

DTIC FILE COPY

AD-A203 177



DTIC
ELECTE
JAN 17 1989
S H D

F-16 SPEAKER-INDEPENDENT
SPEECH RECOGNITION SYSTEM
USING COCKPIT COMMANDS (70 WORDS)

THESIS

Peter Y. Kim
Captain, USAF

AFIT/GE/ENG/88D-18

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

89

1

17

030

AFIT/GE/ENG/88D-18

F-16 SPEAKER-INDEPENDENT
SPEECH RECOGNITION SYSTEM
USING COCKPIT COMMANDS (70 WORDS)

THESIS

Peter Y. Kim
Captain, USAF

AFIT/GE/ENG/88D-18

DTIC
ELECTE
JAN 17 1989
S H D

Approved for public release; distribution unlimited

AFIT/GE/ENG/88D-18

F-16 SPEAKER-INDEPENDENT
SPEECH RECOGNITION SYSTEM
USING COCKPIT COMMANDS (70 WORDS)

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

Peter Y. Kim, B.S.
Captain, USAF

December 1988

Approved for public release; distribution unlimited

Acknowledgments

First, I would like to thank my thesis advisor, Dr. Matthew Kabrisky who provided invaluable guidance essential to the success of this study. Next, I would like to thank everyone who helped me in this effort, especially Dave Doak of the System Research Laboratory and Dr. David Vaughan of the Logistics School. This work would not have been possible without their cooperation. Most importantly, I express my deepest gratitude and love to my wife, [REDACTED] and my daughter, [REDACTED] for their sacrifices and constant encouragement.

Peter Y. Kim



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Table of Contents

	Page
Acknowledgments	ii
Table of Contents	iii
List of Figures	vi
List of Tables	vii
Abstract	viii
 I. Introduction	 1-1
Background	1-2
Problem Statement	1-3
Scope	1-3
Assumptions	1-4
General Approach	1-4
Materials and Equipment	1-5
Sequence of Presentation	1-5
 II. Background of the AFIT Speech Recognition System	 2-1
Computer Analysis and Recognition of Phoneme Sounds	2-1
Phoneme Analysis	2-1
Isolated Word Recognition Using Fuzzy Set Theory	2-3
Fuzzy Set Theory	2-4
Algebraic Properties	2-5
Limited Continuous Speech Recognition by Phoneme Analysis	2-6

	Page
Approaches and Techniques	2-7
Implementation of A Real-Time, Interactive, Continuous Speech Recognition System	2-9
Phoneme Representation	2-9
Distance Rules	2-10
Spatial Comparison	2-11
Speaker-Independent Word Recognition Using Multiple fea- tures, Decision Mechanisms, and Template Sets	2-11
Feature Categories	2-12
Processed Features	2-14
SPIRE Based Continuous Speech Reconition System	2-15
Dynamic Time Warping	2-15
III. Processing Environment	3-1
Lisp	3-1
SPIRE	3-1
Interfacing SPIRE from Lisp	3-2
Hardware	3-5
Lisp Machine	3-5
Array Processor	3-5
Speech Digitizer	3-8
IV. System Design	4-1
Utterance Processing	4-1
Feature Extraction	4-1
Other Processings	4-3
Dynamic Time Warping	4-5
Merged Template	4-5
Reference Utterance	4-7

	Page
Determining Nearest Index	4-7
Averaging Time Slice	4-8
Optimal template	4-10
Vocabulary	4-10
V. Testing and Results	5-1
Single Template with Connected Scan	5-1
Single Template with Isolated Scan	5-4
Merged Template	5-5
Additional Utterances	5-11
VI. Conclusions and Recommendations	6-1
Conclusions	6-1
Recommendations	6-2
Realistic Environment	6-2
Vocabulary Expansion	6-3
Connected Speech	6-3
Clustered Template	6-4
Different Sounds	6-4
Additional Features	6-4
Appendix A. Programming List	A-1
Appendix B. Sample Results	B-1
Bibliography	BIB-1
Vita	VITA-1

List of Figures

Figure	Page
2.1. Feature Sets Used in Brusuelas' Thesis	2-13
2.2. Hypothetical Distance Array for Continuous Speech	2-19
2.3. Schematic Diagram	2-20
2.4. Dynamic Time Warp Algorithm	2-21
3.1. List of SPIRE Displays	3-3
3.2. Example Capability of SPIRE	3-4
3.3. SPIRE Interface Functions	3-6
4.1. Three Features of Processing	4-4
4.2. Template Merging	4-6
4.3. F-16 Cockpit Commands	4-9

List of Tables

Table	Page
2.1. Speech Recognition Procedure	2-3
2.2. Minkowski Distances	2-10
2.3. Distance Arrays	2-17
3.1. SPIRE Result Arrays	3-7
3.2. Computational Time Comparison	3-8
5.1. Selected Utterances	5-2
5.2. Single-Template Results (Connected Scan)	5-3
5.3. Single-Template Results (Isolated Scan)	5-5
5.4. Single-Template Results (*Isolated Scan*)	5-6
5.5. Merged-Template Results (Two Speakers)	5-7
5.6. Merged-Template Results (Three Speakers)	5-8
5.7. Merged-Template Results (Four Speakers)	5-9
5.8. Merged-Template Results (Five Speakers)	5-10
5.9. Additional Selected Utterances	5-12
5.10. Additional Merged-Template Results (Four Speakers)	5-12
5.11. Additional Merged-Template Results (Five Speakers)	5-13

Abstract

A system is developed to achieve speaker-independent isolated speech recognition. The system uses LPC Spectrum, Formants, and Frication Frequency as a feature set. Dynamic programming is applied for distance calculations. The fundamental design concept is to create a universal template for multiple speakers. A new algorithm, which combines the vocabularies of several speakers to produce one optimal template, is incorporated into the system. An advanced speech analysis tool called SPIRE provides the computational functions required to extract appropriate features. Seventy words, from the list of F-16 cockpit commands, are selected as a vocabulary of the system. The use of a merged template based on three of the feature sets achieves an accuracy of 99 percent. The system is implemented in list programming on Symbolics 3600 series computer.

F-16 SPEAKER-INDEPENDENT SPEECH RECOGNITION SYSTEM USING COCKPIT COMMANDS (70 WORDS)

I. Introduction

Speech recognition by machine is a topic that has lured and fascinated engineers and speech scientists for forty years. For many, the ability to converse freely with a machine is the ultimate challenge to our understanding of the production and perception processes involved in human speech communication (5:26). This capability is also becoming a necessity. With the recent surge in the use of computers for information processing and the corresponding increases in the workload of human users, there is a growing need to incorporate speech as an added mode of human/machine communication (15:192).

During the last decade, a significant advance in speech recognition technology has been achieved by the speech researchers in the world. As a result, speech recognition systems with limited capabilities are now available both commercially and militarily. These systems are usually able to work with only a small number of acoustically distinct words spoken by a known speaker, and their performance varies widely as a function of the particular system, vocabulary, speaker, and operating environment (7:1404). Also, these systems utilize little or no speech-specific knowledge, but rely instead primarily on general-purpose pattern-recognition algorithms. While such techniques are adequate for a small class of well-constrained speech recognition problems, their extendability to speaker-independence, large vocabularies, and/or continuous speech is highly questionable. In fact, even for the applications that

these systems are designed to serve, their performance typically falls far short of human performance (18:300).

Background

Speech recognition is the technology by which sounds (either words or sentences) uttered by humans are understood by a machine (15:201). This technology can be applied as a substitute for manual control of switches and keyboards in many complicated operating environments (11:268).

The principle of speech recognition is explained as follows. Analysis of the words to be recognized is the first step, and a reference template (dictionary data) is created from the analysis results. During recognition, words spoken by an individual are analyzed and compared with the contents of the reference template by pattern matching. When a match is made between a spoken word and a word found on the reference template, that word is considered to be the correctly recognized word (12).

Speech recognition may be broadly classified according to the method used to create the reference template: either speaker-dependent or speaker-independent (5:28). In speaker-independent speech recognition, the speech characteristics of many people are used to create the reference template (or templates) by a method that offsets the differences among individuals. This method allows the speech recognition system to be used by more than just a select group of individuals. There is, however, an enormous amount of work involved in creating a reliable reference template (13).

However, in speaker-dependent speech recognition, the reference template is created using the vocal sounds made by the actual user of the system. As far as the user is concerned, although creation of the dictionary is quite troublesome, there are no restrictions whatever on word selection for recognition. Moreover, this method has the advantage that the vocabulary of words for recognition can be altered to increase recognition accuracy. For this reason, speaker-dependent speech recognition units are generally superior in terms of recognition accuracy (3).

In addition, methods for recognition/analysis of monosyllables and individual words may be further divided into isolated spoken forms and continuous spoken forms. The majority of speech recognition systems that are currently in actual use employ the isolated spoken forms. Since it becomes more difficult to obtain high recognition accuracy as the number of words is increased, the scale and cost of the system rise sharply as the number of vocabulary in reference template increases (2:94).

Problem Statement

The primary purpose of this thesis is to design and implement a speaker-independent speech recognition system utilizing F-16 cockpit commands (70 words). The goal is to produce a system that can do isolated word recognition on a SPIRE based system. SPIRE stands for Speech and Phonetics Interactive Research Environment. It is a software program that allows an user to interactively examine and process speech signals. The system will be implemented on the Lisp machine (Symbolics 3600).

Scope

The system implemented in this research is designed to recognize speaker-independent isolated words. The list of vocabulary words used in the system is shown in figure 4.3. Based on the results of thesis conducted by Dawson (3), only one feature set (a combination of Linear Predictive Coding, Formants, and Frication Frequency) is selected to utilize in this study. The dynamic programming algorithm applied in the system is identical to the *One-Stage Dynamic Programming Algorithm* proposed by Ney (11). A new technique, which combines vocabularies of two or more speakers to produce an optimal reference template, is applied in the final stage of the research. Throughout the experiment, SPIRE is used as a library of functions called by the main LISP program for performance of creating the utterance files.

Assumptions

In the process of analog-to-digital conversion, there exist some acoustic noises that are merged with the actual speech waveforms. However, the noises are small enough to be neglected during the entire speech recognition procedure.

SPIRE is used to cut and normalize the individual waveform out of the digitized and recorded continuous speech waveforms. The purpose of this is to create the utterances of each word in the vocabulary. However, in this step, the boundaries are drawn manually. The establishment of a boundary is consistently accurate to include only the actual speech components of the waveforms. In other words, neither noise nor silence has been included as a part of an individual word representation.

General Approach

The general approach for system design and implementation is as follows. First, Dawson's program (3) was used to expand the vocabulary from ten digits to seventy words. This step involved modifications of the program to accommodate all seventy words, in comparing and displaying, the reference template and the testing utterance.

With the modified algorithm, a test of speaker-independence was conducted. Five different speakers were utilized in the test. Due to the limitation of disk space and computational time, only fifty words (ten words from each of five speakers) were selected.

Next, an algorithm was designed to use vocabularies of two or more speakers and produce one reliable reference template. The technique was a one-time calculation and preprocessing method. And it may be a solution to the speaker-independent speech recognition system.

Finally, the new system was tested with five different speakers (ten words from each speaker). The test started with the template utilizing the vocabularies of two speakers only. And the testing was repeated for three, four, and five speakers.

Materials and Equipment

The following materials and equipments were used:

- Digital Sound Corporation (DSC) A/D Converter.
- Symbolics 3600 Lisp Machine;
 1. One Mega-Word of RAM.
 2. Floating Point Accelerator.
 3. Lisp (List Programming).
 4. SPIRE (Version 17.5).
- Noise Reducing Microphone.
- Laser Printer.

Sequence of Presentation

Chapter two gives brief summaries of the previous speech recognition research conducted by AFIT thesis students.

Chapter three presents the technical discussion needed to understand the research being conducted in this thesis. In particular, the chapter describes SPIRE as well as the Symbolics 3600 Series Lisp Machine and List Programming.

Chapter four describes the major processing functions and design concept of the system developed in this research. The concept of the new algorithm will be explained in detail.

Chapter five discusses testing procedures and results for both the modified system and the newly developed system.

Chapter six provides conclusions and recommendations, and appendices show program listing and sample results.

II. Background of the AFIT Speech Recognition System

The development of speech recognition system has been an important area of research at the Air Force Institute of Technology for several years. In this chapter, an overview of previous speech recognition efforts is presented. From 1981 to 1987, research has been conducted to improve the recognition accuracy, increase the size of vocabulary, and/or develop the speaker-independent continuous speech recognizer. In order to familiarize the reader with the concepts and the techniques that have been applied to these researches, a brief description of individual study is given. In addition, important concepts and techniques are explained as the addendums to those descriptions.

Computer Analysis and Recognition of Phoneme Sounds

In 1981, Seelandt investigated the characteristics of phonemes uttered in connected speech (16:1-2). The results, then, were used to obtain a set of prototype phonemes which could be used in a pattern matching recognition scheme. The prototype phonemes were tested and refined to establish an optimal set of prototypes (16:5). Also, testing and refinement of a recognition routine were accomplished. In pattern matching, the routine used the results of distance measurement calculation to choose possible phoneme matches for each time period (16:18-20). The recognition routine also generated information on how good a choice it made, so the accuracy of both the recognition method and prototype phonemes could be measured (16:15). Finally, Seelandt developed an algorithm to apply the optimal set of prototype as a key feature for the speech recognition (16:7).

Phoneme Analysis Speech recognition method, used in Seelandt's thesis, is aimed at the creation of an optimal set of prototype phoneme templates. Since the set of prototypes would be the basis for speech recognition, the development of the

prototype phoneme templates was the critical factor in his speech recognition system (16:17). Therefore, an understanding of phonemes and how they appear in the actual speech is necessary.

A most extensive study of phonemes was conducted by Potter, Kopp, and Green in 1947. They defined the description of phonemes that make up English speech. Their phoneme set consists of 47 phonemes including both voiced and unvoiced fricative and stop sounds, as well as vowels and diphthongs (16:20-29). Diphthongs are the combination sounds. As can be noticed in chapter four of this thesis, the fricative sound plays an important role as one of features applied to digitized speech waveforms of both template and testing utterance.

Most of the time, the actual phonemes in normal speech are quite variable because they change as a function of their usage within the word or the sentence. It takes a finite time to actually vary the shape of the vocal tract and create a different sound (13:29). Also, the actual sound produced by the vocal tract depends on the past position of vocal tract. This phoneme-to-phoneme transition causes many of the problems in the connected speech recognition. Any recognition method which ignores this transition affect will not produce accurate results (16:18). The spectrographic pattern of words, however, may be identified by careful visual inspection (by skilled analyzers) as long as they occur in intelligible speech (12:23-40).

Based on the above principles of phoneme, Seelandt integrated a system called the *Speech Sound Analysis Machine* (16:37). The system is an interactive software controlled tool which is capable of displaying speech spectrograms while concurrently generating the associated speech waveforms. It shows exactly what distinct sounds are uttered in selected speech segment while allowing an operator to listen that segment of speech (16: 39- 45).

After performing analyses, Seelandt produced a recognition algorithm based on the results. The results indicate that a speech utterance can be separated into or defined by a set of distinguishable sound units (16:92-95). Also, there are transition

Table 2.1. Speech Recognition Procedure (16:94)

1. Record and digitize a speech utterance.
2. Run TRYDIST6 to obtain distance measurement results between the speech and a set of templates.
3. Using the data files created in step 2, run a LISTER routine to manipulate the distance results in an attempt to recognize the speech utterance.
4. The output files created by step 3 can be used to: quickly search for identification and location of phonemes, scan scale and range factor data along with the 5 best matches in an effort to determine template accuracy, and synthesize speech.

sounds between two distinguishable sound units in both normal and fast speech. The recognition algorithm is basically a software tool which has been developed and refined to incorporate the above analysis results (16:98-99). Table 2.1 shows an example of how this tool may be used.

Isolated Word Recognition Using Fuzzy Set Theory

Following Seelandt's research, Montgomery developed an algorithm to recognize isolated words (10:4). As a solution to high error rate of the acoustic processor, the research relies on the consistency of phoneme sequences and the errors that typically occur when a word is given to be recognized (10:3-5). Montgomery generated error statistics and phoneme representations for each word in the vocabulary using a set of training speech files. Then he implemented the top five phoneme choices

with the information indicating the accuracy of each choice for each time segment of speech. Those choices were provided by the acoustic processor (10:4-7). Montgomery applied the *Fuzzy Set Theory* to combine the phoneme choices with the error information obtained from the training file in determining the word spoken (10:25). The algorithm produced a word score indicating the plausibility of the word being spoken regardless of the number of errors or types of errors exist in the acoustic output. A score is generated for each word in the vocabulary and the word with the highest score is selected as the spoken word. Montgomery believed that his algorithm could readily be adapted for real time speech recognition (10:45-50). However, the algorithm should be adjusted, as the number of operations required to make a decision increases when the larger size of vocabulary is implemented.

Fuzzy Set Theory Since Montgomery utilized the fuzzy set theory in the process of determining the spoken words, a discussion of the theory is appropriate. The theory provides a concept for sets of information in which the transition from membership to non-membership is not clearly defined (10:20-21). For example, a set of people who are strong is not clearly defined and the transition can not be clearly established. In order to include this fuzziness, the members of fuzzy set are assigned a grade of membership which is some value between zero and one (10:21). A fuzzy set, S , would then be of the form:

$$S = \{x_1/u(x_1), x_2/u(x_2), \dots, x_i/u(x_i), \dots, x_n/u(x_n)\} \quad (2.1)$$

where x_i is an element of set S with a grade of membership of $u(x_i)$ in the set. Continuing the example of the set consisting of strong people, one possible fuzzy set may be

$$S(\text{strong}) = \{10/1.0, 25/0.9, 35/0.8, 50/0.5, 75/0.01\} \quad (2.2)$$

where the strengths, or elements, of the set are 10, 25, 35, 50, and 75 and their respective grades of membership are 1.0, 0.9, 0.8, 0.5, and 0.01.

The grades of membership can be assigned either subjectively, as in those shown above, or may be assigned by using a mathematical equation (10:21). An example of an intuitive equation that could be used to determine the grades of membership of the elements in the set of strong people is shown below.

$$u(x_i) = \begin{cases} 1 & \text{For } x_i \leq 25, \\ (1 + ((x_i - 25)/5)^2)^{-1} & \text{For } x_i > 25 \end{cases} \quad (2.3)$$

where $u(x_i)$ is the grade of membership of the element x_i in the set $S(\text{strong})$.

Algebraic Properties Fuzzy set theory is not an informal mathematical tool. Since the introduction of this theory, numerous papers have been written in attempts to provide a more formal theory. Some of the formal algebraic properties that have been suggested include (10:21-23):

Equality: Two fuzzy sets, A and B , are equal if, and only if, $u_a(x) = u_b(x)$ for all x , where x is an element of the set of all elements.

Containment: A fuzzy set A is contained in, or is a subset of a fuzzy set B , written $A < B$ if, and only if, $u_a(x) \leq u_b(x)$ for all x .

Complementation: A' is the complement of the fuzzy set A if, and only if, $u'_a(x) = 1 - u_a(x)$, for all x .

Intersection: Intersection of the fuzzy sets A and B , denoted by $A \cap B$, is given by $u_{a \cap b}(x) = \text{Min}(u_a(x), u_b(x))$ for all x and is defined as the largest fuzzy set contained in both A and B .

Union: Union of the fuzzy sets A and B , denoted $A \cup B$, is given by $u_{a \cup b}(x) = \text{Max}(u_a(x), u_b(x))$ for all x and is defined as the smallest fuzzy set containing both A and B .

Although formalization is generally useful, strict adherence to the formal theory that some have proposed may significantly diminish its advantages and possible applications (10:24-28). For instance, the operators "Min" and "Max" are used

extensively for the sake of formality. Although these operators are definitely of value, the choice of an operator is always a matter of context, and mainly depends upon the real world situation. In other words, all mathematical properties, regarding the class of fuzzy set theoretic operators, should be interpreted at an intuitive level (10:23). Therefore, any operator which appears reasonable, such as the **Average** or **Product** operators defined below, should also be considered when developing an algorithm employing fuzzy set theory (10:23).

Average: The average of two fuzzy sets A and B , to produce fuzzy set C , is given by $u_c(x) = (u_a(x) + u_b(x))/2$ for all x .

Product: The product of two fuzzy sets A and B , to produce fuzzy set C , is given by $u_c(x) = u_a(x) \cdot u_b(x)$ for all x .

Limited Continuous Speech Recognition by Phoneme Analysis

In 1983, Hussain developed a limited continuous speech recognition system based on phoneme analysis (6:1-3). Sixteen bandpass filters were used to obtain the frequency components of input speech. The input speech was broken into packets of forty milliseconds width. The packets then were compared with phonemes in the template file by differencing the frequency magnitudes (6:4). Finally, the resulted phoneme string representation of the input speech was compressed and compared with strings in the library file for discrete word recognition. For continuous speech recognition, the phoneme string was analyzed, one phoneme at a time, to construct word sequences (6:5). The word string which best matched the input phoneme string was recognized as the word sequence. The system achieved an accuracy of ninety-four percent for discrete word recognition and eighty percent for continuous speech recognition (6:6-8). The vocabulary used was ten digits (zero through nine) and a point.

Approaches and Techniques The outline of Hussain's phoneme based speech recognition is as follows. The first step is to divide the input speech into a sequence of sounds. Each unit of the sound sequence is compared with a reference set of unique sounds. The unique sounds are the phonemes in the library. Then, a sequence of phonemes can be established which represents the original input speech (6:4). This string is called *Phoneme Representation of Input Speech*. Finally, the string is processed for constructing the word or words spoken. The construction process includes compression of the string and comparison of the string with the phonemes in the template or dictionary file (6:5).

In order to achieve successful construction, Hussain needed to set a reliable processing environment. The processing environment is a critical area in any speech recognition process. Hussain puts a heavy emphasis on his processing environment for the purpose of producing more accurate speech recognition system (6:40-41). The following paragraphs briefly describe the environment utilized in development of Hussain's system.

Automatic Gain Control (AGC) circuit is incorporated in the environment for several reasons. After preamplification, the input speech is screened with a preemphasis filter. The filter has a gain of 6 dB/octave above 500 Hz. Then, the input speech passes through the AGC circuit, which has 60 dB dynamic range. Three reasons for using AGC circuit can be identified. First, the spectrum analyzer requires a certain minimum input level for proper operation. The next reason is that the energy threshold applied depends on the AGC circuit. Thirdly, the AGC reduces jittering affects of the input speech.

For an analysis of the input speech, Hussain used a bank of bandpass filters (sixteen of them) instead of applying Fast Fourier Transform (FFT). Sixteen bandpass filter outputs of the input speech were produced by utilizing the ASA-16 spectrum analyzer chip in the hardware (6:8). There are two advantages of using bandpass filters. First one is that the filters eliminate the inherent "noise" produced when FFT

analysis is used for preprocessing. Secondly, the sampling may be accomplished at a lower rate (6:40). A typical sampling rate for FFT is 8 KHz. However, it is only 400 Hz for bandpass filter approach.

Using an analog-to-digital speech digitizer, the outputs of the above filters are digitized at sampling frequency of 400 Hz. This gives each of sixteen filter outputs a sampling rate of 25 Hz (6:43). Next the outputs pass through a lowpass filter which has cutoff frequency of 25 Hz. The cutoff frequency is selected to be 25 Hz because the energy variation bandwidth of human speech does not exceed 25 Hz. Therefore, the output of the low pass filter would have same time length as one packet of the input speech which is forty milliseconds (6:46-52).

Initially, Hussain used two packets of the input speech for the phoneme representation. However, he realized that single output of sixteen channels was sufficient to represent a phoneme sound (6:6). Hence the individual phoneme sound was forty milliseconds long which consisted of sixteen dimensional vectors. These vectors are the outputs of sixteen band pass filters (6:55).

In addition, to eliminate the background noise and the direct current offset errors, twenty millivolts of the threshold level is subtracted from the input signal (6:65). The phonemes or sixteen dimensional vectors are also individually normalized to unit energy. The energy normalization is needed for the phoneme recognition and comparison routine (6:49-50). The sequence of energy normalized vectors represent the input speech.

After normalization, the vectors of input speech are compressed for two reasons. Notice the input speech is in the form of phoneme sequence. The sequence of phonemes is compressed to prevent the variations which inevitably occur when a word is spoken several times (6:52). Also, it is compressed to overcome the "noise" created due to natural variations in the speed of speech (6:53). This compressed phoneme representation of the input speech is now ready for comparison.

Finally, Hussain created a set of unique phonemes for the reference template and compared them with each vector of the compressed input speech (6:55). In the process of comparison, the distance differences were calculated between the vectors of input speech and the reference template. For the calculation, he initially used an approach of difference raised to power of two. However, the approach of difference raised to power of four resulted in better accuracy (6:6-7).

