%d",&nfeatures);
                for(n=1;n<=nvectors;n++)
                     for(i=1;i<=nfeatures;i++)
                          {
                          fscanf(in2,"%f",&b[n][i]);
                          }
                fclose(in2);

/* ..............Start the other Stuff .......................*/

                /* Clean up predecessor array. */
                for(i=0;i<=999;i++)
                     for(j=0;j<=999;j++)
                          pred[i][j] = 0;

                for(i=0;i<=2;i++)
                     for(j=0;j<=999;j++)
                          cost[i][j] = 0.0;

                /* Loop through A. */
                for(i=1;i<=mvectors;i++)
                     {
                     offset = maxnum(50,abs(nvectors-mvectors));
```

```
lim1 = maxnum(1,i-offset);
lim2 = minnum(nvectors,i+offset);

/* Loop through B. */
for(j=lim1;j<=lim2;j++)
     {

     /* Cost for this point. */
     for(k=1;k<=nfeatures;k++)
          {
          x[k] = a[i][k];
          y[k] = b[j][k];
          }
     c = cfunc(x,y,mfeatures);

     /* Cost for path to this point. */
     if(i==1 && j==1)

          /* No predecessors */
          {
          cost[2][j] = c;
          pred[i][j] = pfunc(1,1);
          }
     else
          {

          /* Must consider 3 predecessors */
          c1 = BIG;
          if(pred[i-1][j]>0)
               {
               c1 = cost[1][j]+c;
               }
          c2 = BIG;
          if(pred[i-1][j-1]>0)
               {
               c2 = cost[1][j-1]+c;
               }
          c3 = BIG;
          if(pred[i][j-1]>0)
               {
               c3 = cost[2][j-1]+c;
               }
          /* Find cheapest */
          if(c1>=BIG && c2>=BIG && c3>=BIG)
               {
               pred[i][j] = 0;
               }
          else if(c1<c2 && c1<=c3)
               {
               cost[2][j] = c1;
               pred[i][j] = pfunc(i-1,j);
```

```
                    }
                else if(c2<=c1 && c2<=c3)
                    {
                    cost[2][j] = c2;
                    pred[i][j] = pfunc(i-1,j-1);
                    }
                else if(c3<c1 && c3<=c2)
                    {
                    cost[2][j] = c3;
                    pred[i][j] = pfunc(i,j-1);
                    }
                }
            }

            /* Shift costs down */
            for(j=lim1;j<=lim2;j++)
                {
                cost[1][j] = cost[2][j];
                }
            }

        /* ............ Note total cost................. */

        tcost = cost[2][nvectors];

/* ............. Determine best match ..............*/

        if(fabs(tcost) < best_match_cost)
            {
            best_match = digit;
            best_match_cost = fabs(tcost);
            }
        }

    }

ch[0] = test_file_name[14];
ch[1] = '\0';
if(atoi(ch) == best_match)
    {
    correct++;
    }
else
    {
    errors++;
    }
}

\* ......... Output Classification rate .....------*/

printf("\n\n\t\tNo. correct\t= %d\n",(correct-160));
```

```
        printf("\t\tNo. errors\t= %d\n",errors);
        accuracy = ((float) correct-160.0)/640.0;
        printf("\n\n\t\tACCURACY\t= %f\n",accuracy);
        fclose(test_file);