Implementation of A Real-Time, Interactive, Continuous Speech Recognition System

In 1984, Dixon (4) designed and implemented a speech recognition system to recognize continuous speech in a real time environment (after training). She incorporated several techniques in characterizing phonemes as vectors in space. The words were characterized by phoneme representations which subsequently were used in word recognition (4:5-6). The distance rules were utilized in converting the words to phoneme representation. Dixon believes that her approach to speech recognition offers several possibilities for future investigation such as varying the Minkowski distance computation rule and applying clustering technique (4:8-10). Her algorithm is modularized on a hierarchical basis and is user friendly. Modularity provides an easy modification to system components. User friendly means that the algorithm can be easily used by someone who may not be a computer expert (4:35-40). Dixon basically combined several techniques, developed earlier at AFIT, with additional modifications to produce a viable speech recognition system (4:4-5). The techniques used are explained in the following paragraphs.

Phoneme Representation One of the techniques Dixon used is phoneme representation. Seelandt produced a set of seventy phonemes by combining several time slices or vectors of digitized speech. And Hussain developed single-vector sixteen-dimensional phonemes (4:4). However, the method for phoneme generation in Dixon's system compares and averages the sixteen-dimensional vectors to produce a set of less than seventy phonemes. The comparison and averaging continue until

Table 2.2. Minkowski Distances (4:21)

$$\text{Minkowski 1 } D_1 = \sum_{i=1}^M |x_{i,m} - x_{i,n}|$$

$$\text{Minkowski 2 } D_2 = [\sum_{i=1}^M (x_{i,m} - x_{i,n})^2]^{1/2}$$

$$\text{Minkowski N } D_N = [\sum_{i=1}^M (x_{i,m} - x_{i,n})^N]^{1/N}$$

where N = Minkowski Distance

M = Vector Length

i = Vector Element

m = Vector m

n = Vector n

the phoneme set is established (4:19).

Distance Rules Another technique Dixon applied is an adaptable distance rule. Many pattern recognition algorithm incorporates metrics based upon the Minkowski distance. Table 2.2 shows some of different ways. However, Dixon experimented with several Minkowski exponents and usually used Minkowski 4 in her algorithm because it emphasized the effect of any large discrepancies between compared vectors (4:20).

Spatial Comparison In order to investigate any possible clustering affects, Dixon performed some spatial comparisons. Since some vectors in the n-dimension decision space lie closer together than others, these vectors can be used to define regions of space or clusters (4:28-30). Dixon's algorithm redefines the nearest vectors by averaging and weighting (4:22). Therefore, each vector has an initial weight of one and those vectors, which are closest to one another by Minkowski distance, are averaged together by the equation,

$$X = (X * weight(m)) + (X * weight(n)) / weight(m) + weight(n) \quad (2.4)$$

The highest numbered vector is deleted and a new weight is assigned to the new vector by

$$weight(m) = weight(m) + weight(n) \quad (2.5)$$

The resulted set of vectors represents clusters of similar sounds.

Speaker-Independent Word Recognition Using Multiple features, Decision Mechanisms, and Template Sets

In 1986, Brusuelas developed a speaker-independent word recognition system (1:2-4). The system incorporates multiple decision mechanisms, features, and template sets to recognize whole word, isolated speech inputs. The category of features utilized in the multiple feature sets includes Linear Predictive Coding (LPC) coefficients, LPC gain term, time, zero crossing rate, wide-band spectrogram, and formant tracks (1:5-8). He derived a total of fifty-five processed features from the above category. The computation for raw extraction is provided by an advanced speech analysis tool called *Speech and Phonetics Interactive Research Environment (SPIRE)*. The results of experiment indicate that the system can successfully recognize whole word inputs from a range of independent speakers (1:17-20). Brusuelas conducted the experiment with limited vocabulary inputs and predefined template set. The system is designed in List Programming (LISP) on Symbolics 3600 series computer (1:22-24).

Feature Categories Brusuelas' feature selection is based on two factors. First, he found that the spectrogram is an aid to reading other features. In other words, when other features are aligned with the spectrogram, they can be read more clearly than when they are not aligned (1:22-23). Therefore, he decided to include the spectrogram as one of the feature set. The second factor involves development efficiency. He wanted to choose the features that could be readily implemented with current technology (1:18). Furthermore, only features provided by SPIRE were to be considered. As a result, Brusuelas selected six feature categories, shown in figure 2.1. The descriptions of the features are presented in the following paragraphs.

(1) **Wide-Band Spectrogram.** As shown in figure 2.1, the vertical striation in the spectrogram is indicative of the pitch period. Voicing is depicted by dark horizontal bands (formants). The spectrogram is computed with Fourier transform using the Hamming filter which has window width of 3.3 ms (1:8).

(2) **Zero Crossing Rate.** This feature counts the number of times the waveform passes the region centered around zero.

(3) **LPC Gain Term** and

(4) **LPC Coefficients.** LPC is an useful feature because it produces extremely accurate estimates of the speech parameters. Also, its relative speed of computation is high (1:23). The gain term describes the modulation pattern of the waveform energy while the coefficients describe the modulation patterns of particular frequency bands (1:24).

(5) **Formants.** Formant patterns are one of the primary visual references used in reading spectrograms.

(6) **Time.** Time is the only feature category that is not provided by SPIRE. A relative measure of time is performed by extracting the length of an array which stores the digitized utterance (1:24).

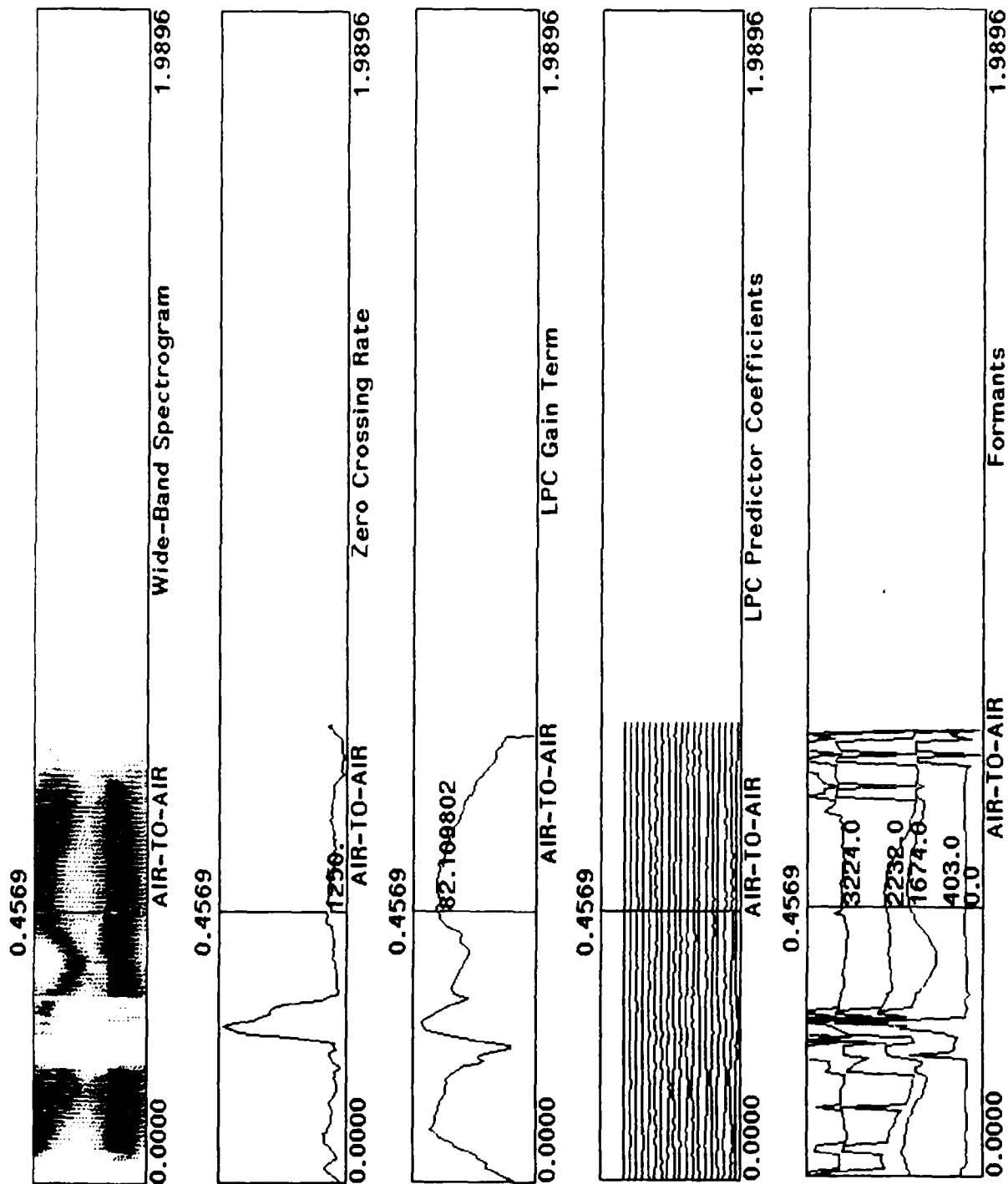


Figure 2.1. Feature Sets Used in Brusuelas' Thesis (1:22-28)

Processed Features The results of all feature computations are one or two dimensional arrays except for time. For example, the zero crossing rate is a one dimensional array where the length of the vector corresponds to time length of the utterance (1:29). Also, the formants computation is a two dimensional array where the second dimension corresponds to time length and the first corresponds to a discrete formant frequency component. The following paragraphs describe the processing techniques which Brusuelas utilized to efficiently run the recognition process (1:29-32).

(1) **Time Compression.** Features are linearly time compressed to a length of twenty. Twenty is an arbitrary length. Brusuelas wanted to compress the data without severely distorting the dynamic characteristics of the feature sets (1:30). Time compression is needed because it saves subsequent computation time and storage requirements; it also smoothes the data. In addition, since dynamic time warp is not applied, the compression is needed to compensate for unequal length of utterances (1:31). In the process of time compression, twenty windows are overlaid on the input data along the time dimension. The windows are adjacent, non-overlapping, and equal length for any given input (1:32). Data in the windows are averaged to produce the time compressed values. The resulting vector or array is always a length of twenty since twenty windows are used (1:30-31).

(2) **Normalization.** The input arrays of features are normalized by dividing the individual array elements by the total array energy (1:30). There are three different ways to normalize the array. The array can be normalized in its entirety, by rows only, or by columns (1:31).

(3) **Median Filter.** Median filter is utilized in the system to bridge formant discontinuities because formant tracking is sometimes erratic (1:31). The filter changes the value of a datum point to the median value of any particular window. The filter is also applied to the LPC coefficients and wide-band spectrum to investigate its effectiveness on other features (1:36).

(4) **Frequency Compression.** In order to reduce the computational requirements during matching and smoothing the data, linear frequency compression is necessary for the wide-band spectrogram feature (1:32). The spectrogram arrays are 250 by 20 where 250 is the number of frequency components and 20 is the number of compressed time slices (1:41). However, after the compression, the resulting array is 16 by 20.

SPIRE Based Continuous Speech Recognition System

In 1987, Dawson developed a SPIRE based continuous speech recognition system (3:1-8). The system incorporates multiple features and dynamic programming to recognize continuous inputs of spoken digits (zero through nine). The features used in his research effort are wide band spectrogram, narrow band spectrogram, LPC spectrum, frication frequency, and formant tracks (3:1-4). SPIRE is utilized to perform the computational functions needed to extract the raw features. The system is implemented in LISP on a Symbolics 3600 series LISP machine (3:2- 10).

Dynamic Time Warping A technique which nonlinearly time aligns the speech patterns is called dynamic time warping. Dawson used the technique in his system to compensate the nonlinear time variations common in speech. However, the algorithm of dynamic time warping was originally presented by Vintsyuk and later modified by Herman Ney (11:263-265).

Distance Arrays The distance arrays are two dimensional arrays, M by N , where M is proportional to the length of the template and N is proportional to the length of the test pattern (3:3-7). M and N are represented by the sequences of vectors. Each vector of these arrays represents the features of both the template and the utterance extracted at each moment m and n respectively. The value of subscripts (m,n) in the distance array then represents the vector distance between the template at moment m and the utterance at moment n (3:3-9). In the speech

recognition process, distance arrays are considered as the key technique for matching. For isolated speech, a measure of matching is taken by tracing the path from point $(0,0)$ to point (M,N) of the distance array which results in the minimum accumulated distance of all the possible points in that path (3:3-10).

An example of distance array using a hypothetical feature set is shown in table 2.3a. At any particular moment, the speech is represented by a three dimensional vector indicating the energy in each of the three frequency bands. In order to recognize an isolated word, the distance array between the test word and each word in the template would be calculated and the word with a minimum accumulated distance will be chosen (3:3-9). The distance rule used here is Minkowski 1. However, to find a minimum path through the connected speech distance array, an accumulated distance array has to be formulated as shown in table 2.3b. In this array, the vector value of each point represents an accumulated distance which is the sum of the local distance of that point and the minimum of the accumulated distance of all possible preceding points (3:3-7). Notice that certain constraints govern the route of the traced path. In other words, the path must continue forward in time for both template and the test pattern. Therefore, the path can not go left or down and points may not be skipped or omitted (3:3-9).

One-Stage Algorithm for Connected Speech Dawson applied Ney's (11) algorithm called *One-Stage Algorithm for Connected Speech* to the main program of his thesis. Figure 2.2 shows how a composite distance array of grid points (i, j, k) is computed. Notice individual time slices of test pattern are referenced by index j and time slices of each word k in template are referenced by index i . A minimum accumulated distance $D(i, j, k)$ is defined for each grid point (i, j, k) in order to find the minimum path through the array (3:3-9). Each point $D(i, j, k)$ is the minimum sum of local distances $d(i, j, k)$ along some path to grid point (i, j, k) . For any grid point (i, j, k) , $D(i, j, k)$ is found by selecting the predecessor with the minimum accumulated distance and adding that accumulated distance to the local distance

Table 2.3. Distance Arrays (3:3-8)

	(0,2,5)	1	13	7	1
Template	(0,2,5)	1	13	7	1
Pattern	(0,0,0)	8	10	2	8
(Six)	(5,4,0)	11	1	9	13
	(0,2,5)	1	13	7	1
	(0,2,5)	1	13	7	1

(0,3,5) (5,5,0) (1,0,1) (0,2,6)

Test Pattern (Six)

2.3a Hypothetical Distance Array

	(0,2,5)	25	37	19	7
Template	(0,2,5)	24	26	12	6
Pattern	(2,0,0)	23	13	5	13
(Six)	(5,4,0)	13	3	12	25
	(0,2,5)	2	14	21	22
	(0,2,5)	1	14	21	22

(0,3,5) (5,5,0) (0,0,1) (0,2,6)

Test Pattern (Six)

2.3b Hypothetical Accumulated Distance Array

$d(i, j, k)$. The transition rules consist of within-template rules and between-template rules (3:3-10). Thus for the template interior, $j > 1$, the recursion rule is,

$$D(i, j, k) = d(i, j, k) + \min[D(i-1, j, k), D(i-1, j-1, k), D(i, j-1, k)] \quad (2.6)$$

At template boundary with $j = 1$, the recursion rule is,

$$D(i, j, k) = d(i, j, k) + \min[D(i-1, J(k^*), k^*)] \quad (2.7)$$

where $k^* = 1, \dots, K$. By keeping track of where the path crosses template boundaries, the problem of boundary detection in the test pattern is handled automatically (3:3-11). A flow chart for the One-Stage Dynamic Time Warping Algorithm for connected speech is shown in figure 2.3.

Time Distortion Penalties Dawson's algorithm applies time distortion penalties using slope dependent weights. For three directions, horizontal, diagonal, and vertical, the local distance is multiplied by the weights $(1 + a)$, 1, and b respectively, prior to evaluating the dynamic programming recursion (3:3-10):

$$D(i, j, k) = \min[(1 + a) \cdot d(i, j, k) + D(i-1, j, k), \\ d(i, j, k) + D(i-1, j, k), \quad b \cdot d(i, j-1, k) + D(i, j-1, k)] \quad (2.8)$$

Summary of Steps The summary of Dawson's dynamic time warp algorithm is shown in figure 2.4. Notice that the unknown sequence is recovered in step 3 above by tracing back the decision taken by the *minimum* operator at each grid point (3:3-11).



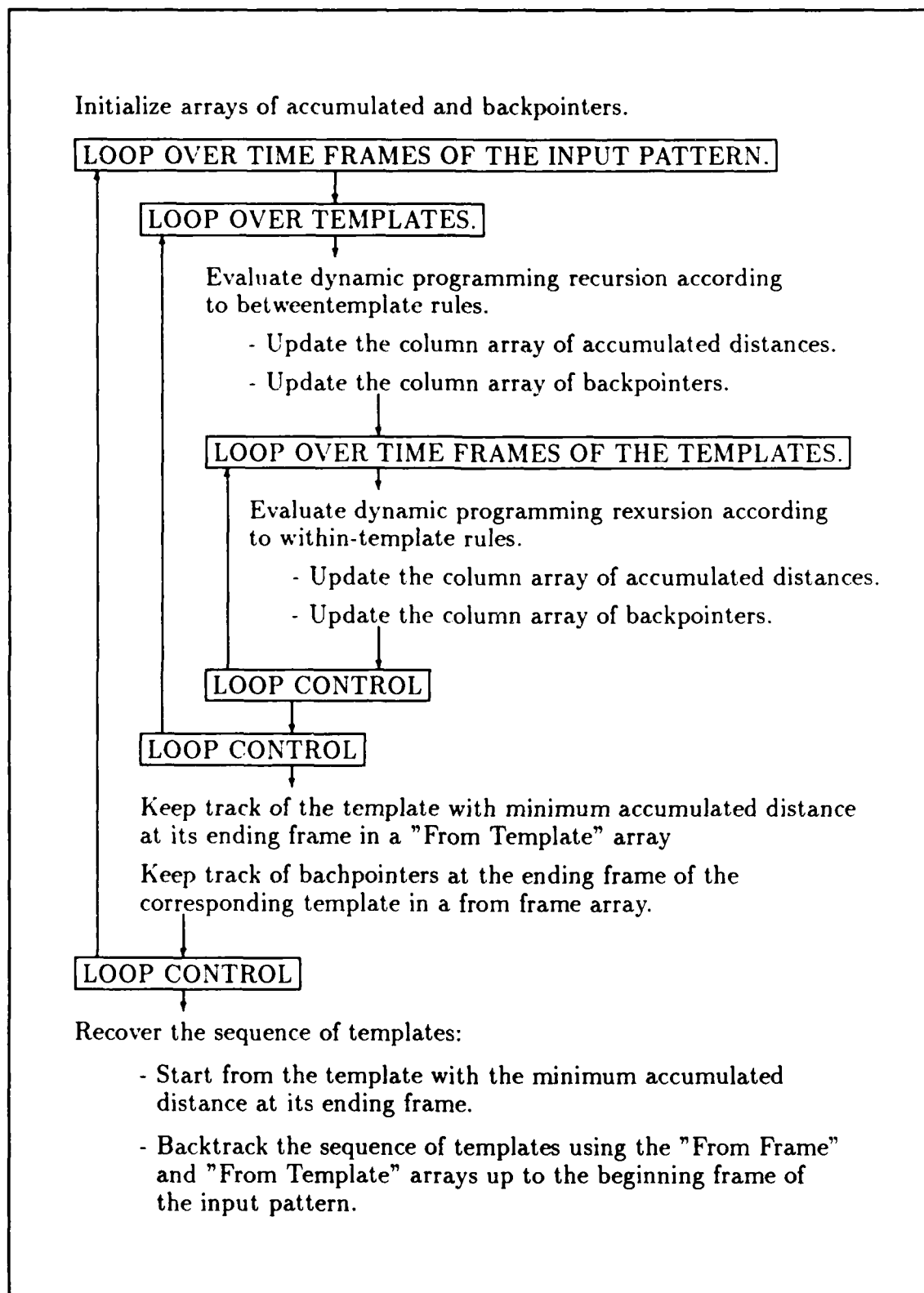


Figure 2.3. Schematic Diagram (3:3-16)

Step 1. Initialize $D(1, j, k) = \sum_{n=1}^j d(1, n, k)$.

Step 2. **(a)** For $i = 2, \dots, N$, **do steps 2b-2e.**

(b) For $k = 1, \dots, K$, **do steps 2c-2e.**

$$(c) \ D(i, 1, k) = d(i, 1, k) + \min[D(i - 1, j(k^*), k^*)].$$

(d) For $j = 2, \dots, J(k)$, do step 2e.

$$\begin{aligned} \text{(e)} \quad D(i, j, k) = \min[& (1 + a) * d(i, j, k) + D(i - 1, j, k), \\ & d(i, j, k) + D(i - 1, j - 1, k), \\ & b * d(i, j - 1, k) + D(i, j - 1, k)]. \end{aligned}$$

Step 3. Trace back the path from the grid point at a template ending frame with the minimum total distance using array $D(i, j, k)$ of accumulated distances.

Figure 2.4. Dynamic Time Warp Algorithm (3:3-11)

III. Processing Environment

This chapter introduces the software and hardware components utilized to develop the speech recognition system. First, List Programming (Lisp) is briefly explained. Next, SPIRE is described in a detail. Then, the last section presents the hardware configuration as well as other optional equipments.

Lisp

Lisp is a high level programming which takes its name from *List Programming*. It is one of the oldest active programming languages and widely used in the field of artificial intelligence (14). Lisp is an extremely powerful language for handling large amounts of data common in artificial applications. In fact, special purpose computers called *Lisp Machines* are designed at the circuit level to run only Lisp. Together, these provide a powerful computing environment with a large virtual address space. Therefore, Lisp is very popular for speech and signal processing applications (19:5).

There are many dialects of Lisp. One dialect called *COMMON Lisp* is becoming a standard. Most Lisp machines available today have *Common Lisp* as a standard language. However, a different dialect called *Zeta Lisp* is used in this thesis.

SPIRE

SPIRE stands for Speech and Phonetics Interactive Research Environment. SPIRE is available by license through the MIT patent office. It is a software program which allows the user to interactively examine and process speech signals.

It is designed to support the need of simple computer users as well as skilled programmers (17:1). In the process of development, the Speech Research Group at MIT made provisions to be easily customized and modified by the users. Also, its

user interface is simple, and it can be easily learned by a beginner. However, the real power of SPIRE comes from its capability to display the information necessary for speech analysis and speech recognition (8:4-5).

Furthermore, its graphical display capability provides a unique feature to users. A speech signal can be displayed in a variety of different ways. Figure 3.1 shows a list of different ways that SPIRE can process the signal and display on the screen. The signal can also be resized, rescaled, recomputed, repositioned, and overlaid (17:2-10). The figure 3.2 demonstrates a small sample of these capabilities for the utterance, "AIR-TO-SURFACE MISSILE."

The top display of the figure shows the orthographic transcription of the utterance. Below that is the original waveform. This waveform is the shape of the particular speech which does not contain any data from the features available in the SPIRE package (1:7). The next display is an overlaid one (Narrow-Band Spectrogram and Formants) which makes it easier to track similarities among various representations of the data. The last two displays demonstrate a unique capability of SPIRE, an ability to synchronize displays. Notice that there is a *cursor* located at 0.9467 seconds of the Wide-Band Spectrogram as well as the original waveform and Narrow-Band Spectrogram. The last display reflects the Wide-Band Spectral Slice at that cursor position. The corresponding Narrow-Band Spectral Slice may also be displayed (1:8).

Interfacing SPIRE from Lisp For each SPIRE display, computations are required to obtain the necessary data. And the process of computations is available through Lisp as simple function calls. The primary functions used to make SPIRE perform computations on an utterance are shown in figure 3.3. However, the three functions of the table can be combined into a single Lisp expression as shown below (1:13).

(SETQ

Energy, Total
Energy, 0 to 5000 Hz
Energy, 120 to 440 Hz
Energy, 3400 to 5000 Hz
Energy, 640 to 2800 Hz
Formants, All Four
Formant, First
Formant, Second
Formant, Third
Formant, Fourth
Frication Frequency
LPC Center of Gravity
LPC Gain Term
LPC Predictor Coefficients
LPC Spectrum Slice
Narrow-Band Spectrogram
Narrow-Band Spectral Slice
Narrow-Band Spectrum Slice
Original Waveform
Orthographic Transcription
Phonetic Transcription
Pitch Frequency
Waveform Envelope
Wide-Band Spectrogram
Wide-Band Spectral Slice
Wide-Band Spectrum Slice
Zero Crossing Rate

Figure 3.1. List of SPIRE Displays (1:12)

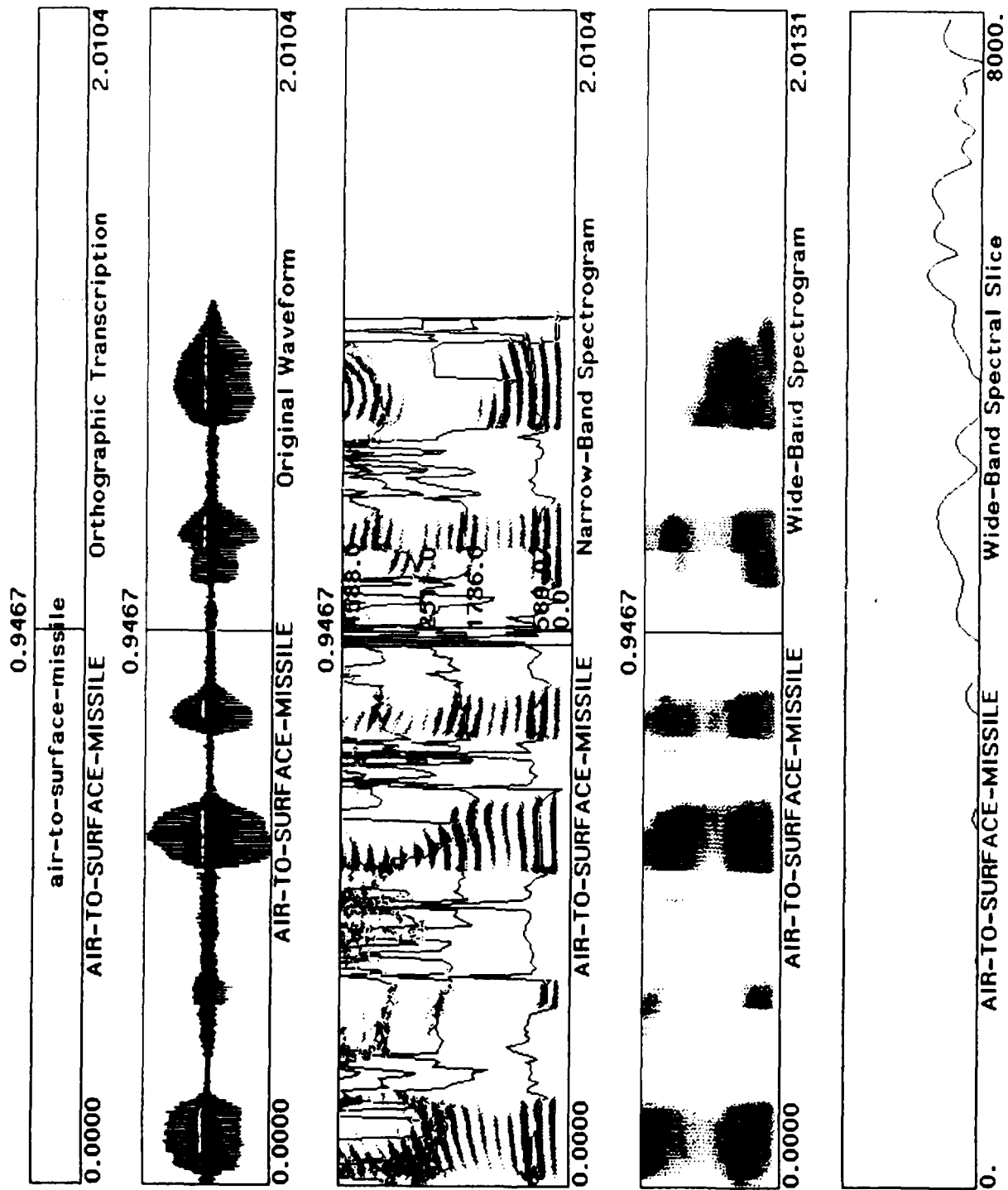


Figure 3.2. Example Capability of SPIRE

RESULT-ARRAY

(SPIRE:ATT-VAL (SEND (SPIRE:UTTERANCE PATHNAME)
:FIND-ATT ATT-NAME)))

where RESULT-ARRAY is the variable containing the result of the computation and ATT-NAME is the features selected to be used such as LPC spectrum or frication frequency. PATHNAME is the location of a particular utterance. After all the necessary computations, the utterance may be killed or unloaded as follows:

(SEND (SPIRE:UTTERANCE PATHNAME) :KILL)

SPIRE returns the results of computation in a form of array and the dimension of the array depends on the type of features used. Table 3.1 shows the list of array types returned for various features of SPIRE function calls.

Hardware

Hardware components used in this research are Lisp Machine, Array Processor, and Speech Digitizer. These features are described in the following subsections.

Lisp Machine Symbolics 3600 Series Lisp Machine is used to accommodate the language (LISP) and the software package (SPIRE). The Symbolics Lisp Machine is a powerful computer specifically designed to efficiently run Lisp code. The programs of SPIRE package are written in Lisp. Also, the machine has an efficient user interface and extensive graphical capabilities (1:13-15). The Floating Point Accelerator (FPA) is included in the machine to speed up floating point operations by about a factor of three. The FPA is an add-on card which is critical to the applications of speech recognition where heavy computations are generally required.

Array Processor The array processor utilized here is the FPA which is compatible with the SPIRE package. This processor is a special purpose device designed to

SPIRE:UTTERANCE

Parameters: pathname (required)

Type: function

Returns: utterance-flavor

Description: The utterance in the file "pathname" becomes the current utterance in SPIRE. If needed the utterance is loaded into memory from disk. This function must be called before any computation can take place.

:FIND-ATT

Parameters: att-name (required)

Type: message to utterance flavor

Returns: att

Description: att-name is a string that identifies what attribute the user desires SPIRE to compute. For example, assume we are to compute the Wide-Band Spectrum of an utterance stored in the file ":>PKIM>UTTS>ARM>UTT". First, select the utterance:

(SETQ TEMP1

(SPIRE:UTTERANCE ":>PKIM>UTTS>ARM>UTT"))

TEMP1 stores the utterance flavor for the next step:

(SETQ TEMP2

(SEND TEMP1 : FIND-ATT "WIDE-BAND SPECTRUM"))

TEMP2 now holds the att from which the actual values may be extracted (see next function).

SPIRE:ATT-VAL

Parameters: att (required)

Type: function

Returns: array (results of computation)

Description: This function returns the computed value of the att we are interested in. For example, if **TEMP2** holds the att (as discussed above), extract the values:

(SETQ TEMP3 (SPIRE:ATT-VAL TEMP2))

TEMP3 now holds the "Wide-Band Spectrum" values. Similar procedures are followed for obtaining the values of any of the standard SPIRE computations.

Figure 3.3. SPIRE Interface Functions (1:14)

Table 3.1. SPIRE Result Arrays (3:2-10)

<u>Attribute Name</u>	<u>Result Array</u>
Wide-Band Spectrum	2-D, 256 X N
Narrow-Band Spectrum	2-D, 256 X N
LPC Spectrum	2-D, 256 X N
LPC Coefficients	2-D, 19 X N
Formants (four)	2-D, 5 X N
LPC Gain Term	1-D, N
LPC Center of Gravity	1-D, N
Zero Crossing Rate	1-D, N
Frication Frequency	1-D, N
Total Energy	1-D, N

$N = \text{time} * \text{analysis rate}$

Table 3.2. Computational Time Comparison (1:15)

<u>Configuration</u>	<u>Ratio</u>	<u>Example</u>
FPS Array Processor	10	1.0 minute
Floating Point Accelerator	3	3.0 minutes
Bare Lisp Machine	1	10.0 minutes

quickly handle computations on large arrays of data files. The FPA is connected to the Symbolics Lisp Machine through a UNIBUS interface. This array processor significantly saves the computational time required for certain SPIRE functions (1:15). An approximate comparison of a bare Symbolics Lisp Machine, one with FPA, and one with FPS array processor is shown in table 3.2.

Speech Digitizer For the process of converting raw speech waveforms into the binary data, an analog-to-digital converter is needed. In this study, Digital Sound Corporation (DSC) A/D converter is utilized. The DSC is connected to the Symbolics Lisp Machine via UNIBUS interface. The digitized data of speech waveforms may be either directly sent to Lisp Machine or prerecorded for transferring (3:2-11).

IV. System Design

This chapter presents the major processing functions and design concept of the system developed in the research. First, it will provide details on how the templates and utterances are processed before dynamic time warp is performed to calculate the distances between each template word and testing utterance word. Next, a brief description of dynamic time warp is presented. Then, the algorithm of creating merged template will be explained. Finally, the vocabulary used in this study is given.

Utterance Processing

Processing of an utterance requires specific computations executed by SPIRE on the original digitized waveform, and any additional processing called by the programming. Several Lisp functions are available for this study depending on the desired arrays of data (See Appendix A). The methods of feature extraction and other processings are discussed below.

Feature Extraction As discussed in chapter three, feature extraction requires some calling functions of SPIRE. In order to perform the necessary computations and obtain the desired arrays of data, the filename of an utterance and the name of features should be inputted. The process is accomplished by calling COMPUTE-ATT (See Appendix A).

In Dawson's thesis, several features or feature sets were applied and the combination of LPC Spectrum, Formants, and Frication Frequency resulted in the best accuracy of speech recognition. For the purpose of concentrating this research on creating an optimal template set, only the feature set above was utilized in processing the utterances throughout the research. The optimal template set is a reference

dictionary which is reliable for multiple speakers. The above three features are described individually in the following paragraphs.

LPC Spectrum In order to obtain LPC spectrum, the original waveform is preemphasized and run through a 256 point Fast Fourier Transform (FFT) routine incorporating a Hamming window. The calculation is achieved using a filter bandwidth of 300.0 Hz and the LPC coefficients. The results are returned in 256 discrete frequency components representing from 0 to 8000 Hz in log-magnitude form. Therefore, processed utterance will be two dimensional arrays, 256 X N, where N is proportional to the time length of the utterance. However, to reduce computational time, LPC Spectrum is compressed to 16. Figure 4.1 shows an example of LPC Spectrum Slice as well as Formants and Frication Frequency.

Formants Formant values are computed from the LPC Spectrum. The peaks are found by fitting a polynomial to each LPC Spectral Slice. Then, the polynomial is differentiated and solved for zeros. SPIRE returns the result of Formants as a two dimensional array, 5 X N, where N is proportional to the length of an utterance. Rows zero through four of this array represent the five formant frequency sets. However, row zero is not used because of inconsistency of the data. An additional feature, Frication Frequency, is utilized in the research since the formants lose track during the fricative sounds.

Frication Frequency It is a fair indicator of whether a fricative or vowel sound is occurring. Throughout each utterance, it attempts to determine the frequency of fricative sounds. During non-fricative sounds, the frequency is below 500 Hz, and during fricative sounds, the frequency is above 1000 Hz. These frequency values are approximated numbers based on the series of testing. In order to keep track of these fricative sounds during the recognition process, the algorithm multiplies a local distance by 0.4 whenever the frequency of that local position is greater

than 1500 Hz.

Other Processings Additional processings were necessary to perform speech recognition process more efficiently. Clipping, Median Filtering, Frequency Compression, and Energy Normalization are utilized in the thesis. These processings will reduce computing time, eliminate unnecessary data, and prevent side effects. They are discussed below.

Clipping Purpose of the processing is to ignore the last five time slices of all the results returned by the SPIRE functions. Due to the predictive nature of the LPC coefficients calculations, the last five time slices can not be computed and these are returned by SPIRE as zero values. Therefore, any feature associated with the LPC coefficients also has zero values in the last five time slices. Formants calculation is one of these examples. The values of LPC coefficients are used to calculate the formant frequencies. In order to set the uniformity during the recognition process, the last five time slices of any utterance are ignored no matter which features have been applied on the utterance.

Median Filtering This processing excludes unnecessary spikes in the formant trackings. As mentioned in Formant section, the Formants lose track during fricative sounds. Therefore, the processing was necessary for an efficient recognition of speech signals. For more detail, refer to Lisp function MEDIAN-FILTER in Appendix A.

Frequency Compression It is performed to reduce computational time, emphasize resolution in the lower frequencies, and de-emphasize resolution of the higher frequencies. The results of LPC Spectrum are compressed from 256 discrete frequency components down to 16. The lower 132 frequency components (0 to 4,125 Hz) are linearly compressed to 12 components and the upper 124 components (4,125 to 8,000 Hz) are linearly compressed to 4 components. However, the process is

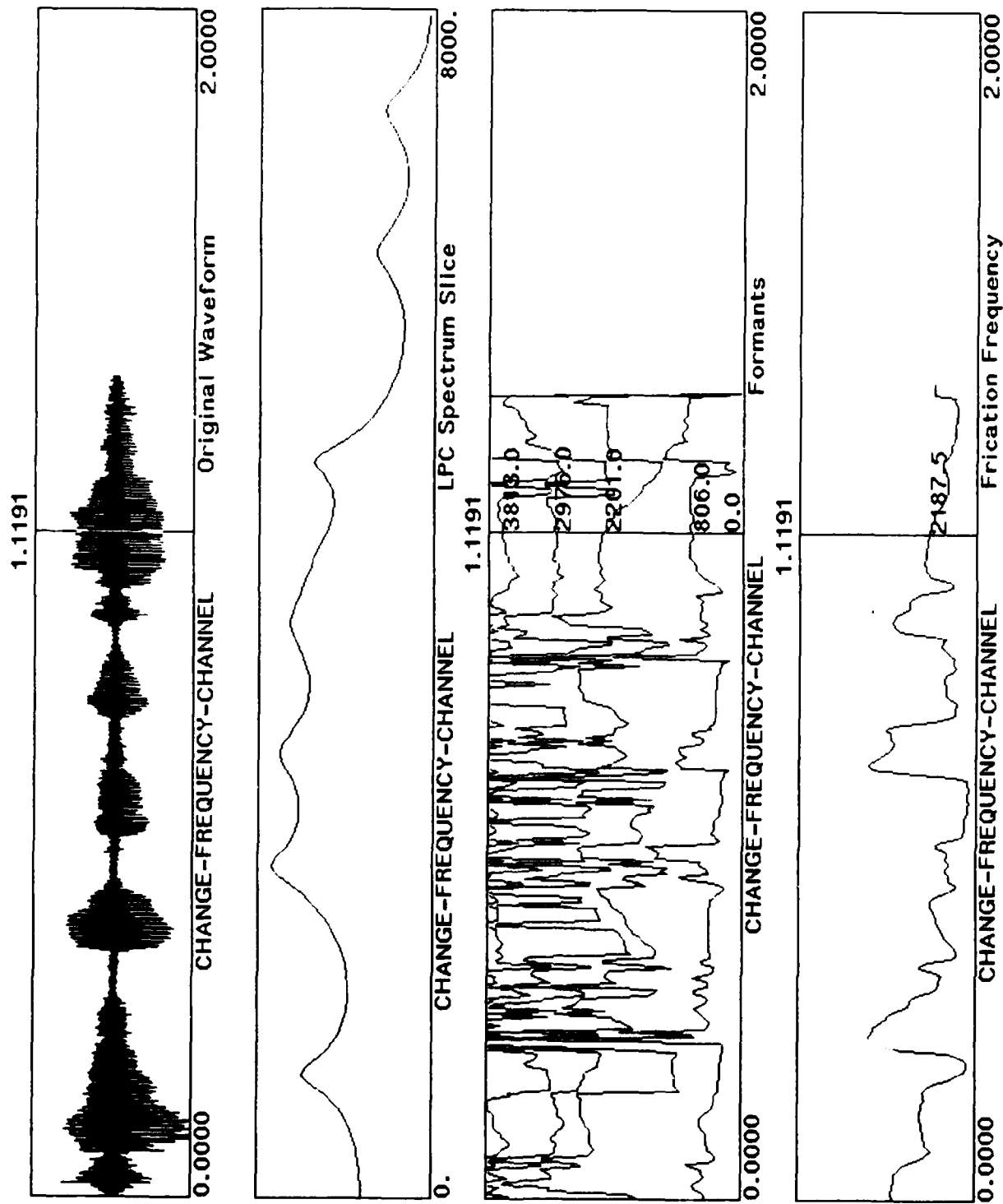


Figure 4.1. Three Features of Processing

achieved by averaging instead of addition since the original speech waveforms are preemphasized by SPIRE. Refer to Lisp function FREQUENCY-COMPRESS-LFE of Appendix A for more detail.

Energy Normalization For the purpose of solving energy disparity problem, each time slice of LPC Spectrum is energy normalized. With this problem, the system is unable to recognize the utterances properly. See ENERGY-NORMALIZE in Appendix B for more information.

Dynamic Time Warping

As discussed in Dawson's thesis (3), dynamic time warping is a method by which speech patterns are nonlinearly time aligned. This time alignment is necessary due to the nonlinear time variations common in speech. In this research, the same dynamic time warping is utilized for calculations of the distances between each template word and the testing utterance word. However, the algorithm is modified so that it can search all seventy words and display them in five separate screens. Also, a different weight scheme is used for three directions (horizontal, vertical, and diagonal) of the path.

Merged Template

The fundamental goal in speaker-independent speech recognition system is that of creating a reference dictionary or reference template which can be reliably used with many different speakers. Several techniques have been tested to achieve this goal by many engineers, but none of them were successful enough to solve all the problems associated with speech recognition process. The technique of merging template applied in this thesis is a one-time preprocessing method. The diagram of this technique is displayed in figure 4.2. And the description of how this technique applies to the algorithm that creates an optimal template is discussed in the following paragraphs.

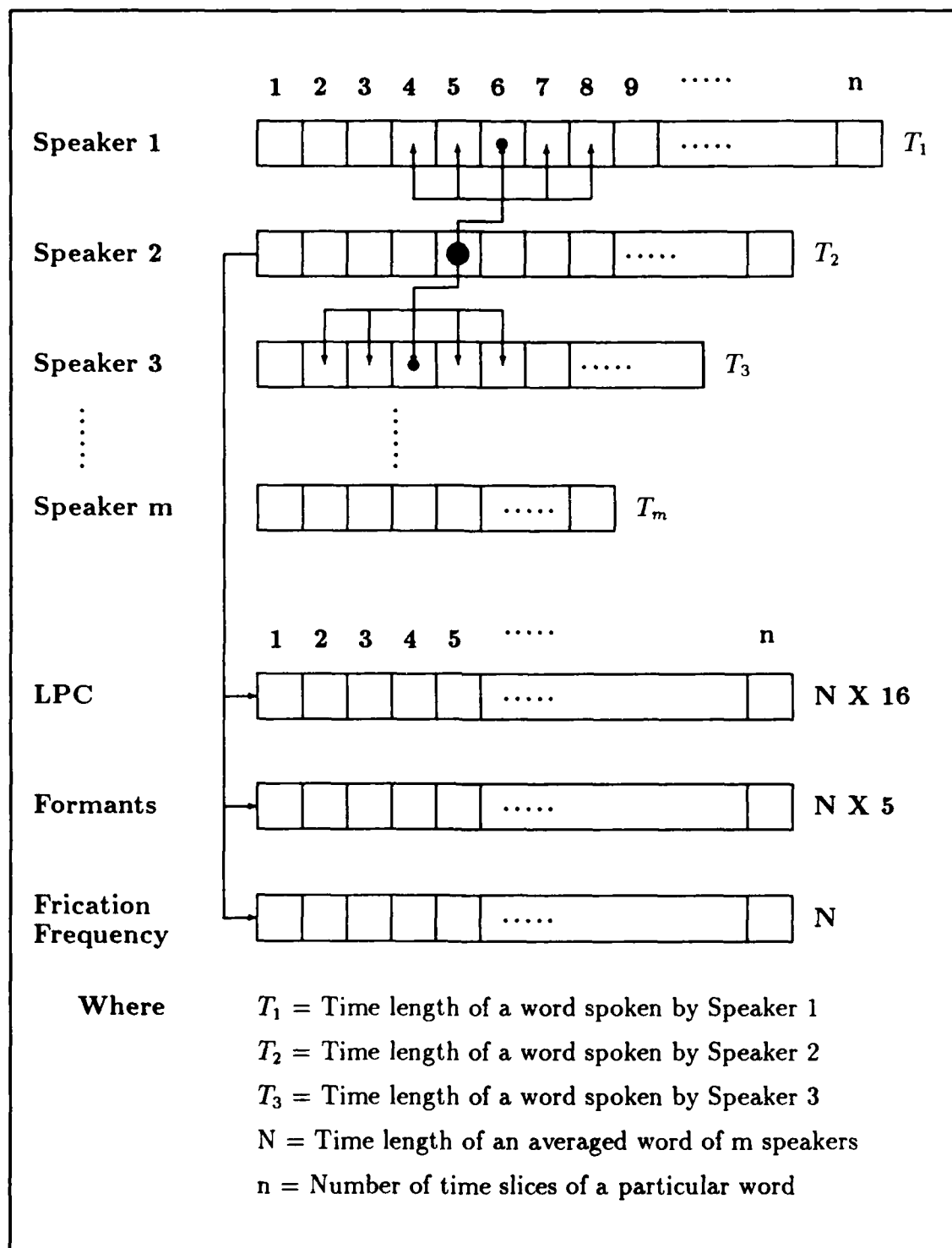


Figure 4.2. Template Merging

Reference Utterance First step of this technique is to select a reference utterance. The program obtains the first words from the vocabularies of specified number of speakers and calculates the average time length of these utterances. Then it scans the utterances and selects the one that is closest to this averaged time length. The process is repeated for each of the seventy words in the vocabulary. The selected utterance will be the reference utterance for the rest of steps required to create an optimal template. For the case of figure 4.2, the selected utterance would be the word spoken by speaker two.

Determining Nearest Index The next step is to determine the nearest index at each time slice. However, to better understand the technique applied here, the description of a processed utterance is appropriate. When an utterance is processed with the three features (LPC Spectrum, Formants, and Frication Frequency), the resulted data contains three sets of arrays (16 X N, 5 X N, and N) where N is proportional to time length of the utterance. Notice 16 X N is the result of LPC Spectrum, 5 X N is the result of Formants, and N is the result of Frication Frequency.

The program takes LPC Spectrum array of the reference utterance and scans through every time slice in time length, N. At each time slice, it looks at LPC Spectrum arrays of the other utterances and determines the center point for those utterances which corresponds to this particular time slice. This step is necessary because every utterance has a different time length even though the words spoken are same. The equation for this center point, C is shown below.

$$C = i \cdot (T_m/T_r) \quad ; \text{ For } i = 1 \text{ to } n \quad (4.1)$$

where n = Number of time slices of the reference utterance.
 i = Particular time slice of the reference utterance.
 T_m = Time length of word spoken by speaker m .
 T_r = Time length of reference utterance.

Notice the center point is at sixth time slice for speaker one and at fourth time slice for speaker three in figure 4.2. Next, the program reads a specified radius to determine the radius of other utterances which gives the bandwidth of time slices to be looked at. The specified radius is a variable input controlled by the user. The radius of other utterances is evaluated using equation shown below.

$$R = r \cdot (T_m/T_r) \quad ; \quad \text{For } r \geq 1 \quad (4.2)$$

where

r = Specified radius.

R = Radius of other utterances to be looked at for any particular time slice of the reference utterance.

For each time slice of the reference utterance, a number of time slices in other utterances is selected to be scanned. The number of time slices in other utterances is determined using the equation below.

$$K = 2R + 1 \quad (4.3)$$

where K = Number of time slices of other utterances under scanning region.

All the time slices in the scanning region are analyzed and only one time slice from each utterance is selected which is the closest to that particular time slice of the reference utterance. Now only one set of data from each time slice is obtained by the computer.

Averaging Time Slice For each time slice of the reference template, a number of data from the corresponding time slices will be available. The number depends upon the number of speakers used in this process. For example, if the vocabularies of five speakers are used, five sets of data for each time slice of the reference utterance will be available. The final step is to average these data. The data are added and divided by the number of speakers used. The averaged data for each time slice are stored in the newly created LPC array. The above process will be repeated for the arrays of Formants and Frication Frequency.