        return(0);
}
```

Appendix G. Re-run of Experiments 9 and 14 using Identical Windowing

Experiments 9 and 14 were re-run using an identical windowing technique. Previously, these experiments were run with:

1. Experiment 9 - oversampled wavelet, Db4 filter, 4ms window and no augmenting features. Windowing was performed after the complete signal was decomposed.
2. Experiment 14 - Fourier, 11 single octave bands, 4ms window and no augmenting features. Windowing was performed before the Fourier coefficients were calculated.

For the re-runs, the experiment parameters were kept the same, however, the windowing methods were made the same. The windowing method used in the re-runs was:

1. window the speech signal into a 4ms window (50 samples),
2. zero pad the window to 64 samples,
3. decompose into:
   (a) 6 levels of oversampled wavelet coefficients or
   (b) 6 single octave frequency bands of Fourier coefficients,

The remaining processing remained the same as used previously. The results of the re-runs are shown in Tables G.1 and G.2.

It can be seen from these tables that the maximum classification accuracy for the wavelet method was 26.4%, compared to 87.5% for the Fourier method. When these are compared to the results of the original Experiments 9 and 14:

1. the Fourier performance is virtually unchanged and
2. the wavelet performance is halved.

Therefore, the windowing method used originally, in this thesis, provides better recognition than the method used in the re-run. For recognition purposes, decomposing the complete signal then windowing provides better recognition features than windowing then decomposing.

Table G.1 Results of Experiment 9 Re-run

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	26.4	56.0
2	22.0	88.2
3	20.6	98.2
4	18.8	99.9
5	18.3	99.9
6	17.8	100.0

Table G.2 Results of Experiment 14 Re-run

KLT Coefficients Kept	*Classification Rate %*	*Relative Variance Kept %*
1	59.7	81.9
2	73.4	95.4
3	77.8	98.7
4	80.2	99.9
5	85.8	99.9
6	87.5	100.0

Bibliography

1. Daubechies, Ingrid. "Orthonormal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, *41*:909–996 (November 1988).
2. Kadambe, Shubha and G. Faye Boudreaux-Bartels. "A Comparison of Wavelet Functions for Pitch Detection in Speech Signals," *IEEE Conference on Acoustics, Speech and Signal Processing*, *1*:449–452 (1991).
3. Kadambe, Shubha and G. Faye Boudreaux-Bartels. "Application of the Wavelet Transform for Pitch Detection of Speech Signals," *IEEE Transactions on Information Theory*, *38*:917–924 (March 1992).
4. Keller, John G. *Identity Verification through the Fusion of Face and Speaker Recognition*. MS thesis, Air Force Institute of Technology (AFIT), 1993.
5. Mallat, Stephane G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*:674–693 (July 1989).
6. Parsons, Thomas. *Voice and Speech Processing*. New Jersey: McGraw-Hill, 1987.
7. Petropulu, Athina P. "Detection of Transients using Discrete Wavelet Transform," *IEEE Conference on Acoustics, Speech and Signal Processing*, *2*:477–480 (1992).
8. Pols, Louis C.W. "Real-Time Recognition of Spoken Words," *IEEE Transactions on Computers*, *20*:972–978 (September 1971).
9. Press, W.H., et al. *Numerical Recipes in C - The Art of Scientific Computing Second Edition*. New York: Cambridge University Press, 1992.
10. Rabiner, L.R. and B.H. Juang. "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine*, 4–16 (January 1986).
11. Rabiner, L.R. and M.R. Sambur. "An Algorithm for Determining the Endpoint of Isolated Utterances," *The Bell System Technical Journal*, *54*:297–315 (February 1975).
12. Reilly, A.P. and B. Boashash. "A Comparison of Time-Frequency Signal Analysis Techniques with Application to Speech Recognition," *Proceedings of SPIE - The International Society for Optical Engineering: Advanced Signal Processing Algorithms, Architectures and Implementations III*, *1770*:339–350 (1992).
13. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition*. Bellingham: SPIE Optical Engineering Press, 1991.
14. Sari-Sarraf, Hamed and Dragana Brzakovic. "One Dimensional Signal Classification by Multiscale Wavelet Representation," *Proceedings of SPIE - The International Society for Optical Engineering: Adaptive and Learning Systems*, *1706*:102–112 (1992).
15. Shartle, Gary. *Handwritten Word Recognition Based on Fourier Coefficients*. MS thesis, Air Force Institute of Technolcgy (AFIT), 1993.
16. Sinha, Deepen and Ahmed H. Tewfik. "Synthesis/Coding of Audio Signals using Optimized Wavelets," *IEEE Conference on Acoustics, Speech and Signal Processing*, *1*:113–116 (1992).
17. Suzuki, Laura R.C. "A Simplistic Speaker ID Method." Part of AFIT/ENG PhD Minor Exam Requirement EENG 617, 618, June 1993.
18. Tuteur, Franz B. "Wavelet Transformations in Signal Detection," *IEEE Conference on Acoustics, Speech and Signal Processing*, *2*:1435–1438 (1988).

Vita

Flight Lieutenant Stephen Ainge was born on May 10, 1959 in London, England. His family emigrated to Australia in 1966. He graduated from Thornlie Senior High School in 1975 and joined the Royal Australian Air Force in 1979 as a Telecommunications Technician. He spent six years, reaching the rank of Corporal, maintaining communication center equipment. In 1985, he began a Bachelor of Engineering degree at Curtin University of Technology (formerly Western Australian Institute of Technology). He graduated from Curtin University of Technology in 1988 and earned his Queen's commission. Flight Lieutenant Ainge then spent three years at the Royal Australian Air Force Logistics Command as the system's engineer responsible for in-service secure communications equipment. In June 1992, he entered the School of Engineering, Air Force Institute of Technology at Wright-Patterson Air Force Base, Ohio, to pursue a Master of Science in Electrical Engineering degree with an emphasis in pattern recognition, optics and signal processing. He is married to Debbie Ainge and they have three children, Rebecca, Ben and Nicole.

Permanent address: 1 Gilcoe Place
Roleystone, WA 6111
Australia