1. Advise	25. Five	48. Profile
2. Affirmative	26. Flares	49. Radar
3. Aft	27. Forward	50. Range
4. Air-to-Air	28. Four	51. Report
5. Air-to-Surface	29. Foxtrot	52. Rhaw
6. Alpha	30. Frequency	53. Search
7. Arm	31. Fuel	54. Select
8. Backspace	32. Gun	55. Seven
9. Bearing	33. Heading	56. Six
10. Bravo	34. Hundred	57. SMS
11. Cancel	35. Knots	58. South
12. Chaff	36. Lock-On	59. Station
13. Change	37. Map	60. Strafe
14. Charlie	38. Mark	61. Tail
15. Channel	39. Miles	62. Target
16. Clear	40. Minus	63. Thousand
17. Confirm	41. Missile	64. Threat
18. Degrees	42. Negative	65. Three
19. Delta	43. Nine	66. Two
20. East	44. North	67. Waypoint
21. Echo	45. Nose	68. Weapon
22. Eight	46. One	69. West
23. Enter	47. Radar	70. Zero
24. Fault		

Figure 4.3. F-16 Cockpit Commands

Optimal template Merged utterance consists of a set of three arrays, one for each feature that has been used. As shown in figure 4.2, the LPC array will be two dimensional, $N \times 16$, where N is the proportional to the time length of the merged utterance and 16 is the compressed data which came from 256 discrete frequencies. The Formants array will also be two dimensional, $N \times 5$, where N is again proportional to the time length of the merged utterance and 5 is the number of formant frequency. However, SPIRE uses only four formant frequency components because of the inconsistency of data. The Frication Frequency will be one dimensional, N , where N is proportional to the time length. Finally, the optimal template is established by clustering seventy merged utterances.

Vocabulary

The vocabulary used in this research is shown in figure 4.3. Instead of using ten digits, actual commands of AFTI F-16 are incorporated. These seventy words best represent the vocabulary necessary for military environment. As will be discussed in chapter five, there are some similar words in the vocabulary that the system has trouble recognizing.

V. *Testing and Results*

This chapter presents an overall description of testing procedure and the results of each testing phase. The result of each phase is followed by an analysis. Testing of this research consisted of three phases. The first phase was accomplished by using Dawson's dynamic time warp algorithm. However, the program was modified to accommodate the vocabulary (70 words) and display waveforms on five separate screens. The second phase was performed to reduce scanning time and to eliminate any confusion between isolated words and connected words. The final phase was conducted to test the optimal template which is designed for accurate recognition of words spoken by multiple speakers.

Single Template with Connected Scan

The primary goal of the first phase was to apply all seventy words on Dawson's dynamic time warp algorithm instead of ten digits (zero through nine). To achieve the goal, a modification of the algorithm was necessary to accommodate the vocabulary and display speech waveforms properly. The waveforms of the vocabulary plus testing utterance produce too much information to be displayed on one screen. On a single screen, the data would be tightly compressed and might be hard to analyze. Therefore, the program was modified to display the data on five separate screens. However, the content of the data remained unchanged. *Connected Scan* refers to the scheme, which is established in Dawson's algorithm, to inform *Process of Searching for Match* that the testing utterance has connected digits.

The reference template was created using the vocabulary of one person (Matt). Matt is one of five speakers whose utterances were digitized to conduct this research. The names of five speakers are Matt, Gary, Dave, Kris, and Debbie. The vocabulary list is shown in Figure 4.3. Notice the size of vocabulary is significantly increased from Dawson's ten digits. Large amount of disk space and computational time was

Table 5.1. Selected Utterances

AFT	FLARES	AIR-TO-AIR
ARM	HUNDRED	BACKSPACE
CHAFF	NEGATIVE	THOUSAND
DELTA		

required to create data files. Therefore, only fifty words (ten words from each of five speakers) were arbitrarily selected for testing purpose. The list of selected words is shown in table 5.1.

The results of this phase showed sixty percent accuracy which means that only thirty out of fifty words were correctly recognized. However, ten of thirty words are those used to create the reference template. When the template words and the testing words are spoken by the same speaker, the system is called *Speaker-Dependent*. Therefore, in order to measure the *Speaker-Independence* of the system properly, those ten words spoken by Matt should be excluded from the list of testing utterance (fifty words). In other words, only forty words should be considered for the measurement of *Speaker-Independence*. Then, the actual results indicate only fifty percent accuracy since twenty words out of forty testing words were correctly recognized. The result is shown in table 5.2.

The scheme of connected scan is not appropriate in the case of template with large vocabulary. The results showed that the program was confused whether the testing utterance was isolated or connected. The problem was especially serious

Table 5.2. Single-Template Results (Connected Scan)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	4/10	40
Kris (Female)	2/10	20
Dave (Male)	8/10	80
Gary (Male)	6/10	60
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 30/50	 60

for vocabulary sets which have words with different time lengths. For example, Debbie's "DELTA" was recognized as "AFT-AFT" and Gary's "AIR-TO-AIR" was recognized as "BEARING-ENTER". As a solution, a new function was designed and implemented for the next phase of testing.

Single Template with Isolated Scan

In order to solve the problem observed in phase one, the program called *Isolated Scan* was written. The program reduced scanning time to approximately thirty percent of the connected scan. It directed the computer to assume that the testing utterance is a single word, avoid the unnecessary steps, and search for the word with minimum distance.

However, the program had a tendency to select the shortest template word when the testing utterance was long and was unable to find a good match. To eliminate this tendency, a weight scheme was established. The weight scheme depended on time lengths of individual word. The calculated distance between the template word and the testing word was multiplied by L_u/L_t where L_u is the time length of testing utterance and L_t is the time length of template word.

Once more, the vocabulary of one speaker (Matt) was used in creating the reference template. The selected fifty words of phase one were tested. The testing results indicated sixty-eight percent accuracy. However, for proper measurement of *Speaker-Independent* system, ten words from *Matt* should be excluded. Then, the actual accuracy is only sixty percent. The resulting data are displayed in table 5.3.

Throughout the phase of testing, significant time reduction and complete elimination of confusion problem were observed. However, a bug in the SPIRE system was discovered. The function, which calculates the frequency of fricative sounds, was not operating properly. As discussed in chapter three, the feature of frication frequency obtains an one-dimensional array, N , where N is proportional to time length of utterance. But the SPIRE system returned the array that was not proportional

Table 5.3. Single-Template Results (Isolated Scan)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	8/10	80
Kris (Female)	3/10	30
Dave (Male)	5/10	50
Gary (Male)	8/10	80
Matt (Male)	10/10	100
TOTAL	34/50	68

to the time length of the utterance. Instead, it was always constant length of array. An effort to correct this system error was initiated. After correcting the error, the test was rerun and the results are presented in table 5.4.

Merged Template

Last phase of testing was accomplished by utilizing newly designed algorithm called *Merged Template*. The purpose of the algorithm is to create an optimal template set which would be reliable for recognition of words spoken by multiple speakers. The algorithm combines the vocabularies of two or more speakers and produces one template. The merged template requires less disk space than any single-speaker template. In fact, the size of template file decreases as more speakers are used

Table 5.4. Single-Template Results (*Isolated Scan*)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	9/10	90
Kris (Female)	5/10	50
Dave (Male)	5/10	50
Gary (Male)	8/10	80
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 37/50	 74
* After Correcting FF and Modification		

Table 5.5. Merged-Template Results (Two Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	10/10	100
Kris (Female)	10/10	100
Dave (Male)	10/10	100
Gary (Male)	5/10	50
Matt (Male)	10/10	100
TOTAL	45/50	90

when creating merged templates. Also, creating a merged template is a one-time calculation and preprocessing method. Once the template is produced, the rest of recognition process is the same as phase one.

A series of testing was conducted using merged templates which were created by combining different numbers of speakers. First, the merged template of one male and one female, was tested with fifty selected words of phase one. Then the process was repeated for three speakers (two males and one female), four speakers (two males and two females), and five speakers (three males and two females). The results are shown in tables 5.4 through 5.7.

As can be seen from the results, the merged template of two-speakers has ninety percent accuracy, the merged template of three-speakers obtains ninety-four

Table 5.6. Merged-Template Results (Three Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	9/10	90
Kris (Female)	8/10	80
Dave (Male)	10/10	100
Gary (Male)	10/10	100
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 47/50	 94

Table 5.7. Merged-Template Results (Four Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	10/10	100
Kris (Female)	10/10	100
Dave (Male)	10/10	100
Gary (Male)	10/10	100
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 50/50	 100

Table 5.8. Merged-Template Results (Five Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	8/10	80
Kris (Female)	10/10	100
Dave (Male)	10/10	100
Gary (Male)	10/10	100
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 48/50	 96

percent, the merged template of four-speakers achieves one hundred percent, and the merged template of five-speakers gains ninety-six percent accuracy. The reason for better accuracy with the merged template of four-speakers instead of the merged template of five-speakers is uncertain. It seems that the recognition accuracy of the merged template increases as the number of speakers combined are increased. However, after it reaches a certain number of speakers, the recognition accuracy decreases as the number of speakers combined are increased. In other words, to produce the best merged template, the optimal number of speakers should be determined before the vocabularies of the speakers are combined. The results of all the merged templates show the recognition accuracy above ninety percent. Based on these results, a conclusion may be drawn that the technique of merged template can be one of solutions to *Speaker-Independent* system. However, before any conclusion, it would be appropriate to have additional words tested.

Additional Utterances In order to confirm the efficiency of merged template and determine the optimal number of speakers in creating the best template set, fifty additional words (ten from each of five speakers) were arbitrarily selected. However, only two best template sets (four-speakers and five-speakers) were tested at this time. The list of ten additional utterances is shown in table 5.8. And the results of testing are shown in table 5.9 and 5.10.

Once more, the merged template of four-speakers resulted in better accuracy over the merged template of five-speakers. It had ninety-eight percent for four-speakers and ninety-six percent for five-speakers. Therefore, a conclusion may be drawn that the technique of merged template may be one of solutions to *Speaker-Independent* speech recognition system. Also, the optimal number of speakers which is to be combined in producing the best merged template is four. However, this number will probably vary as the vocabulary or processing environment is changed.

Table 5.9. Additional Selected Utterances

EAST	CLEAR	AFFIRMATIVE
FUEL	PROFILE	FREQUENCY
MAP	REPORT	WAYPOINT
TAIL		

Table 5.10. Additional Merged-Template Results (Four Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	9/10	90
Kris (Female)	10/10	100
Dave (Male)	10/10	100
Gary (Male)	10/10	100
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 49/50	 98

Table 5.11. Additional Merged-Template Results (Five Speakers)

<u>Speaker</u>	<u>Number of Correct Recognition</u>	<u>Percent</u>
Debbie (Female)	8/10	80
Kris (Female)	10/10	100
Dave (Male)	10/10	100
Gary (Male)	10/10	100
<u>Matt (Male)</u>	<u>10/10</u>	<u>100</u>
 TOTAL	 48/50	 96

VI. Conclusions and Recommendations

This chapter states conclusions drawn based on the system's performance and provides recommendations for future research in the area of *Speaker-Independent Continuous-Speech Recognition*. First, for conclusion of the research, important concepts, significant results, and notable constraints are briefly discussed. Then a variety of suggestions is presented for some sub-areas of the speech recognition process.

Conclusions

The concept of *Merged Template* is a simple potential solution to the *Speaker-Independent Speech Recognition System*. The processes of determining an average utterance, selecting a reference template which is closest to the average, using the reference template to find best time slices, and producing the optimal template, are not overly complicated. In fact, the technique used in the research is a well known method of pattern recognition. However, it is not certain the feature set utilized here is the best one. Other combinations of features may produce more reliable templates.

The results of the first phase indicate that Dawson's dynamic time warp algorithm works well for speaker-dependence but not for speaker-independence. Only fifty percent of selected words were correctly recognized. Furthermore, it was shown that the single-speaker template would not be reliable for multiple speakers because of the intrinsic characteristics of speech waveforms and individual pronunciation differences. Therefore, it is necessary to implement an optimal (universal) template to solve the above problems. The optimal template developed here is not a universal solution to the *Speaker-Independent System*. But it certainly would be a contribution to the development of perfect speech recognition system. Notice all the optimal templates resulted in the accuracies above ninety percent. In the case of merged template with four-speakers, the accuracy is almost one hundred percent.

This study focused on only isolated words because of computational time and disk space constraints. The system, however, is confused when the operating program does not specify whether the testing utterance is isolated or connected. This confusion would be a great contributor to the failure of any speech recognition system. But the modification of algorithm in phase two completely eliminates the confusion. In addition, the problem of the tendency to select the shortest template word is serious. In Dawson's research, the problem could not be realized because the vocabulary consisted of only ten digits (zero through nine). There is no significant difference in time lengths of ten digits. However, when the words like "AFT" and "AIR-TO-AIR" are used to create the template, Dawson's algorithm has trouble distinguishing the time length difference between two words. For example, the testing utterance, "AIR-TO-AIR" would be recognized as "AFT" because the accumulated distance between the template word "AFT" and the testing utterance "AIR-TO-AIR" results to be the minimum. This happens whenever the testing utterance is long and is unable to find a perfect match from the vocabulary list of the reference template. The problem, however, is completely eliminated by the weight scheme established in the phase two of the research.

Recommendations

The research was conducted in an ideal environment with four processings and only three feature sets. Also, only isolated speech, with a limited size of vocabulary, was applied. However, there still are many areas that need to be explored for a real-time implementation of speech recognition system. Suggestions for future research in some of these areas are presented in the following paragraphs.

Realistic Environment The speech recognition system of this research was tested in a laboratory environment where the background noise level was that of a computer laboratory. It will be more realistic to test the system in conditions of low noise, cockpit noise, and background conversations and then compare the per-

formance results of these different environments. The Armstrong Aerospace Medical Research Laboratory (AAMRL) at Wright-Patterson AFB has sufficient facilities for simulating various environmental conditions.

Vocabulary Expansion The performance of the system should be investigated with a larger size vocabulary. Seventy words are not adequate to test a general real-time implementation. A more realistic size of vocabulary would be two hundred words or more. However, a vocabulary of two hundred words would significantly increase computational time and require larger memory size. There are two ways to reduce the computational time. The first is to acquire an array processor which reduces the processing time to thirty percent of current time. And the other is to implement an algorithm called *Approximating and Eliminating Search Algorithm (AESAs)*. In the experiment conducted by Vidal (17), AESA saved ninety-four percent of computational time for a vocabulary which consisted of two hundred words. With a reasonable amount of work, the algorithm can be implemented on Symbolics 3600 Lisp machine using SPIRE and Lisp. Also, in order to accommodate the larger size of data files as the number of vocabulary is increased, a dedicated hardware with sufficient disk space is necessary.

Connected Speech Only isolated words were tested with the speech recognition system. However, in the real world, the system which is reliable for both speaker-independence and connected-speech, is desired. Follow-on research is recommended to investigate both areas. The data files of connected speech would require a large amount of disk space. Also, the process of connected speech will significantly increase the computational time. As suggested in the previous subsection, one needs to have an array processor and apply AESA to the system in order to successfully develop the recognizer.

Clustered Template Dawson obtained successful results with the clustered template (redundant template) in his thesis. He also concluded that the redundant template is efficient in handling different pronunciations of many speakers. Unfortunately, the redundant template pays a high price in terms of computational time and memory size. The suggestion given in two previous subsections would be the solution to these constraints, a dedicated hardware and application of AESA. With the dedicated hardware and application of AESA, the vocabularies of multiple speakers may be clustered to produce an universal reference template. This template then would contain several differently pronounced words for comparison of any testing utterance.

Different Sounds The system accurately discriminates among the differences in vowel sounds or vowel and fricative sounds. However, it is not certain whether it can discriminate between similar fricative sounds. For example, the system may have trouble distinguishing the words, "can" and "tan." Therefore, it is recommended to include more similar words in the vocabulary for future research of speech recognition.

Additional Features This thesis uses only one feature set (a combination of LPC Spectrum, Formants, and Frication Frequency). Other features such as Wide-Band and Narrow-Band Spectrums should be applied to the system. The merged template produces good results with the feature set used here. However, it is possible that the merged template with other features might produce better recognition accuracy.

Appendix A. *Programming List*

```

.....
;;;
;;; FILE SYSTEM:
;;;
;;; This is an organizational chart of the file
;;; system that has been created in order to run the
;;; speech recognition process more efficiently.
;;;
;;;
;;; SPL:>PKIM>
;;; |
;;; |--- TEMPLATES>
;;; |   |
;;; |   |--- speakername>
;;; |   |   |
;;; |   |   *.utt <---- Digitized Vocabulary Words
;;; |   |
;;; |   |--- *.bin <---- Ready-Templates
;;; |
;;; |--- UTTERANCES>
;;; |   |
;;; |   |--- speakername>
;;; |   |   |
;;; |   |   *.utt <---- Digitized Vocabulary Words
;;; |   |
;;; |   |--- speakername-r>
;;; |   |   |
;;; |   |   *.bin <---- Ready Utterances
;;; |
;;; |--- DTW>
;;; |   |
;;; |   |--- speakername-r>
;;; |   |   |
;;; |   |   *.bin <---- Composite Dtw Files
;;; |
;;; |--- THESIS>
;;; |   |
;;; |   |--- *.lisp <---- Program Files
;;; |
.....

```

```

.....
;;;
;;; Digitized Vocabulary Word: This is the individually
;;; digitized word that makes up the vocabulary (70 words).
;;; This is simply digitized version of the original
;;; speech waveform.
;;;
;;; Ready Template: The file is the processed result of
;;; the Digitized Vocabulary Word above. Features used are
;;; LPC Spectrum, Formants, and Frication Frequency. The
;;; result of this process will consists 70 sets of data,
;;; (array 16 X N, array 5 X N, array N), where N is
;;; proportional to the time length of individual word.
;;; It basically is a reference template (dictionary data)
;;; and stored on the Lisp Machine for speech recognition.
;;;
;;; Ready Utterance: This data is same as Ready Template.
;;; However, the individual words are processed separately
;;; in order to create a single testing word. Each Ready
;;; Utterance will therefore consists only one set of data,
;;; (array 16 X N, array 5 X N, array N), instead of 70 of
;;; them for the case of Ready Template.
;;;
;;; Composite Dtw File: This is the file created by doing
;;; dynamic time warp on Ready Template and Ready Utterance
;;; for purpose of distance scanning in finding matched word.
;;;
.....

.....
;;;
;;; CONVERT-WAVE-TO-UTT:
;;;
;;; It takes an input file containing digitized speech
;;; waveform and creates an utterance file. Infile is the
;;; pathname for the file on the host machine. Uttfile is
;;; the pathname for the utterance file to be created.
;;;
;;; * Rate is the sampling frequency (16K).
;;;
.....

```

```

(defun convert-wave-to-utt (infile &optional

```

```

                                (uttfiler "tmp.utt")
                                (rate 16000.))
(declare (special current-utt))
(with-open-file (stream infile :direction :input :characters nil
                                :byte-size 16.)
  (let* ((length (send stream :length))
         (wave-with-header (make-fix-array length))
         (wave (make-fix-array (- length 256.)
                                :displaced-to wave-with-header
                                :displaced-index-offset 256.)))
    (declare (sys:array-register wave))
    (dotimes (i length)
      (setf (aref wave-with-header i)
            (sign-extend-16 (swap-bytes (or (send stream :tyi)
                                            0))))))
    (setf current-utt (spire:create-utterance-from-waveform
                      wave rate))
    (spire:dump-utterance current-utt uttfiler)
    current-utt)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  LOAD-WAVE-INTO-RECORDING-BUFFER:                                ;;;
;;;                                                                    ;;;
;;;      It takes an input file containing digitized speech        ;;;
;;;  waveform and loads the data into SPIRE's Recording Buffer.    ;;;
;;;  This function is included to give the user access to the      ;;;
;;;  speech editing facilities provided by the Recording          ;;;
;;;  Layout in SPIRE.                                              ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun load-wave-into-recording-buffer (infile)
  (with-open-file (stream infile :direction :input :characters nil
                          :byte-size 16.)

    (let* ((file-length (send stream :length))
           (length (min (- file-length 256) spire:*recording-
                          buffer-size*)))

      (dotimes (i 256) (send stream :tyi))
      (dotimes (i length)

```

```

(setf (aref spire:recording-buffer-array i)
      (sign-extend-16 (swap-bytes (or (send stream
                                             :tyi) 0)))))

(setf (array-leader spire:recording-buffer-array 0)
      (1-length))

(send (send recording-buffer-utterance :find-cursor
                                         :marker-time)
      :set-values 0.0)

(send (send recording-buffer-utterance :find-cursor
                                         :cursor-time)
      :set-values 0.0)

(let ((wv-attr (send recording-buffer-utterance
                     :fine-attribute "Original Waveform")))
  (send wv-attr :update-monitors))))

(defun swap-bytes (x)
  (multiple-value-bind (hi lo)
    (floor x 256)
    (+ (lsh lo 8) hi)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; VOCABULARY:
;;;
;;; This is the list of words utilized in the vocabulary.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defvar *vocabulary*

  '("Advise" "Affirmative" "Aft" "Air-to-Air" "Air-to-Surface"
    "Alpha" "Arm" "Backspace" "Bearing" "Bravo" "Cancel"
    "Chaff" "Change" "Charlie" "Channel" "Clear" "Confirm"
    "Degrees" "Delta" "East" "Echo" "Eight" "Enter" "Fault"
    "Five" "Flares" "Forward" "Four" "Foxtrot" "Frequency"
    "Fuel" "Gun" "Heading" "Hundred" "Knots" "Lock-On" "Map"
    "Mark" "Miles" "Minus" "Missile" "Negative" "Nine" "North"
    "Nose" "One" "Point" "Profile" "Radar" "Range" "Report"

```

```

"Rhaw" "Search" "Select" "Seven" "Six" "SMS" "South"
"Station" "Strafe" "tail" "Target" "Thousand" "Threat"
"Three" "Two" "Waypoint" "Weapon" "West" "Zero"))

```

```

(defvar *vo-list*

```

```

'("01" "02" "03" "04" "05" "06" "07" "08" "09" "10"
  "11" "12" "13" "14" "15" "16" "17" "18" "19" "20"
  "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
  "31" "32" "33" "34" "35" "36" "37" "38" "39" "40"
  "41" "42" "43" "44" "45" "46" "47" "48" "49" "50"
  "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"
  "61" "62" "63" "64" "65" "66" "67" "68" "69" "70"))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  LOAD:
;;;
;;;  This function loads all of the thesis files necessary
;;;  to perform speech recognition processes.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(load "spl:>pkim>thesis>globals")
(load "spl:>pkim>thesis>utilities")
(load "spl:>pkim>thesis>word-search!")
(load "spl:>pkim>thesis>dtw")
(load "spl:>pkim>thesis>isolated-scan")
(load "spl:>pkim>thesis>myfile")
(load "spl:>pkim>thesis>ltemp")
(load "spl:>pkim>thesis>merge-templates-kab-style")

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;  WORD-SEARCH!:
;;;
;;;  This function presents a menu of the processes that
;;;  are required to create the composite dtw file and search
;;;  for minimum distance to choose the best matched word out
;;;  the vocabulary (Seventy F-16 Cockpit Commands).
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun word-search! ()
  (let* ((item-list '("Create Ready-Template File"
                      "Load Ready-Template File"
                      "Create Ready-Utterance File"
                      "Load Ready-Utterance File"
                      "Create Composite DTW File"
                      "Load Composite DTW File"
                      "Apply Connected Word Search"
                      "Apply Isolated Word Search"
                      "QUIT")))
    (menu (tv:make-window 'tv:momentary-menu
                        ':label "Word-Search!"
                        "Select one of the following..."))
    (choice))
    (send menu ':set-item-list item-list)
    (setq choice (send menu ':choose))

    (cond ((equal choice "Create Ready-Template File")
            (create-ready-template-file))
          ((equal choice "Load Ready-Template File")
            (load-ready-template-file))
          ((equal choice "Create Ready-Utterance File")
            (create-ready-utterance-file))
          ((equal choice "Load Ready-Utterance File")
            (load-ready-utterance-file))
          ((equal choice "Create Composite DTW File")
            (create-composite-dtw-file))
          ((equal choice "Load Composite DTW File")
            (load-composite-dtw-file))
          ((equal choice "Apply Connected Word Search")
            (scan-dtw *cdtw*)
            (word-search!))
          ((equal choice "Apply Isolated Word Search")
            (scan-dtw-isolated *cdtw*)
            (word-search!))
          ((equal choice "QUIT")))
    "You have exited Word-Search!")

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                         ;;;
;;;  CREATE-READY-TEMPLATE-FILE:                             ;;;
;;;                                                         ;;;

```

```

;;;      This function creates a Ready-Template file.  It is      ;;;
;;;      accomplished by reading each word of the vocabulary one  ;;;
;;;      by one.  Requested SPIRE computations are performed on   ;;;
;;;      individual word and the data is saved into a disk file.  ;;;
;;;      The user is prompted for both input and output pathnames. ;;;
;;;
;;;      Input :  Typing the name of speaker.                      ;;;
;;;      Output:  Writes Ready-Template File to Disk.              ;;;
;;;
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun create-ready-template-file ()
  (let* ((read-directory (string-append
                           "spl:>pkim>templates>"
                           (prompt-and-read :string
                                              "Please enter speaker
                                              name: ") ">")))
    (read-path)
    (write-path (string-append
                  "spl:>pkim>templates>"
                  (prompt-and-read :string
                                   "Please enter Ready-Template
                                   name: "))))
  (choice nil))
  (setq *t-set* nil)
  (setq *tempath* read-directory)
  (setq choice (menu-feature-set))
  (loop for v-word in *vocabulary* do
    (setq read-path (string-append read-directory
                                    v-word ".utt")))
    (terpri)
    (princ "Processing ")
    (princ read-path)
    (princ "...")
    (setq *t-set* (append *t-set*
                          (list
                           (cond ((equal choice
                                           "Wide Band Spectrum")
                                (process-utterance-wbs
                                 read-path))
                                ((equal choice
                                           "Narrow Band Spectrum")

```

```

                                (process-utterance-nbs
                                read-path))
                                ((equal choice
                                "LPC Spectrum")
                                (process-utterance-lpc
                                read-path))
((equal choice "Formants")
                                (process-utterance-formants
                                read-path))
                                ((equal choice
                                "LPC, Formants, Fr. Freq.")
                                (process-utterance-lpc-
                                formants-ff read-path))))))

;;;      (send (spire:utterance read-path) :kill))
      (dump-to-disk write-path (list *t-set* *tempath*))
      (word-search!))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;      CREATE-READY-UTTERANCE-FILE:
;;;
;;;      This function creates a Ready-Utterance file. It is
;;;      accomplished by reading a particular word and computing
;;;      requested features on that word. The data is saved into
;;;      a disk file. The user is prompted for both input and
;;;      output pathnames.
;;;
;;;      Input :   Typing speakername and a desired word.
;;;      Output:   Writes a Ready-Utterance File to Disk.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun create-ready-utterance-file ()
  (let* ((read-path
          (string-append
            "spl:>pkim>utterances>"
            (prompt-and-read :string
              "Name of Digitized Continuous Utterance :")
              ".utt")))
    (write-path
      (string-append

```

```

"spl:>pkim>utterances>"
(prompt-and-read :string
  "Name of Ready-Utterance : "))
(choice nil))
(setq choice (menu-feature-set))
(setq *ready-utterance*
  (cond ((equal choice "Wide Band Spectrum")
    (process-utterance-wbs read-path))
    ((equal choice "Narrow Band Spectrum")
    (process-utterance-nbs read-path))
    ((equal choice "LPC Spectrum")
    (process-utterance-lpc read-path))
    ((equal choice "Formants")
    (process-utterance-formants read-path))
    ((equal choice "LPC, Formants, Fr. Freq.")
    (process-utterance-lpc-formants-ff read-path))))
(setq *uttpath* read-path)

; (send (spire:utterance read-path) :kill)
  (dump-to-disk write-path (list *ready-utterance* *weight-list*
    *uttpath*)))

(word-search!))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  LOAD-READY-TEMPLATE-FILE:                                         ;;;
;;;                                                                    ;;;
;;;      This function loads a Ready-Template file.                   ;;;
;;;                                                                    ;;;
;;;  Input   : Typing filename of a Ready-Template.                  ;;;
;;;  Output  : Writes data of the Ready-Template on the buffer.       ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun load-ready-template-file ()
  (let* ((read-path (string-append
    "spl:>pkim>templates>"
    (prompt-and-read :string
      "Name of Ready-Template : "))))
    (load read-path)
    (setq *t-set* (car *data*))
    (setq *tempath* (cadr *data*)))

```

```

(word-search!))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  LOAD-READY-UTTERANCE-FILE:                                       ;;;
;;;                                                                    ;;;
;;;      This function loads a Ready-Utterance file.                 ;;;
;;;                                                                    ;;;
;;;  Input   : Typing filename of a Ready-Utterance file.           ;;;
;;;  Output  : Writes dat of the Ready-Utterance on the buffer.      ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun load-ready-utterance-file ()
  (let* ((read-path (string-append
                      "spl:>pkim>utterances>"
                      (prompt-and-read :string
                                        "Name of Ready-Utterance : "))))
    (load read-path)
    (setq *ready-utterance* (car *data*))
    (setq *weight-list* (cadr *data*))
    (setq *uttpath* (caddr *data*)))
  (word-search!))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  MENU-FEATURE-SET:                                                ;;;
;;;                                                                    ;;;
;;;      It provides a menu of features to be processed.             ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun menu-feature-set ()
  (let* ((item-list '("Wide Band Spectrum"
                     "Narrow Band Spectrum"
                     "LPC Spectrum"
                     "Formants"
                     "LPC, Formants, Fr. Freq."))
    (menu (tv:make-window 'tv:momentary-menu
                        ':label "Word-Search
Select Feature Set to Use..."))
    (choice))
    (send menu ':set-item-list item-list))

```

```

      (setq choice (send menu ':choose))
      choice))

;;;;
;;;; PROCESS-UTTERANCE-LPC-FORMANTS-FF:
;;;;
;;;; It processes an utterance using features of LPC
;;;; Spectrum, Formants, and Frication frequency and produces
;;;; output data necessary for Ready-Template and Ready-Utt.
;;;;
;;;;

(defun process-utterance-lpc-formants-ff (pathname)
  (let ((returned-list nil))
    (setq returned-list (list (row-normalize-array
                              (frequency-compress-lfe
                               (compute-att pathname
                                            "LPC Spectrum")))))
    (setq returned-list (append returned-list (list (regionize
                                                    (median-filter
                                                     (compute-att
                                                      pathname
                                                       "Formants"))))))
    (setq returned-list (append returned-list (list
                                          (make-array-from-fps-structure
                                           (compute-att
                                            pathname
                                             "Frication Frequency")))))
    (setq *weight-list* '(4.5 2))
    returned-list))

(defun make-array-from-fps-structure (array)
  (let ((return-array (make-array (array-dimensions array))))
    (dotimes (i (car (array-dimensions array)))
      (setf (aref return-array i) (aref array i)))
    return-array))

;;;;
;;;; ROW-NORMALIZE-ARRAY:
;;;;

```

```

;;;      It normalizes the rows of array in LPC Spectrum.      ;;;
;;;                                                                ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun row-normalize-array (array)
  (let* ((height (array-dimension-n 1 array))
         (width  (array-dimension-n 2 array))
         (total-energy 0)
         (result-array (make-array (list height width)
                                   ':initial-value 0)))
    (dotimes (row height)
      (setq total-energy 0)
      (dotimes (column width)
        (setq total-energy (+ total-energy (sqr
                                           (aref array row column))))))
      (setq total-energy (sqrt total-energy))
      (dotimes (column width)
        (aset (/ (aref array row column) (cond
                                                    ((= total-energy 0) 1)(t total-energy)))
              result-array row column)))
      result-array))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                ;;;
;;;      FREQUENCY-COMPRESS-LFE:                                ;;;
;;;                                                                ;;;
;;;      It compresses the frequency components computed        ;;;
;;;      in LPC Spectrum from 256 to 16.                        ;;;
;;;                                                                ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun frequency-compress-lfe (array)
  (let* ((length (array-dimension-n 1 array))
         (return-array (make-array (list length 16))))
    (do ((count 0 (1+ count)))
        ((= count length))
      (aset (row-average array count 0 10) return-array count 0)
      (aset (row-average array count 11 21) return-array count 1)
      (aset (row-average array count 22 32) return-array count 2)
      (aset (row-average array count 33 43) return-array count 3)
      (aset (row-average array count 44 54) return-array count 4)
      (aset (row-average array count 55 65) return-array count 5))

```

```

(aset (row-average array count 66 76) return-array count 6)
(aset (row-average array count 77 87) return-array count 7)
(aset (row-average array count 88 98) return-array count 8)
(aset (row-average array count 99 109) return-array count 9)
(aset (row-average array count 110 120) return-array count 10)
(aset (row-average array count 121 131) return-array count 11)
(aset (row-average array count 132 162) return-array count 12)
(aset (row-average array count 163 193) return-array count 13)
(aset (row-average array count 194 224) return-array count 14)
(aset (row-average array count 225 255) return-array count 15))
  return-array))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  REGIONIZE:                                                         ;;;
;;;                                                                    ;;;
;;;    This function takes an input Formants and assigns a           ;;;
;;;    region, for each point of time, based on the first and        ;;;
;;;    second formants. Each region represents a specific            ;;;
;;;    vowel sound.                                                    ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun xor (alist)
  (let ((count 0))
    (loop for thing in alist do
      (cond (thing
              (setq count (1+ count))))))
  (oddp count)))

```

```

(defun intersect (seg1 seg2)
  (let* ((x11 (nth 0 seg1))
         (y11 (nth 1 seg1))
         (x12 (nth 2 seg1))
         (y12 (nth 3 seg1))
         (x21 (nth 0 seg2))
         (y21 (nth 1 seg2))
         (x22 (nth 2 seg2))
         (y22 (nth 3 seg2))
         (m1 (/ (float (- y12 y11)) (- x12 x11)))
         (m2 (/ (float (- y22 y21)) (- x22 x21)))
         (x (/ (+ y22 (* m1 x12) (- 0 y12 (* m2 x22))) (- m1 m2)))
         (t1 (/ (- x x11) (- x12 x11))))

```

```

(t2 (/ (- x x21) (- x22 x21)))
(result (cond ((and (<= t1 1.0)
                    (>= t1 0.0)
                    (<= t2 1.0)
                    (>= t2 0.0))
              T)
        (T nil))))
      result))

(defun regionize (formants)
  (let* ((f1 0)
         (f2 0)
         (result (make-array (array-dimension-n 1 formants)
                             :type 'art-8b)))
    (loop for time fixnum from 0 below (array-dimension-n 1
                                                         formants) do
      (setq f1 (aref formants time 1))
      (setq f2 (aref formants time 2))
      ;(terpri) (princ f1) (princ ",") (princ f2) (princ "-")
      (cond ((xor (list (intersect (list f1 f2 1500 f2)
                                   '(0 1750 250 3500))
                     (intersect (list f1 f2 1500 f2)
                                   '(250 1750 450 3500))))
            ;(princ 1)
            (aset 1 result time))
            ;(aset 300 formants 1 time)
            ;(aset 2750 formants 2 time))
            ((xor (list (intersect (list f1 f2 1500 f2)
                                   '(250 1750 450 3500))
                     (intersect (list f1 f2 1500 f2)
                                   '(450 1750 700 3500))))
            ;(princ 2)
            (aset 2 result time))
            ;(aset 420 formants 1 time)
            ;(aset 2300 formants 2 time))
            ((xor (list (intersect (list f1 f2 1500 f2)
                                   '(450 1750 700 3500))
                     (intersect (list f1 f2 1500 f2)
                                   '(900 2500 901 3500))
                     (intersect (list f1 f2 1500 f2)
                                   '(600 1750 900 2500))))
            ;(princ 3)

```

```

(aset 3 result time))
;(aset 600 formants 1 time)
;(aset 2200 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(600 1500 601 1750))
          (intersect (list f1 f2 1500 f2)
                        '(600 1750 900 2500))
          (intersect (list f1 f2 1500 f2)
                        '(750 1500 1200 2500))))))

;(princ 4)
(aset 4 result time))
;(aset 700 formants 1 time)
;(aset 1800 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(750 1500 1200 2500))
          (intersect (list f1 f2 1500 f2)
                        '(600 1100 601 1500))
          (intersect (list f1 f2 1500 f2)
                        '(650 1100 1200 1750))
          (intersect (list f1 f2 1500 f2)
                        '(1200 1750 1201 2500))))))

;(princ 5)
(aset 5 result time))
;(aset 800 formants 1 time)
;(aset 1500 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(650 950 651 1100))
          (intersect (list f1 f2 1500 f2)
                        '(650 1100 1200 1750))
          (intersect (list f1 f2 1500 f2)
                        '(800 950 1200 1100))
          (intersect (list f1 f2 1500 f2)
                        '(1200 1100 1201 1750))))))

;(princ 6)
(aset 6 result time))
;(aset 900 formants 1 time)
;(aset 1100 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(350 1300 351 1750))
          (intersect (list f1 f2 1500 f2)
                        '(600 1300 601 1750))))))

;(princ 7)

```

```

(aset 7 result time))
;(aset 500 formants 1 time)
;(aset 1500 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(400 950 401 1300))
          (intersect (list f1 f2 1500 f2)
                        '(600 950 601 1300))))))

;(princ 8)
(aset 8 result time))
;(aset 500 formants 1 time)
;(aset 1000 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(200 500 201 1300))
          (intersect (list f1 f2 1500 f2)
                        '(400 500 401 1300))))))

;(princ 9)
(aset 9 result time))
;(aset 300 formants 1 time)
;(aset 900 formants 2 time))
((xor (list (intersect (list f1 f2 1500 f2)
                        '(400 500 401 950))
          (intersect (list f1 f2 1500 f2)
                        '(600 950 601 1100))
          (intersect (list f1 f2 1500 f2)
                        '(650 950 651 1100))
          (intersect (list f1 f2 1500 f2)
                        '(600 500 800 950))))))

;(princ 10)
(aset 10 result time))
;(aset 600 formants 1 time)
;(aset 800 formants 2 time))
(t ;(princ 0)
(aset 0 result time))))
;(aset 0 formants 1 time)
;(aset 0 formants 2 time))))
result))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  MEDIAN-FILTER:                                                    ;;;
;;;                                                                    ;;;
;;;  It smooths out the formant values and eliminates                ;;;
;;;  the glitches when the formant frequency loses a track.          ;;;

```

```

;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun median-filter (array)
  (let* ((rows (array-dimension-n 1 array))
         (columns (array-dimension-n 2 array))
         (return-array (make-array (list rows columns)))
         (window-vector (make-array 11)))
    (copy-array-contents array return-array)
    (do* ((column-index 1 (1+ column-index))
          ((= column-index columns))
          (do* ((row-index 5 (1+ row-index))
                ((= row-index (- rows 5))
                 (do* ((window-index (- row-index 5) (1+ window-index))
                       ((window-vector-index 0 (1+ window-vector-index))
                        ((= window-vector-index 11))
                        (aset (aref array window-index column-index)
                             window-vector window-vector-index))
                        (aset (aref (sort window-vector '<) 4)
                             return-array row-index column-index)))
                  return-array)))
          return-array)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; COMPUTE-ATT:
;;;
;;; This function computes an utterance using requested
;;; features that contains various attribute values.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun compute-att (pathname att-name)
  (let ((return-array))
    (terpri)
    (princ "Computing ")
    (princ att-name)
    (princ "...")
    (setq return-array (spire:att-val
                        (send (spire:utterance pathname)
                             :find-att att-name) nil))

    (princ "Done.")
    return-array))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  CREATE-COMPOSITE-DTW-FILE:                                       ;;;
;;;                                                                    ;;;
;;;      This function uses Ready-Template and Ready-Utterance      ;;;
;;;      to create a file needed for searching a matched word.      ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun create-composite-dtw-file ()
  (let* ((write-path (string-append
                      "spl:>pkim>dtw>"
                      (prompt-and-read :string
                                       "Please enter CDTW name
                                       to create: "))))
    (setq *cdtw* (compute-composite-dtw))
    (dump-to-disk write-path (list *cdtw* *weight-list*
                                   *tempath* *uttpath*))
    (word-search!)))

```

```

(defun compute-composite-dtw ()
  (let ((result-list nil))
    (princ "Count-Down: ")
    (loop for template in *t-set*
          for count from (length *t-set*) downto 0 do
            (princ (format nil "~D-" count))
            (setq result-list (append result-list
                                     (list (new-ready-dtw-lpc-formants
                                           template *ready-utterance*))))))
    (terpri)
    result-list))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  LOAD-COMPOSITE-DTW-FILE:                                       ;;;
;;;                                                                    ;;;
;;;      It loads the above file into the buffer.                  ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun load-composite-dtw-file ()

```

```

(load (string-append
      "spl:>pkim>dtw>"
      (prompt-and-read :string
                        "Please enter CDTW name to load: ")))

(setq *cdtw* (car *data*))
(setq *weight-list* (nth 1 *data*))
(setq *tempath* (nth 2 *data*))
(setq *uttpath* (nth 3 *data*))
(word-search!))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; NEW-READY-DTW-LPC-FORMANTS:
;;;
;;; This function is to keep track of fricative sounds.
;;; Whenever the frequency is higher than 1500 Hz, its local
;;; distance is multiplied by 0.4.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun new-ready-dtw-lpc-formants (template utterance)
  (let* ((dtw (timewarp (car template) (car utterance)))
         (m-dimension (array-dimension-n 1 dtw))
         (n-dimension (array-dimension-n 2 dtw))
         (return-dtw (make-array (array-dimensions dtw)
                                 :type 'art-16b)))
    (loop for m from 0 below m-dimension do
      (loop for n from 0 below n-dimension do
        (let ((frfrt (aref (caddr template) m))
              (frfru (aref (caddr utterance) n))
              (t-region (aref (cadr template) m))
              (u-region (aref (cadr utterance) n))
              (distance (* 1000 (car *weight-list*)
                          (aref dtw m n))))
          (cond ((or (> frfrt 1500)
                     (> frfru 1500)
                     (= t-region 0)
                     (not (= t-region u-region)))
                 (aset (fix distance) return-dtw m n))
                (t (aset (fix (* 0.4 distance)) return-dtw m n))))))
    return-dtw))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

;;;
;;; TIMEWARP:
;;;
;;; It receives a pair of arrays, determines their
;;; dimensionality and calls TIMEWARP-1D or TIMEWARP-2d.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defun timewarp (arrayM arrayN)
  (cond ((= 1 (array-#-dims arrayM)) (timewarp-1d arrayM arrayN))
        ((= 2 (array-#-dims arrayM)) (timewarp-2d arrayM arrayN))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; TIMEWARP-1D:
;;;
;;; This function performs Dynamic Time Warp on
;;; one-dimensional arrays.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun timewarp-1d (vectorM vectorN)
  (let* ((M (- (array-dimension-n 1 vectorM) 5))
         (N (- (array-dimension-n 1 vectorN) 5))
         (return-array (make-array (list M N))))
    (dotimes (m-index M)
      (dotimes (n-index N)
        (aset (abs (- (aref vectorM m-index) (aref vectorN n-index)))
              return-array m-index n-index)))
    return-array))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; TIMEWARP-2D:
;;;
;;; This function performs Dynamic Time Warp on
;;; two-dimensional arrays.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun timewarp-2d (arrayM arrayN)
  (let* ((M (- (array-dimension-n 1 arrayM) 6))

```

```

(N (- (array-dimension-n 1 arrayN) 6))
(length (cond((= (array-dimension-n 2 arrayM) 5) 2)
          ((= (array-dimension-n 2 arrayM) 16) 16)
          ((= (array-dimension-n 2 arrayM) 19) 19)
          (t
           (princ "Timewarp ERROR. Hit Control-Abort")
           (do ((x 0))
               ((= x 1))))))
(start (cond((= (array-dimension-n 2 arrayM) 5) 1)
        ((= (array-dimension-n 2 arrayM) 16) 0)
        ((= (array-dimension-n 2 arrayM) 19) 0)))
(distance 0)
(result-array (make-array (list M N))))
(loop for m-index from 0 below M do
  (loop for n-index from 0 below N do
    (setq Distance 0.0)
    (loop for v-index from start below (+ start length) do
      (setq distance (+ distance (abs (- (aref arrayM
                                              m-index v-index)
                                              (aref arrayN
                                              n-index v-index))))))

    (aset distance result-array m-index n-index)))
  result-array))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  CONNECTED-SCAN:                                                  ;;;
;;;                                                                    ;;;
;;;      This function scans the composite Dynamic Time Warp        ;;;
;;;  file and determines the best matched word out of the list.      ;;;
;;;  However, it is primarily designed for connected words.          ;;;
;;;  The algorithm used is the "One-Stage Programming                ;;;
;;;  Algorithm for Connected Word Recognition" by Herman Ney.        ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun scan-dtw (composite-dtw)
  (let* ((title (prompt-and-read :string "Title? "))
        (thresh (prompt-and-read :number "Threshold? "))
        (N (array-dimension-n 2 (car composite-dtw)))
        (D-list (mapcar 'make-array
                        (mapcar 'array-dimension-n (circular-list 1)

```

```

        composite-dtw)))
(B-list (mapcar 'make-array
               (mapcar 'array-dimension-n (circular-list 1)
                       composite-dtw)))
(from-template (make-array N :type 'art-8b))
(from-frame (make-array N :type 'art-16b))
(d-min)
(save-b)
(save-d)
(save-temp)
(a 1.0)
(b 0.5)
(return-list)
(dummy +1e^N))

;;; STEP 1

(terpri) (princ "Computing Accumulated Distance Array")
(terpri) (princ "Begin Step 1 ... ")
(loop for current-dtw in composite-dtw
      for current-ada in D-list
      for current-B in B-list do
  (loop for n from 0 below (array-dimension-n 1 current-dtw)
        sum (aref current-dtw n 0) into local-sum
        do (aset local-sum current-ada n)
        (aset 0 current-B n)))
(princ "Done.")

;;;STEP 2

(terpri) (princ "Begin Step 2 ... ")
(loop for i fixnum from 1 below N do
  (setq dummy +1e^N)
  (loop for current-dtw in composite-dtw
        for current-ada in D-list
        for current-B in B-list
        for k from 0 to (length composite-dtw) do
    (setq d-min (min (aref current-ada 0)
                     (apply 'min (mapcar 'aref D-list
                                           (mapcar '1- (mapcar
                                                         'array-dimension-n

```

```

(circular-list 1)

D-list))))))

(cond ((not (= d-min (aref current-ada 0)))
      (aset (+ i 1) current-B 0)))

(setq save-d (aref current-ada 0))
(setq save-b (aref current-B 0))
(aset (+ (aref current-dtw 0 i) d-min) current-ada 0)
(loop for j fixnum from 1 below (array-dimension-n 1
current-ada) do
  (setq d-min (min (+ (* (1+ a) (aref current-dtw j i))
                      (aref current-ada j))
                  (+ (aref current-dtw j i) save-d)
                  (+ (* b (aref current-dtw (1- j) i))
                     (aref current-ada (1- j))))))
  (setq save-temp (aref current-B j))
  (cond ((= d-min (+ (aref current-dtw j i) save-d))
        (aset save-b current-B j))
        ((= d-min (+ (* b (aref current-dtw (1- j) i))
                      (aref current-ada (1- j))))
         (aset (aref current-B (1- j)) current-B j)))
  (setq save-d (aref current-ada j))
  (setq save-b save-temp)
  (aset d-min current-ada j))

(cond (((< (aref current-ada (1- (array-dimension-n 1
current-ada))) dummy)
      (setq dummy (aref current-ada (1- (array
-dimension-n 1 current-ada))))

      (aset k from-template i)
      (aset (aref current-B (1- (array-dimension-n 1
current-B)))
            from-frame i))))))

(princ "Done.")

;;;STEP 3

(terpri) (princ "Begin Step 3 ...")
(setq return-list

```

```

      (do* ((word-end (1- N) pred)
            (word (aref from-template (1- N)) (aref
              from-template pred))
            (pred (aref from-frame (1- N)) (aref
              from-frame pred))
            (answer (list word) (append (list word) answer))
            (boundry-list (list (list word pred word-end))
              (append (list (list word pred
                word-end)) boundry-list)))
            ((<= pred 1) boundry-list)))
    (my-plot-dtws composite-dtw
      (* thresh (length *weight-list*))
      return-list
      *tempath*
      *uttpath*
      title)))

(defun my-plot-dtws (cdtw threshold search-list tempath
  uttpath title)

  (loop for window in *window-list*
    for array-list in (split-into-five cdtw)
    for vocabulary in (split-into-five *vocabulary*)
    for vo-list in (split-into-five *vo-list*)
    do (my-plot-composite-dtw window array-list threshold
      vocabulary vo-list search-list
      tempath uttpath title)

      (cl:sleep 10)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  Isolated-Scan:                                                  ;;;
;;;                                                                    ;;;
;;;      It scans composite dtw file after telling the              ;;;
;;;  computer that the testing utterance is a single word.          ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun scan-dtw-isolated (composite-dtw)
  (let* ((title (prompt-and-read :string "Title? "))
        (thresh (prompt-and-read :number "Threshold? "))
        (N (array-dimension-n 2 (car composite-dtw))))

```

```

(M-list (mapcar 'array-dimension-n (circular-list 1)
               composite-dtw))

(distance-array (make-array (list (apply 'max M-list) N)))
(min-score cl:most-positive-fixnum)
(new-score)
(best-match))

(loop for dtw in composite-dtw
      for I from 0 to (length M-list) do
      (fill-distance-array distance-array dtw)
      (setq new-score (find-score distance-array dtw))
      (if (< new-score min-score)
          (progn (setf min-score new-score)
                  (setf best-match I))))

(my-plot-dtws composite-dtw
  (* thresh (length *weight-list*))
  (list (list best-match 0 N))
  *temppath*
  *uttpath*
  title)))

(defun fill-distance-array (distance-array dtw)
  (let ((M (array-dimension-n 1 dtw))
        (N (array-dimension-n 2 dtw))
        (a -.25)
        (b .75))
    (dotimes (i M)
      (dotimes (j N)
        (setf (aref distance-array i j) (aref dtw i j))))
    (dotimes (i M)
      (dotimes (j N)
        (cond ((not (and (zerop i) (zerop j)))
              (let ((local-dist (aref distance-array i j)))
                (setf (aref distance-array i j)
                      (min (+ (* (1+ a) local-dist)
                              (min (if (zerop i)
                                       cl:most-positive-fixnum
                                       (aref distance-array (1- i) j))))
                          (+ (* b local-dist)
                              (min (if (zerop j)
                                       cl:most-positive-fixnum
                                       (aref distance-array i (1- j)))))))
              (t (setf (aref distance-array i j) 0)))))))

```

```

(+ local-dist
  (min (if (or (zerop i)(zerop j))
            cl:most-positive-fixnum
            (aref distance-array (1- i)
                               (1- j))))))

(defun find-score (distance-array dtw )
  (let ((M (1- (array-dimension-n 1 dtw)))
        (N (1- (array-dimension-n 2 dtw))))
    (// (aref distance-array M N) (path-length distance-array dtw))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; SEARCH-DISTANCE-ARRAY:
;;;
;;; It is a program for the weight scheme which eliminates
;;; the tendency of selecting the shortest template word.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun path-length (distance-array dtw)
  (let ((M (1- (array-dimension-n 1 dtw)))
        (N (1- (array-dimension-n 2 dtw))))
    (do ((i 0) (j 0)
        (coordinate-count 1)
        (point-alist nil nil)
        (next-point))
        ((and (= i M) (= j N)) coordinate-count)
      (if (< i M) ;left
          (setf point-alist
                (cons (list (aref distance-array (1+ i) j)
                           (list (1+ i) j)) point-alist)))

          (if (< j N) ;up
              (setf point-alist
                    (cons (list (aref distance-array i (1+ j))
                               (list i (1+ j))) point-alist)))

              (if (and (< i M) (< j N)) ;diagonally
                  (setf point-alist
                        (cons (list (aref distance-array (1+ i) (1+ j))
                                   (list (1+ i) (1+ j))) point-alist)))

```

```

      (setf next-point (find-closest-point point-alist))
      (setf i (car next-point))
      (setf j (cadr next-point))
      (incf coordinate-count))))

(defun find-closest-point (point-alist)
  (do* ((plist point-alist (cdr plist))
        (point (car plist) (car plist))
        (min-dist cl:most-positive-fixnum)
        (min-coordinates nil))
    ((null plist) min-coordinates)
    (if (< (car point) min-dist) (progn (setf min-dist (car point))
                                         (setf min-coordinates
                                             (cadr point))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  PLOT-COMPOSITE-DTW:                                              ;;;
;;;                                                                    ;;;
;;;  It displays speech waveforms, dtw, and recognized word.      ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun my-plot-composite-dtw (window array-list threshold vocabulary
                             vo-list &OPTIONAL search-list tempath
                             uttpath title)

  (let* ((total-M (apply '+ (mapcar 'zl:array-dimension-n
                                     (circular-list 1) array-list)))
        (total-N (zl:array-dimension-n 2 (car array-list)))
        (x1 400)
        (y1 30)
        (y2 (+ y1 (min 595 total-M)))
        (yrange (- y2 y1))
        (x2 (floor (+ x1 (* 1 (* total-N (/ (float yrange)
                                              total-M))))))
        (xrange (- x2 x1)))
    (send window :expose)
    (send window :clear-window)
    (my-drawborder window x1 y1 x2 y2)
    (send window :draw-string title x2 (- y1 4) 0 (- y1 4) nil
      '(:dutch :bold :normal))
  )

```

```

(do* ((a-list array-list (cdr a-list))
      (k 0 (1+ k))
      (array (car a-list) (car a-list))
      (bottom y2 (- bottom current-yrange))
      (v-word (car vocabulary) (cond ((not (null a-list))
                                       (nth k vocabulary))
                                      (t nil))))
      (current-M (zl:array-dimension-n 1 array)
                  (if a-list (zl:array-dimension-n 1 array) 0))
      (current-N (zl:array-dimension-n 2 array)
                  (if a-list (zl:array-dimension-n 2 array) 0))
      (current-yrange (* yrange (/ (float current-M) total-M))
                       (* yrange (/ (float current-M) total-M))))
      ((null a-list))
      (send window :draw-line (- x1 70) (round bottom) (+ 10 x2)
              (round bottom))

      (my-display-waveform-rot window
                                (- x1 50) (round (- bottom
                                                    current-yrange))
                                (1- x1) (round bottom)
                                (string-append tempath v-word ".utt"))
      (send window :draw-string (nth k vocabulary)
              (- x1 60)
              (+ (round (- bottom (/ current-yrange 2))) 6) 0
              (+ (round (- bottom (/ current-yrange 2))) 6) nil
              '(:fix :roman :very-large))

      (loop for m-index from 0 below current-M do
        (loop for n-index from 0 below current-N do
          (if (< (aref array m-index n-index) threshold)
              (send window :draw-point
                      (round (+ x1 (* n-index (/ (float xrange)
                                                  current-N))))
                      (round (- bottom (* m-index (/ current-yrange
                                                         current-M))))))))

      (loop for n-index from 0 to total-N by 10 do
        (send window :draw-line
                  (round (+ x1 (* n-index (/ (float xrange) total-N))))
                  (- y2 4)
                  (round (+ x1 (* n-index (/ (float xrange) total-N))))
                  (+ 5 y2)))

```

```

(loop for m-index from 0 to total-M by 10 do
  (send window :draw-line
    (- x1 5)
    (round (- y2 (* m-index (/ (float yrange) total-M))))
    (+ x1 5)
    (round (- y2 (* m-index (/ (float yrange) total-M))))))
(if search-list
  (loop for word in search-list do
    (send window :draw-line
      (round (+ x1 (* (nth 1 word) (/ (float xrange)
                                     total-N))))
      y1
      (round (+ x1 (* (nth 1 word) (/ (float xrange)
                                     total-N))))
      (+ y2 70))
    (send window :draw-string (nth (car word) *vocabulary*)
      (- (round (+ x1
        (* (/ (+ (nth 1 word) (nth 2 word)) 2)
        (/ (float xrange) total-N)))) 10)
      (+ 65 y2)
      (- (round (+ x1
        (* (/ (+ (nth 1 word) (nth 2 word)) 2)
        (/ (float xrange) total-N)))) 10)
      (+ 65 y2)
      nil
      '(:fix :roman :very-large))
    (send window :draw-line
      x2 y2 x2 (+ y2 70)))
  (if uttpath (my-display-waveform window x1 (1+ y2) x2 (+ y2 50)
    uttpath))))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  DRAWBORDER:                                                         ;;;
;;;                                                                    ;;;
;;;  It draws a border on each of five splited windows.                ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun my-drawborder ( window x1 y1 x2 y2)
  (send window :draw-line x1 y1 x2 y1)
  (send window :draw-line x1 y1 x1 y2)

```

```

(send window :draw-line x1 y2 x2 y2)
(send window :draw-line x2 y2 x2 y1))

(defun split-into-five (l)
  (do* ((start 0 (+ start 14))
        (end 14 (+ end 14))
        (rlist (list (cl:subseq l start end)))
        (cons (subseq l start end) rlist)))
    ((= end 70) (reverse rlist))))

(defun my-display-waveform-rot (window x1 y1 x2 y2 pathname)
  (let* ((utt (send (spire:utterance pathname) :find-att
                    "original waveform"))
        (display-array (spire:att-val utt utt))
        (length (zl:array-length display-array))
        (width (- x2 x1))
        (height (- y2 y1)))
    (declare (sys:array-register display-array))
    (my-drawborder window x1 y1 x2 y2)
    (loop for index1 fixnum from 0 to (- length 2)
          for index2 fixnum from 1 to (1- length) do
            (send window :draw-line
                  (+ x1 (floor (* (+ (aref display-array index1) 32767.0)
                                   (/ width 65535.0))))
                  (- y2 (floor (* index1 (/ height (float length)))))
                  (+ x1 (floor (* (+ (aref display-array index2) 32767.0)
                                   (/ width 65535.0))))
                  (- y2 (floor (* index2 (/ height (float length))))))))))

(defun my-display-waveform (window x1 y1 x2 y2 pathname)
  (let* ((utt (send (spire:utterance pathname) :find-att
                    "original waveform"))
        (display-array (spire:att-val utt utt))
        (length (zl:array-length display-array))
        (width (- x2 x1))
        (height (- y2 y1)))
    (declare (sys:array-register display-array))
    (my-drawborder window x1 y1 x2 y2)
    (loop for index1 fixnum from 0 to (- length 2)
          for index2 fixnum from 1 to (1- length) do
            (send window :draw-line

```

```

(+ x1 (floor (* index1 (/ width (float length))))))
(+ y1 (floor (* (+ (aref display-array index1) 32767.0)
                (/ height 65535.0))))
(+ x1 (floor (* index2 (/ width (float length))))))
(+ y1 (floor (* (+ (aref display-array index2) 32767.0)
                (/ height 65535.0))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;;   Merged Template:
;;;
;;;   This is newly designed algorithm which combines the
;;;   vocabulary of several speakers to produce one optimal
;;;   template that is useful for multiple speakers.
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setf *speakers* '(">pkim>utterances>kab>"
                   ">pkim>utterances>deb>"
                   ">pkim>utterances>bar>"
                   ">pkim>utterances>dave>"
                   ">pkim>utterances>kris>"))

(defun create-merged-template-set ()
  (mapcar '(lambda (utt)
    (let* ((set (merge-composite-templates
                (create-composite-templates
                 *speakers* utt)))

           (temps (car set))
           (path (cadr set)))
      (copyf (fs:merge-pathnames path (string-append
                                     utt ".utt"))

              (fs:merge-pathnames ">pkim>utterances>ave>"
                                   (string-append utt ".utt")))

              :characters nil
              :byte-size 16.)
      (loop for speaker in *speakers* do
        (send (spi:utterance (fs:merge-pathnames speaker
                                                    (string-append utt ".utt"))))

```

```

:kill))
temps))
*vocabulary*))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  SELECT-CLOSEST-TO-AVERAGE-LENGTH:                                ;;;
;;;                                                                    ;;;
;;;      This function averages a number of different                ;;;
;;;      utterances and selects one which is closest to that        ;;;
;;;      averaged utterance as a reference utterance.                ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun select-closest-to-average-length (template-list)
  (let ((ave-length (// (apply '+
                           (mapcar 'array-dimension-n
                                   (circular-list 1)
                                   template-list))
                         (length template-list)))
        (best cl:most-positive-fixnum)
        diff
        p)
    (loop for template in template-list
          for template-no from 0 below (length template-list) do
            (setq diff (abs (- ave-length (array-dimension-n 1
                                                            template)))))
      (if (< diff best)
          (progn (setq best diff)
                 (setq P template-no))))
    P))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  FIND-INDEX-TO-NEAREST-VECTOR:                                    ;;;
;;;                                                                    ;;;
;;;      This function looks at a specified number of time          ;;;
;;;      slices and finds one that is closest to the index of        ;;;
;;;      a feature of the reference utterance. This is repeated      ;;;
;;;      for the other features that have been applied.              ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun find-index-to-nearest-vector (X P p-index)
  (let* ((C 3)
         (p-slices (array-dimension-n 1 P))
         (x-slices (array-dimension-n 1 X))
         (radius (max 1 (floor (/ (* C x-slices) p-slices))))
         (center (floor (/ (* p-index x-slices) p-slices)))
         (min-dist cl:most-positive-fixnum)
         dist
         index)
    (loop for x-index from (max 0 (- center radius))
          below (min x-slices (+ center radius)) do
            (setq dist (find-distance-between P p-index X x-index))
            (if (< dist min-dist)
                (progn (setq min-dist dist)
                       (setq index x-index))))
    index))

```

```

(defun find-distance-between (P p-index X x-index)
  (let ((cols (array-dimension-n 2 P))
        (dist 0))
    (loop for col from 0 below cols do
      (setq dist (+ dist (abs (- (aref P p-index col)
                                (aref X x-index col))))))
    dist))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  MERGE-COMPOSITE-TEMPLATES:                                       ;;;
;;;                                                                    ;;;
;;;    This function obtains a number of arrays found by           ;;;
;;;    the above step, averages them, and combine them as one       ;;;
;;;    template set.                                                 ;;;
;;;                                                                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```

(defun merge-composite-templates (composite-template-list)
  (let* ((LPC-list (mapcar 'car composite-template-list))
         (FRM-list (mapcar 'cadr composite-template-list))
         (FFR-list (mapcar 'caddr composite-template-list))
         (template-count (length composite-template-list))
         (P-nth (select-closest-to-average-length LPC-list))

```

```

(P-LPC (nth P-nth LPC-list))
(P-path (nth P-nth *speakers*))
(Q-slices (array-dimension-n 1 P-LPC))
(Q-LPC (cl:make-array '(',Q-slices 16) :initial-element 0))
(Q-FRM (cl:make-array '(',Q-slices 5) :initial-element 0))
(Q-FFR (cl:make-array Q-slices :initial-element 0))
(indices nil))
(declare (special P-LPC))
(loop for q-index from 0 below Q-slices do
  (setq indices (mapcar '(lambda (x index)
                          (find-index-to-nearest-vector
                           x P-LPC index))

                        LPC-list (circular-list q-index)))
  (loop for LPC in LPC-list
    for index in indices do
      (loop for col from 0 below 16 do
        (incf (aref Q-LPC q-index col)
              (// (aref LPC index col)
                  template-count))))

  (loop for FRM in FRM-list
    for index in indices do
      (loop for col from 0 below 5 do
        (incf (aref Q-FRM q-index col)
              (// (aref FRM index col)
                  template-count))))

  (loop for FFR in FFR-list
    for index in indices do
      (incf (aref Q-FFR q-index)
            (// (aref FFR index) template-count))))
(list (list Q-LPC (regionize Q-FRM) Q-FFR) P-path )))

```

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                    ;;;
;;;  CREATE-COMPOSITE-TEMPLATES:                                       ;;;
;;;                                                                    ;;;
;;;    This function performs additional processing required          ;;;
;;;    to eliminate errors and simplified computations and            ;;;
;;;    produce the final optimal template.                             ;;;
;;;                                                                    ;;;

```

.....

```
(defun create-composite-templates (speakers utterance)
  (mapcar `(lambda (speaker utt)
    (let ((pathname (fs:merge-pathnames speaker
      (string-append utt ".utt"))))
      (list (row-normalize-array
        (frequency-compress-lfe
          (compute-att pathname "LPC Spectrum")))
        (median-filter
          (compute-att pathname "Formants"))
        (make-array-from-fps-structure
          (compute-att pathname
            "Frication Frequency"))))))
    speakers (circular-list utterance)))
```

Appendix B. *Sample Results*

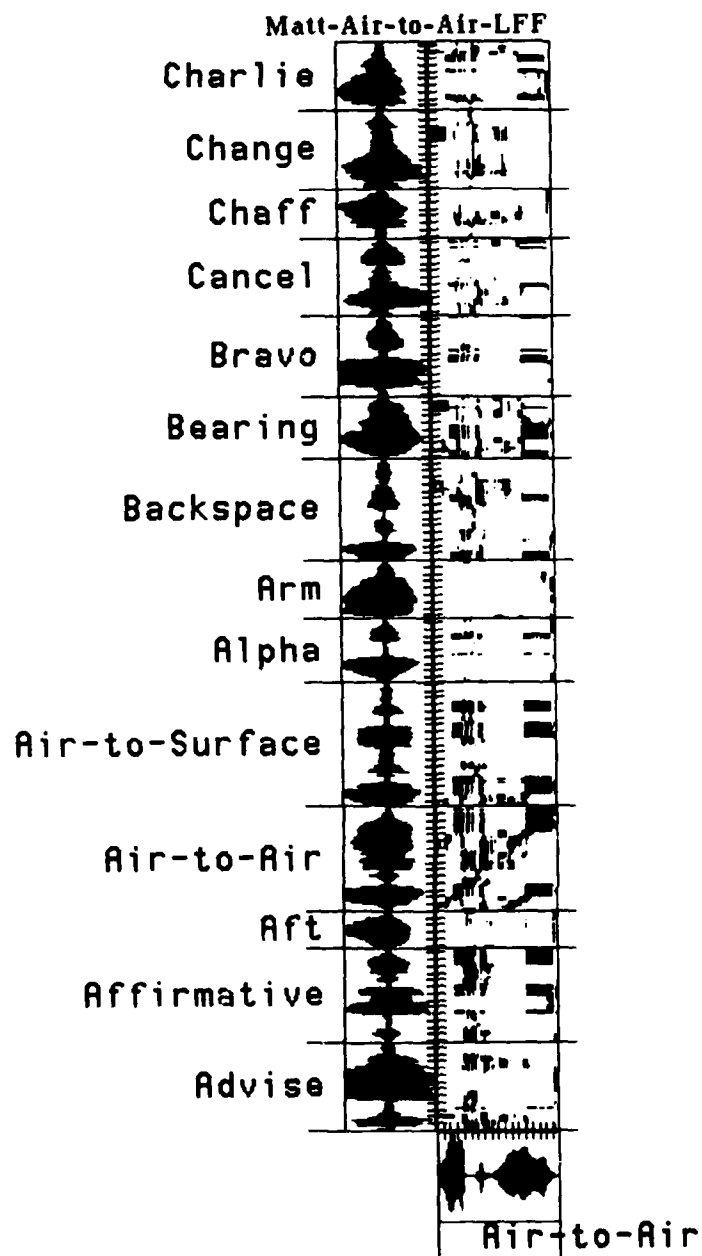


Figure B.1. Matt's Air-to-Air

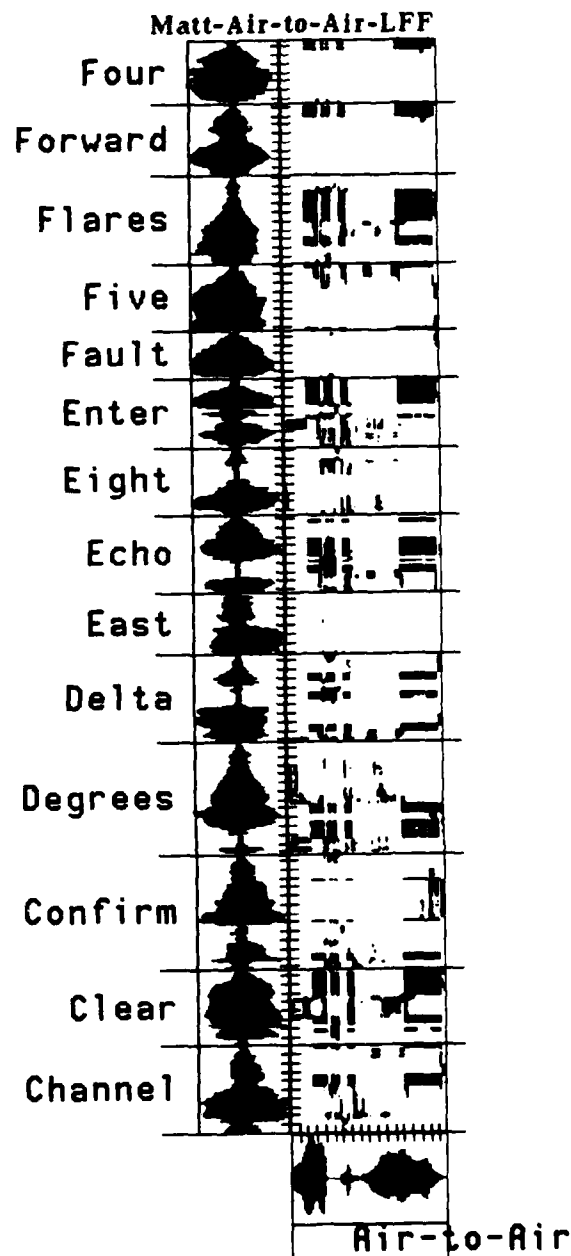


Figure B.1. Matt's Air-to-Air (Continued)

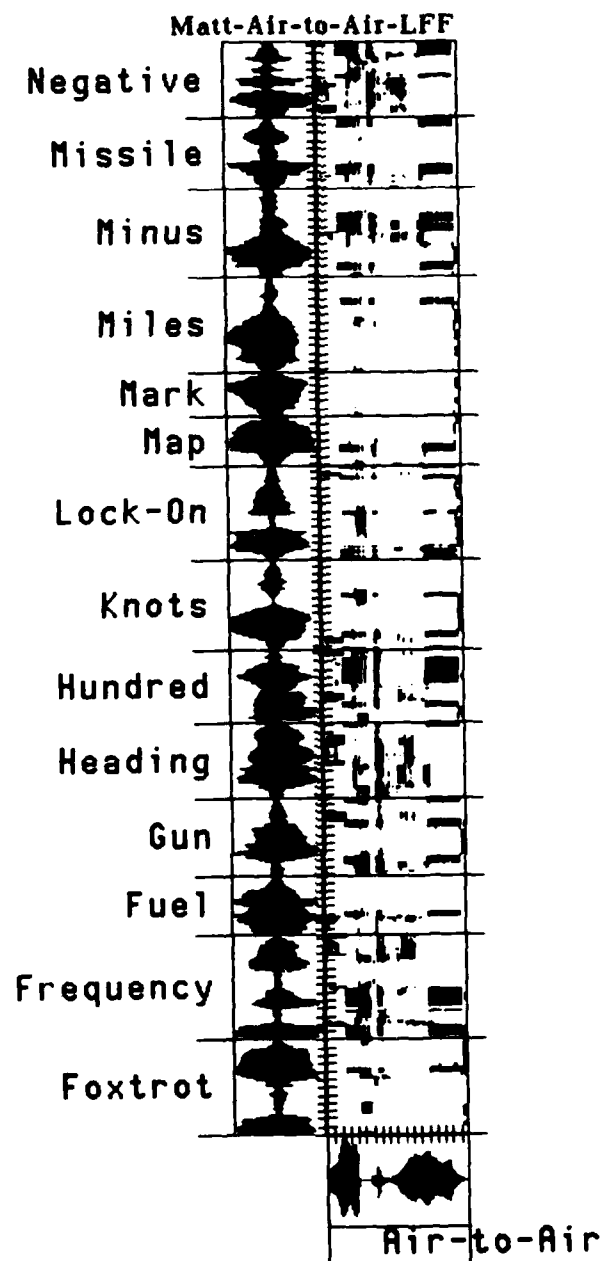


Figure B.1. Matt's Air-to-Air (Continued)

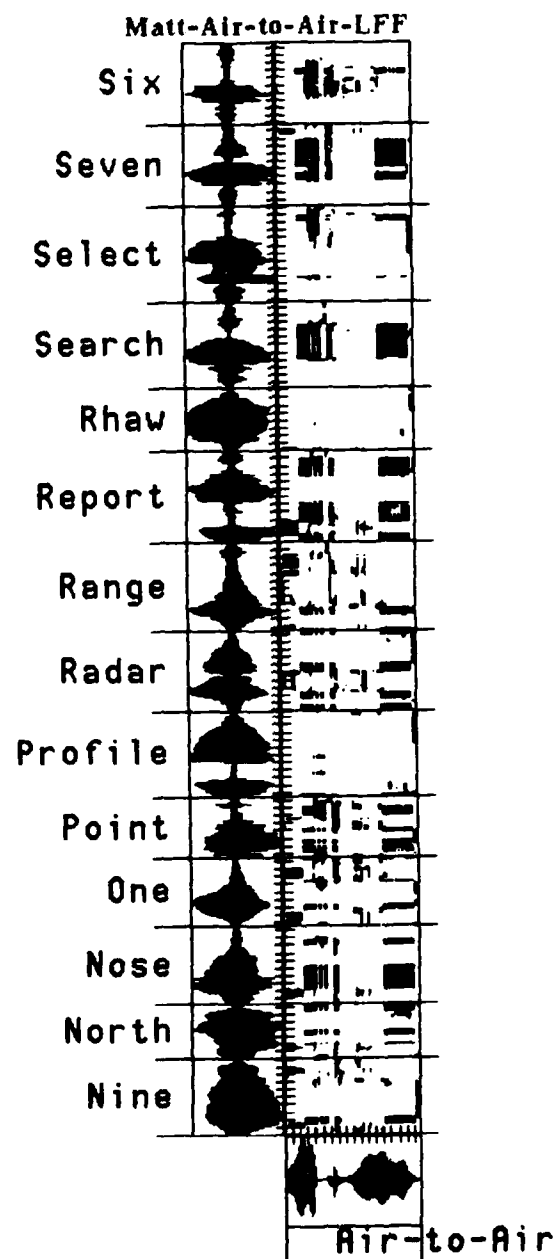


Figure B.1. Matt's Air-to-Air (Continued)

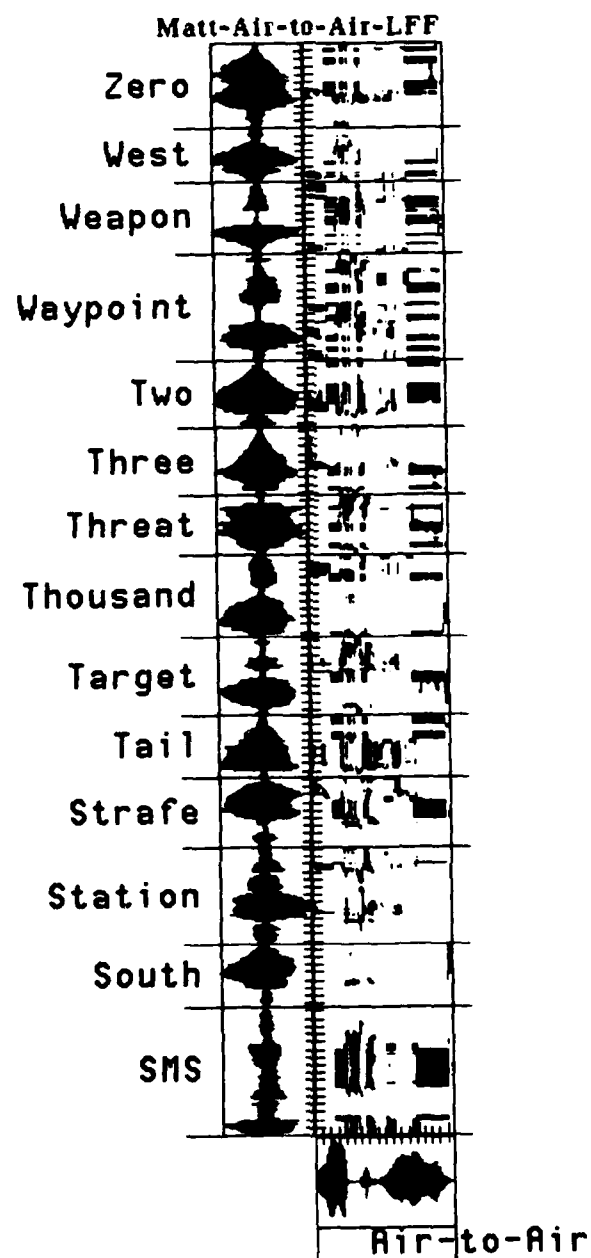


Figure B.1. Matt's Air-to-Air (Continued)

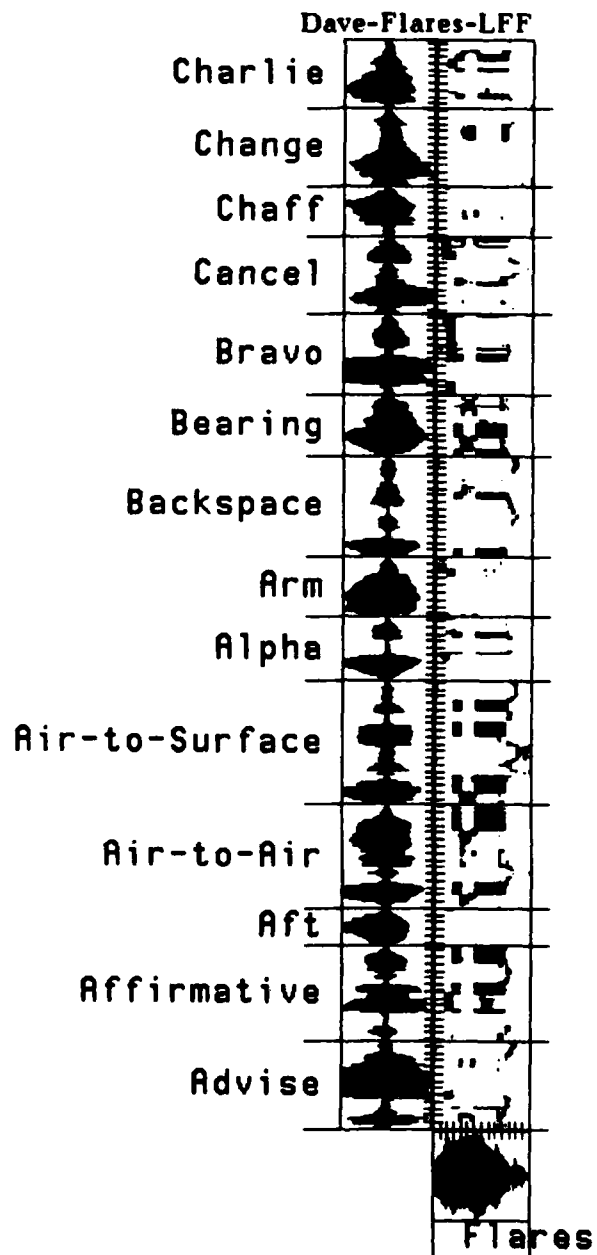


Figure B.2. Dave's Flares

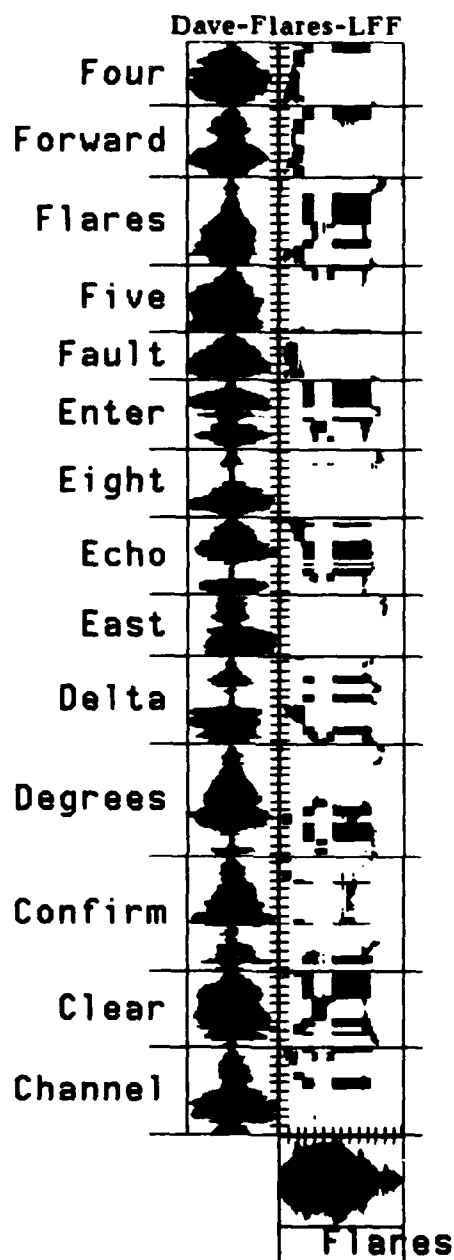


Figure B.2. Dave's Flares (Continued)

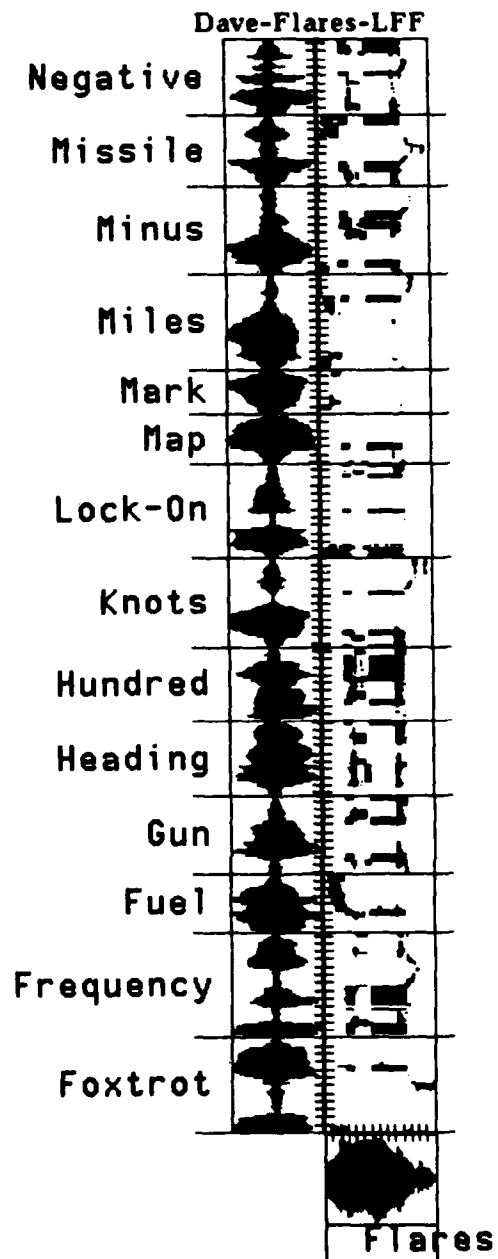


Figure B.2. Dave's Flares (Continued)

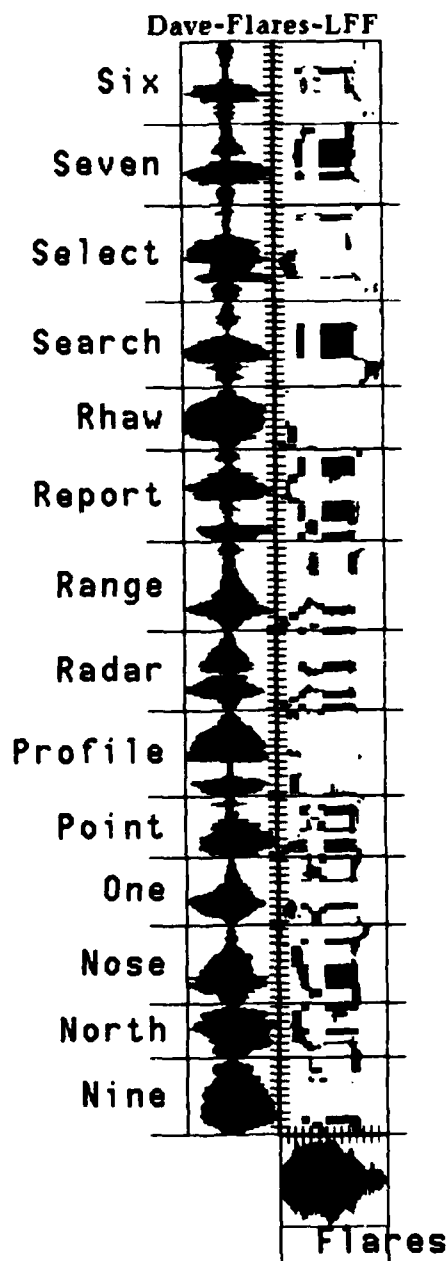


Figure B.2. Dave's Flares (Continued)

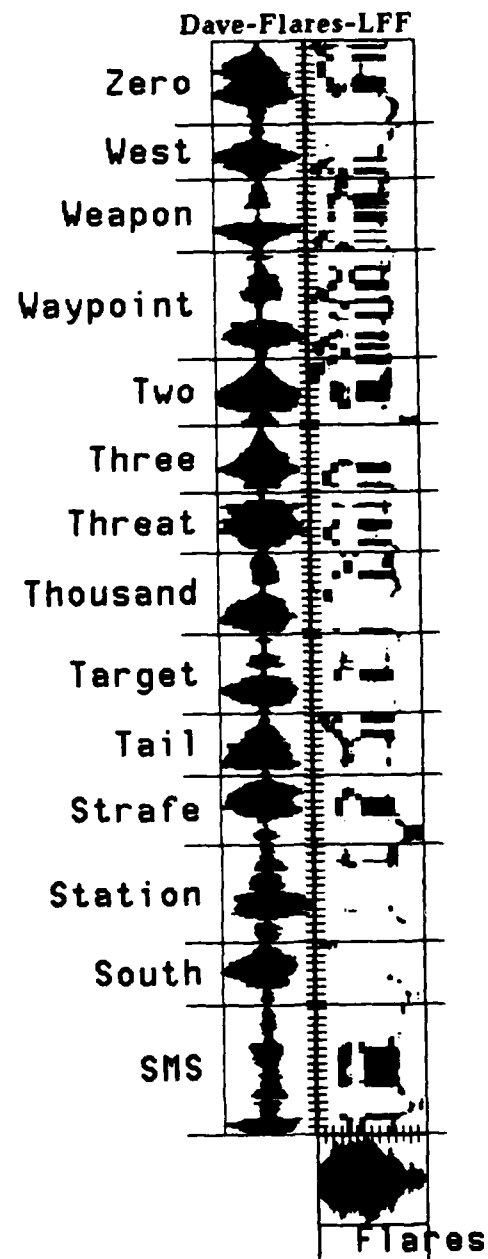


Figure B.2. Dave's Flares (Continued)

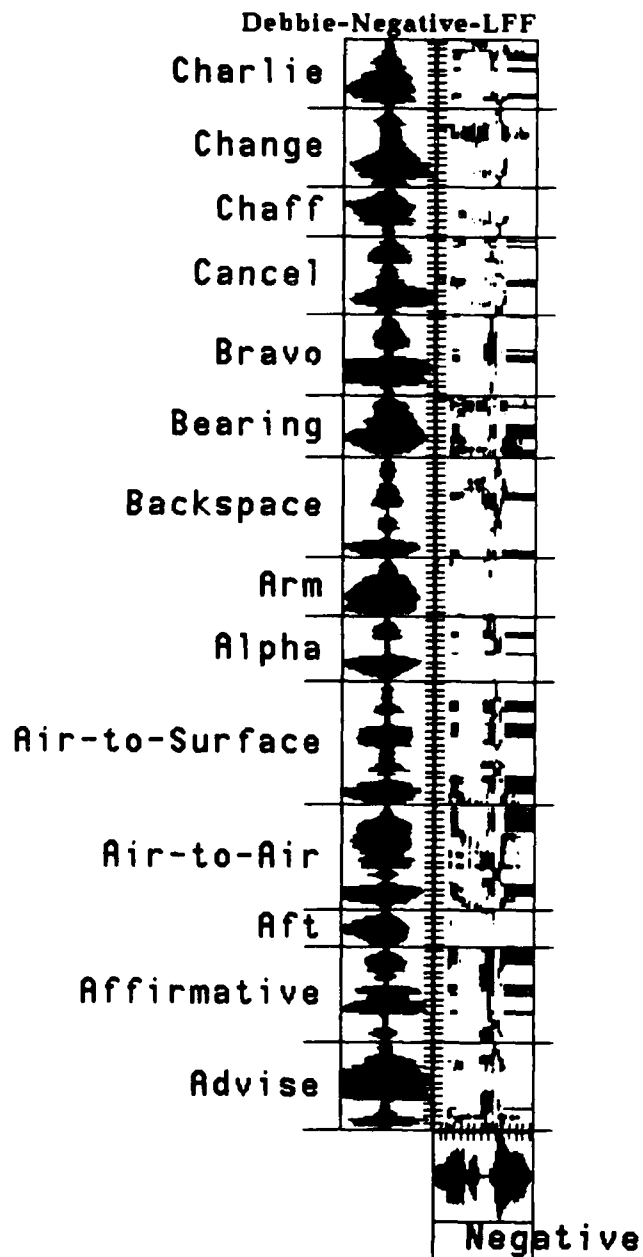


Figure B.3. Debbie's Negative

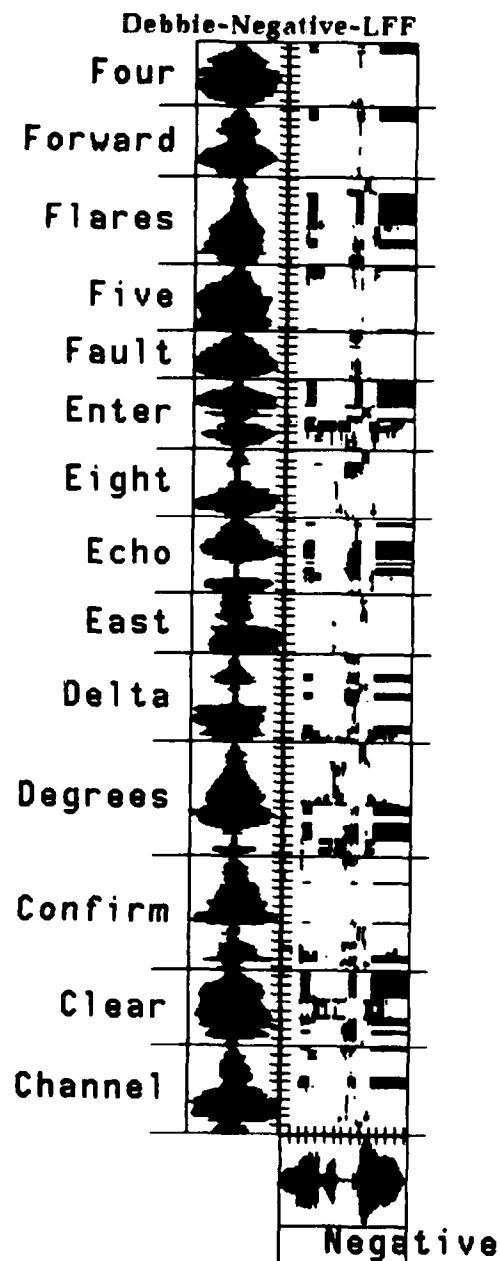


Figure B.3. Debbie's Negative (Continued)

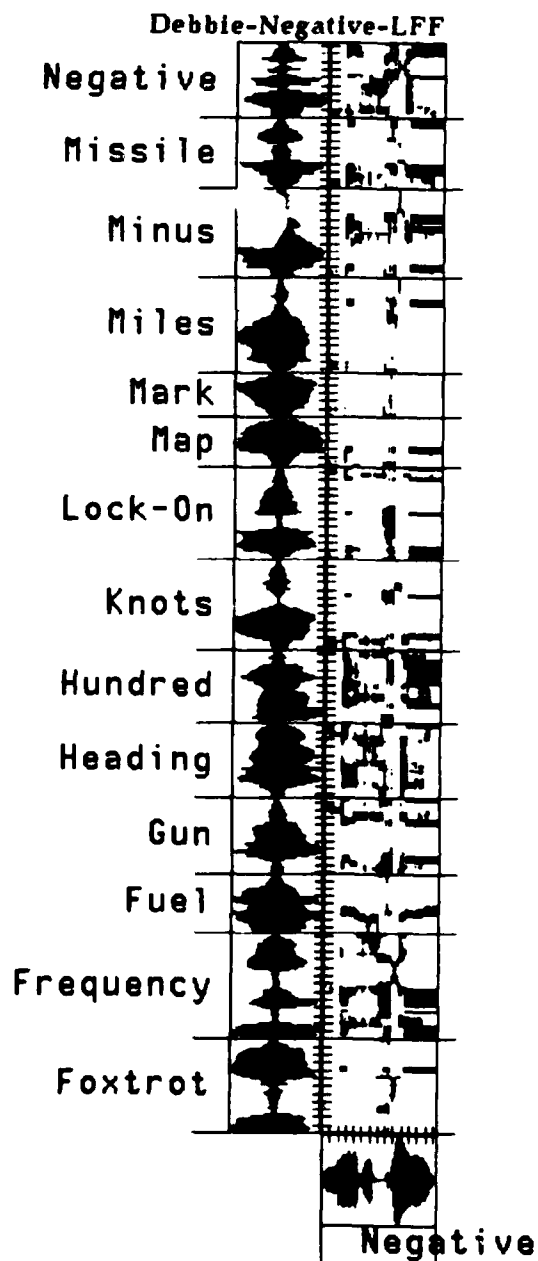


Figure B.3. Debbie's Negative (Continued)

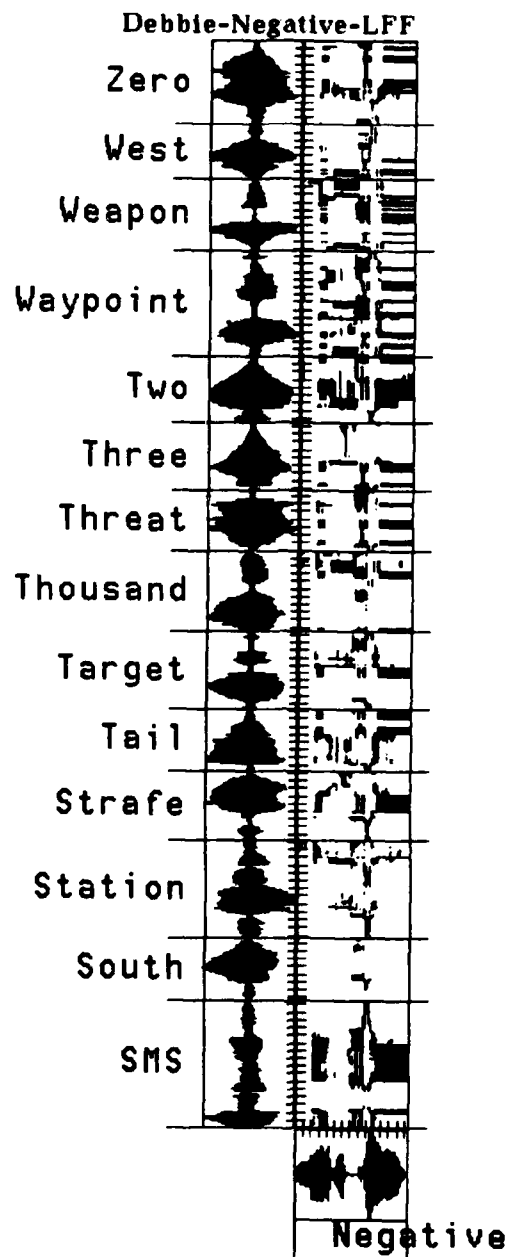


Figure B.3. Debbie's Negative (Continued)

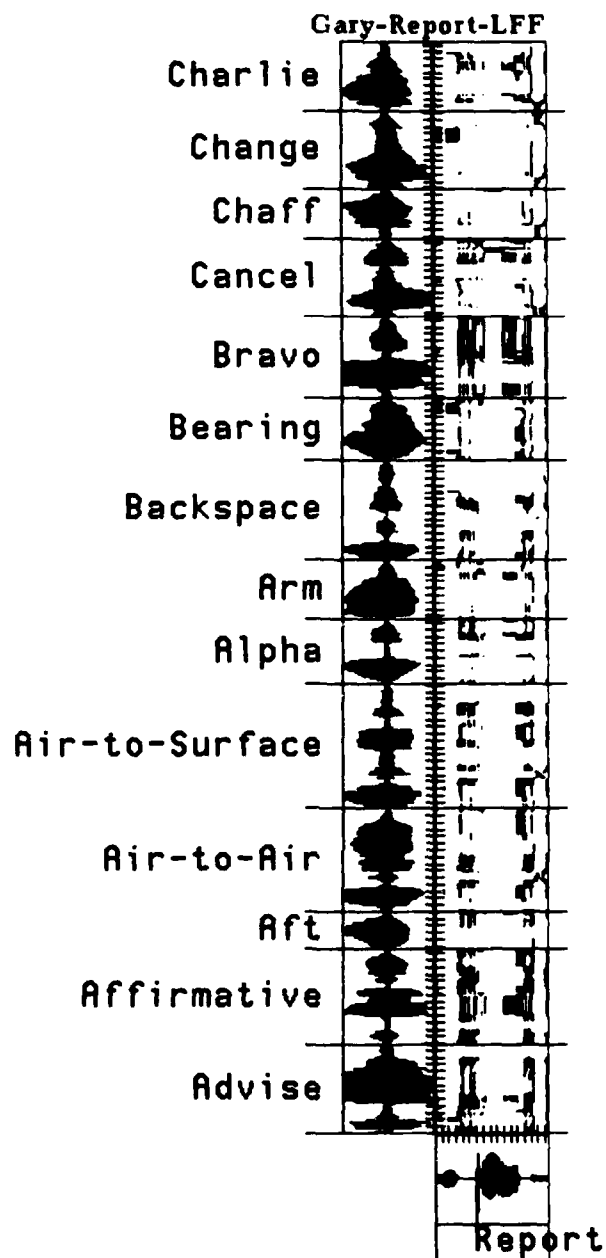


Figure B.4. Gary's Report

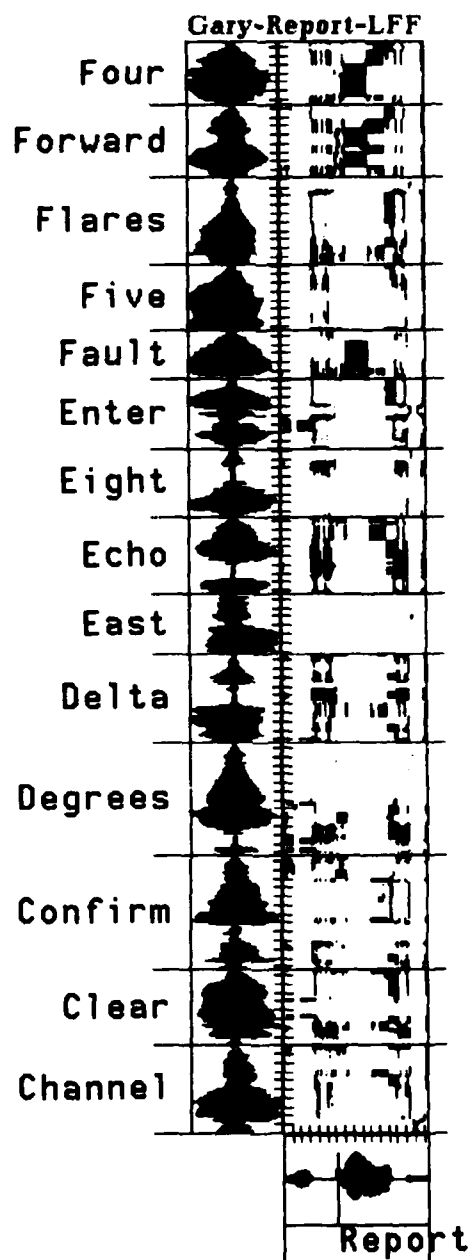


Figure B.4. Gary's Report (Continued)

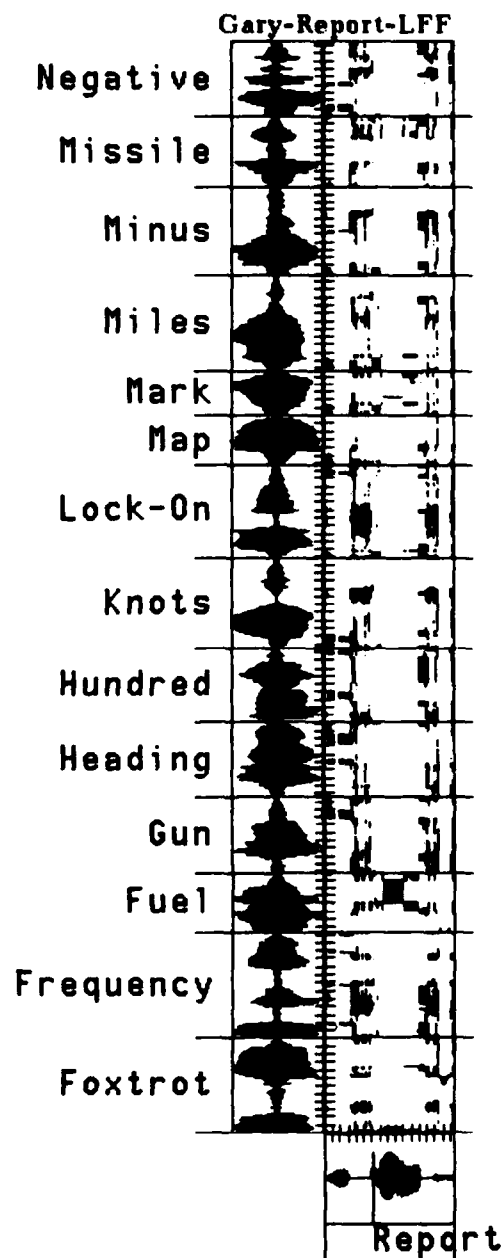


Figure B.4. Gary's Report (Continued)

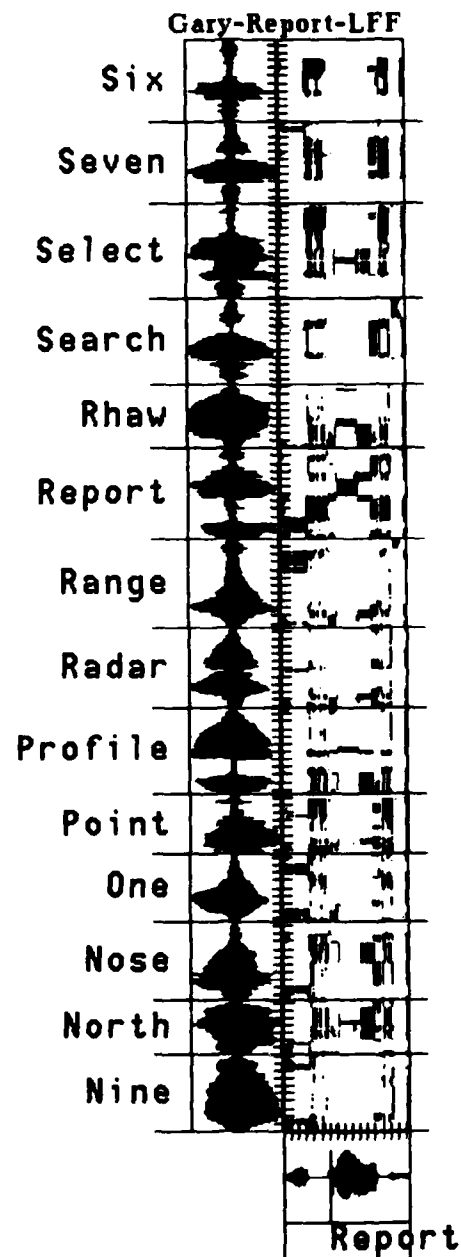


Figure B.4. Gary's Report (Continued)

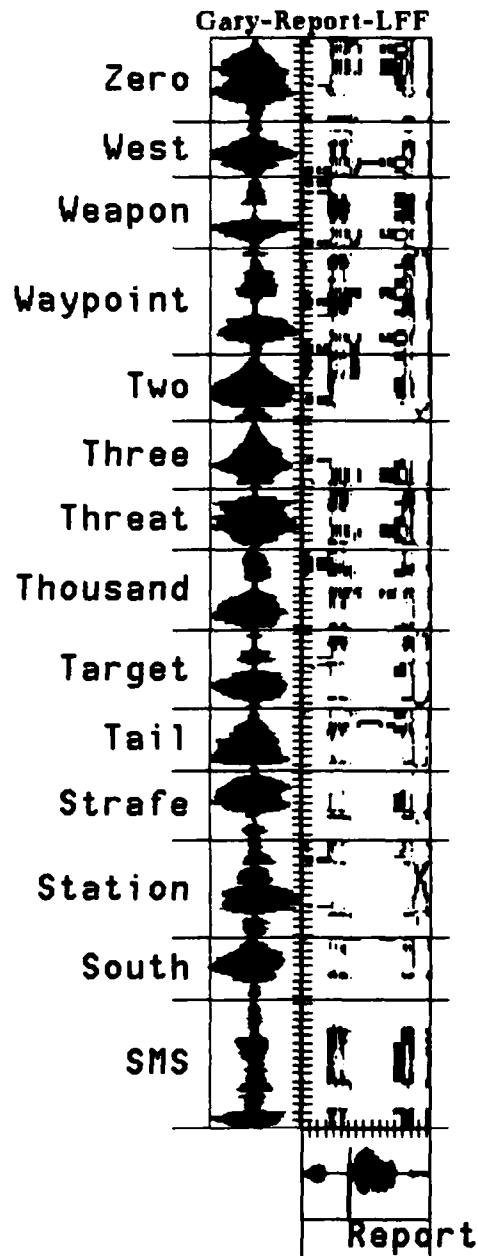


Figure B.4. Gary's Report (Continued)

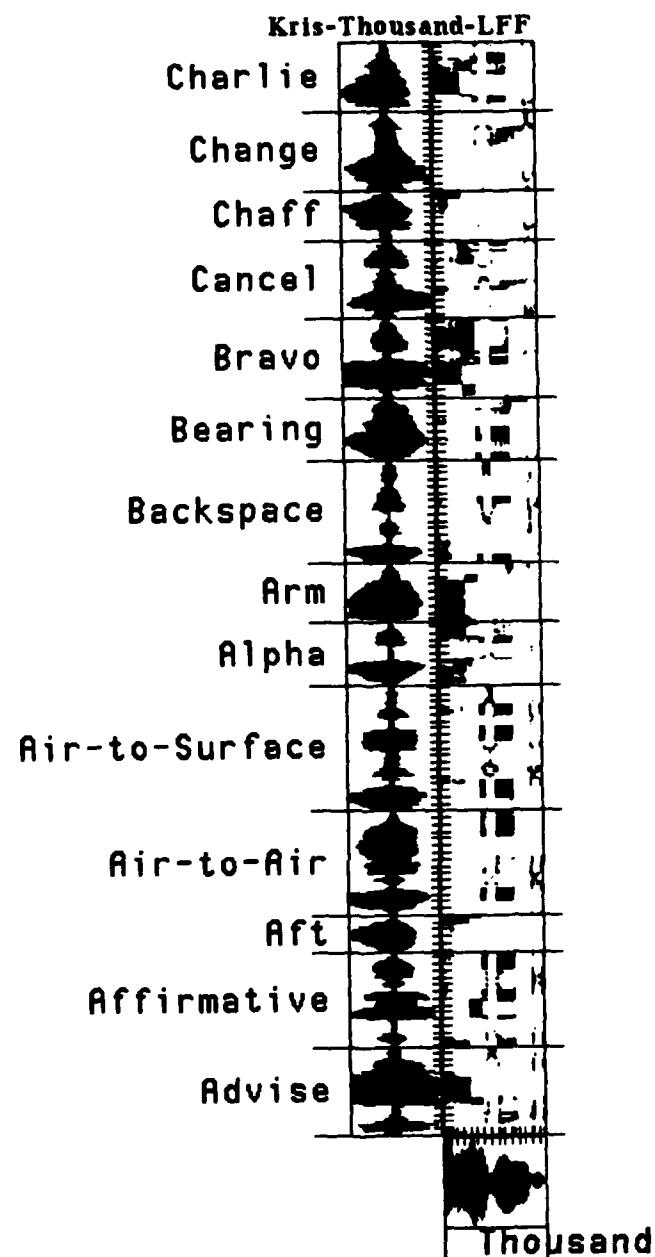


Figure B.5. Kris' Thousand

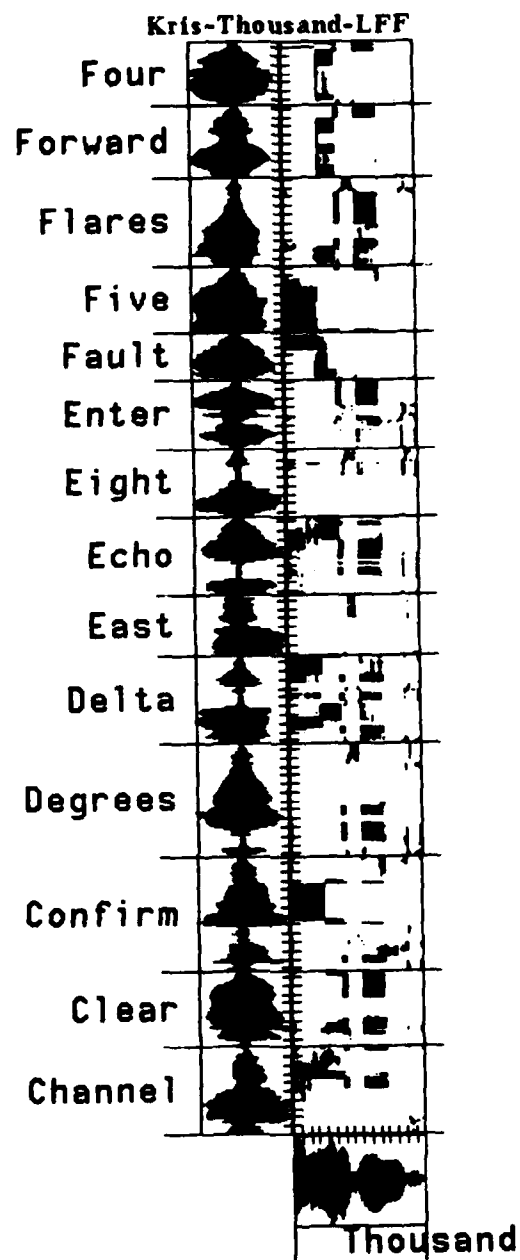


Figure B.5. Kris' Thousand (Continued)

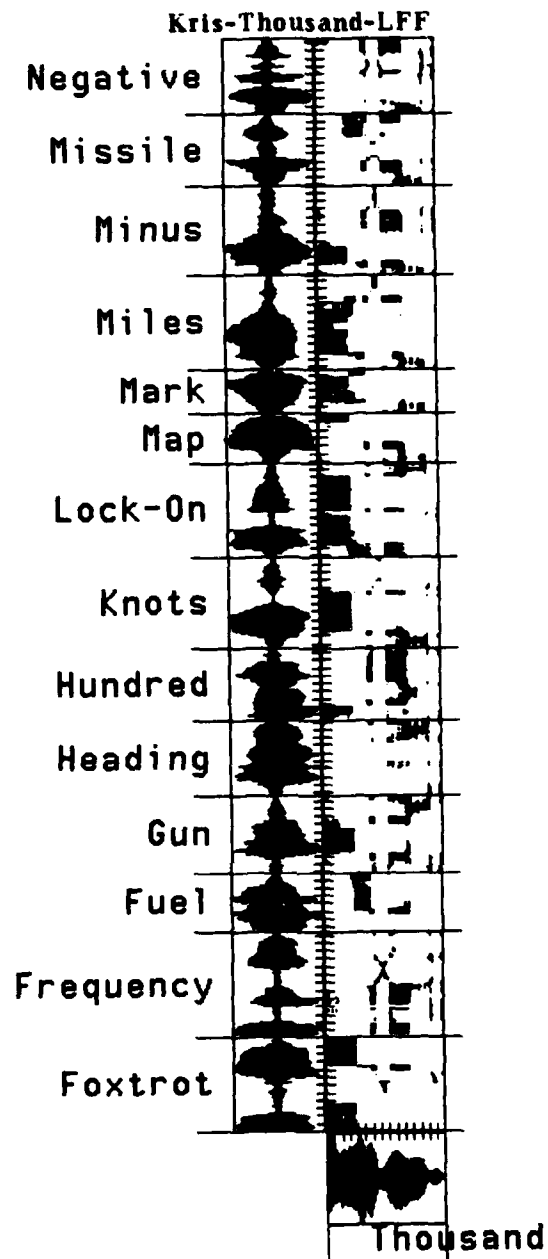


Figure B.5. Kris' Thousand (Continued)

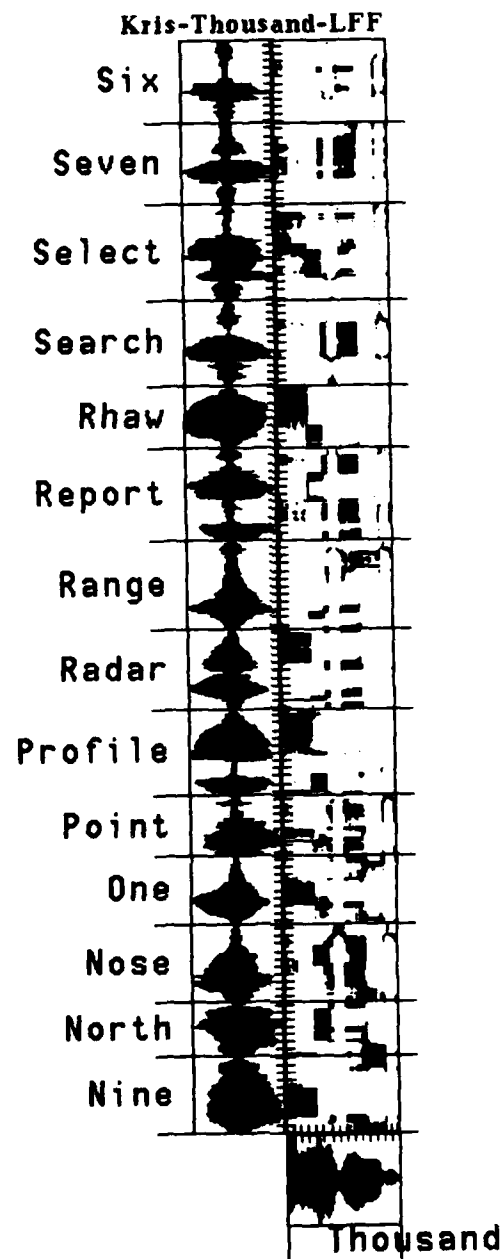


Figure B.5. Kris' Thousand (Continued)

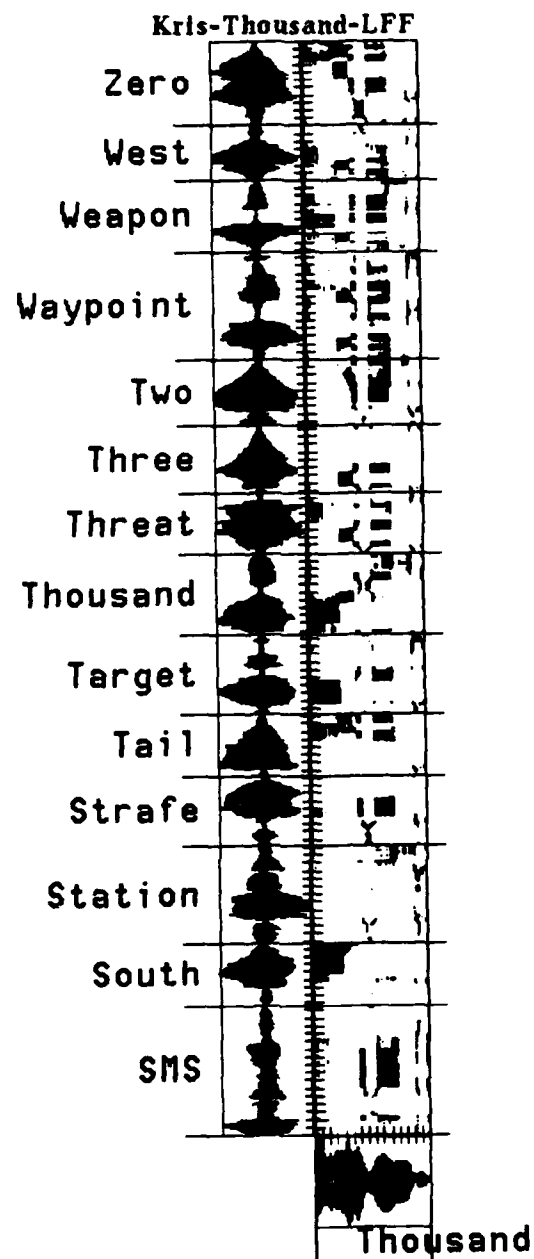


Figure B.5. Kris' Thousand (Continued)

Bibliography

1. Brusuelas, Capt Micheal A. *Investigation of Speaker-Independent Word Recognition Using Multiple Features, Decision Mechanisms, and Template Sets*. MS Thesis, GCE/EE/86D-5, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986.
2. Bursky, Dave. "Algorithm and Chips Cooperate to Squeeze More Speech Signals into Less Bandwidth," *Electronic Design*: 90-96 (October 3, 1985).
3. Dawson, Capt Robert G. *Spire Based Speaker-Independent Continuous Speech Recognition Using Mixed Feature Sets*. MS Thesis, GE/EE/87D-14. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.
4. Dixon, 1Lt Kathy R. *Implementation of a Real-Time, Interactive, Continuous Speech Recognition System*. MS Thesis, GE/EE/84D-26. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1984.
5. Doddington, George R. and Thomas B. Schalk. "Speech Recognition, Turning Theory to Practice," *IEEE Spectrum*. 18: 26-32 (September 1981).
6. Hussain, Ajmal. *Limited Continuous Speech Recognition by Phoneme Analysis*. MS Thesis, GE/EE/83D-31. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.
7. Juang, Biing-Hwang and Lawrence R. Rabiner. "Mixture Autoregressive Hidden Markov Models for Speech Signals," *IEEE Trans. Acoust., Speech, Signal Processing*. ASSP-33: 1404-1413 (December 1985).
8. Kassel, Robert H. *A User's Guide to SPIRE*. [corresponds to version 17.5] MIT Speech Recognition Group, March 1985.
9. Kauffman, David H. *SPIRE 17 Release Notes*. MIT Speech Group [supported by DARPA contract N00039-85-C-0290 monitored through Naval Electronic Systems Command], January 1986.
10. Montgomery, 2Lt Gerard J. *Isolated Word Recognition Using Fuzzy Set Theory*. MS Thesis, GE/EE/82D-74. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.
11. Ney, Hermann. "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Trans. Acoust., Speech, Signal Processing*. ASSP-32: 263-271 (April 1984).

12. Potter, R. K., George Kopp, and Harriet Green. *Visible Processing of Speech Signals*. New York: D. Van Nostrand Company, Inc., 1947.
13. Rabiner, Lawrence R. and Ronald Schafer. *Digital Processing of Speech Signals*. New Jersey: Prentice Hall, Inc., 1978.
14. Rich, Elaine. *Artificial Intelligence*. New York: McGraw-Hill, 1983.
15. Rothfeder, Jeffery. "Hardware: A Few Words about Voice Technology," *PC Magazine*. 5 : 191-205 (30 September 1986).
16. Seelandt, 2Lt Karl G. *Computer Analysis and Recognition of Phoneme Sounds in Connected Speech*. MS Thesis, GE/EE/81D-53. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.
17. *SPIRE 17.2 Reference Manual*. Speech Communications Group, Research Laboratory of Electronics, Massachusetts Institute of Technology, June 1986.
18. Vidal, Enrique, Hector M. Rulot, Francisco Casacuberta, and Jose-Miguel Bendi. "On the Use of a Metric-Space Search Algorithm (AESAs) for Fast DTW-Based Recognition of Isolated Words," *IEEE Trans. Acoust., Speech, Signal Processing*. ASSP-25: 299-309 (August 1977).
19. Winston, Patrick H. and Berthold Horn. *LISP*. (Second Edition) Massachusetts: Addison-Wesley Publishing Company, 1984.

Vita

Captain Peter Y. Kim [REDACTED] He graduated from Douglas S. Freeman High School, Richmond Virginia in 1979. He received the Bachelor of Science degree in Electrical Engineering from Virginia Military Institute in May 1983. Upon graduation he received a commission in the USAF and was assigned to the Aeronautical Systems Division (AFSC), Wright-Patterson AFB Ohio. In May 1987, Captain Kim entered the School of Engineering, Air Force Institute of Technology.

[REDACTED]

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/88D-18			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, OH 45433-6583			7b. ADDRESS (City, State, and ZIP Code)		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
					WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) (UNCLASSIFIED) F-16 Speaker-Independent Speech Recognition System Using Cockpit Commands (70 Words)					
12. PERSONAL AUTHOR(S) Peter Y. Kim, Captain, USAF					
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Year, Month, Day) December 5 1988	
15. PAGE COUNT 135					
16. SUPPLEMENTARY NOTATION <i>fr. back</i>					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Speech Recognition, SPIRE, Dynamic Programming, Multiple Feature Sets, Merged Template, LISP (List Processing Language), Data processing Thesis (SD)		
17	02				
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
Thesis Chairman : Matthew Kabrisky, PhD Professor of Electrical Engineering (see reverse) <i>Approved for release in accordance with ARJ 150-1 88Fremier 12 Jan 1989</i>					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Matthew Kabrisky, GS-15			22b. TELEPHONE (Include Area Code) (513) 255-5276		22c. OFFICE SYMBOL AFIT/ENG

Continued from block 19 : Abstract

A system is developed to achieve speaker-independent isolated speech recognition. The system uses LPC Spectrum, Formants, and Frication Frequency as a feature set. Dynamic programming is applied for distance calculations. The fundamental design concept is to create a universal template for multiple speakers. A new algorithm, which combines the vocabularies of several speakers to produce one optimal template, is incorporated into the system. An advanced speech analysis tool called SPIRE provides the computational functions required to extract appropriate features. Seventy words, from the list of F-16 cockpit commands, are selected as a vocabulary of the system. The use of a merged template based on three of the feature sets achieves an accuracy of 99 percent. The system is implemented in list programming on Symbolics 3600 series computer. *Keywords:*

FLD 18